Modular Privacy Flows: A Design Pattern for Data Minimization

CMU-HCII-22-105
September 2022

Haojian Jin
Human-Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA, USA 15213
haojian@cs.cmu.edu

Thesis Committee:
Jason I. Hong, HCII, CMU, Co-chair
Swarun Kumar, ECE & HCII, CMU, Co-chair
Yuvraj Agarwal, ISR & HCII, CMU
Laura Dabbish, HCII, CMU
Ben Y. Zhao, University of Chicago

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

Keywords: Data privacy, data overaccess, design pattern, software architecture, data minimization, human-computer interaction, ubiquitous computing, smart home, smart city, privacy engineering, mobile privacy, software engineering.

Abstract

Computing systems often allow developers to access more data than needed, violating the principle of data minimization - *a data controller should limit data collection to only what is necessary to fulfill a specific purpose*. For example, most calendar applications only allow users to grant third parties either full access to all their calendar events or none, even though third parties often only need a small portion. Conventional wisdom is to ask data controllers to offer many fine-grained data accesses for every potential use case. However, this would lead to an explosion of accesses that would be onerous for system builders to implement, unwieldy for users to configure, and complex for developers to learn.

This dissertation introduces a new design pattern, called *Modular Privacy Flows* (MPF), for designing systems that allow developers to collect data on a need-to-know basis. MPF combines three simple ideas. First, instead of enumerating all-or-nothing fine-grained data access, system builders offer a small and fixed set of stateless **operators** to developers. Second, developers declare intended data access by authoring a Unix-like pipeline using these operators and save the pipeline representation in a text-based **manifest**. Third, given a manifest, a trusted **runtime** assembles a data transformation executable using pre-loaded open-source operator implementations, which relays data flows in a structured and enforceable manner.

MPF offers a few important advantages over the conventional all-or-nothing permission approach and other relevant approaches. First, system builders can now support numerous fine-grained APIs by implementing a small set of reusable operator implementations. Second, developers only need to learn the semantics of a few operators to customize their data access. Third, since the operators have clearly defined semantics and the manifests are non-proprietary, MPF can facilitate many independent privacy features to help users manage their privacy in a centralized and unified manner. Further, MPF also allows third-party privacy advocates (e.g., consumer reports) to analyze manifests programmatically and alert users of bad practices.

This dissertation has three main parts. The first part includes three empirical studies to characterize developers' data collection behaviors, illustrating that most developers only need partial or derived data rather than raw data. The second part introduces two MPF software architectures (Peekaboo and MapAggregate) that can reduce developers' data collection and demonstrate the abovementioned advantages. Finally, the third part presents two design methods to help developers navigate data minimization's design space, including data collection decision-making and designing independent privacy features through MPF. Combined, this dissertation will scaffold the future development of data minimization.

To all my family, to all my friends:
the journey has been long,
but made all the more worthwhile because of all of you.

Acknowledgments

I would not be where I am now without the help of my mentors, collaborators, friends, and family. They have enabled my work, made my Ph.D. path extremely rewarding, and helped develop me into a better person.

I owe a great deal to my advisers, Jason Hong and Swarun Kumar. Jason took me on as a student since I started my Ph.D. in HCII. He has been a crucial part of my growth and development. In retrospect, I might have acted like a stubborn teenager in research in the first few years. Thank you for being so patient in discussing all the crazy ideas with me and guiding me to discover the final direction outlined in this dissertation. I am also fortunate to find Swarun as my co-advisor. Swarun mentored me first in a few side projects and later became my formal co-advisor. Thank you for working through so many paper deadlines with me. The few 8-AM deadlines were the most relaxed and enjoyable moments of my Ph.D. life. And thank you for always being there and teaching me to articulate my thinking. I couldn't have done this dissertation and Ph.D. without you.

I want to thank my pseudo-advisor, Yuvraj Agarwal, with whom I have collaborated since my first year at CMU. Thank you for always taking the time to listen to my ideas and help me refine them. I am grateful for the opportunity of working with you and learning about research, life, and academics from you. I also thank my committee members, Laura Dabbish, Ben Y. Zhao, for their feedback and inspiration. I thank Gram Liu and David Hwang, who assisted me in implementing Peekaboo's crucial operator components.

I would like to thank my mentors, collaborators, and friends from the CHIMPS Lab, WiTech Lab, Synergy Lab, HCII, CMU, and the broader research community. In particular, I would like to thank Atul Bansal, Kareem Bedri, Sudershan Boovaraghavan, Alex Cabrera, Xiang 'Anthony' Chen, Fanglin Chen, Tianying Chen, Sauvik Das, Cori Faklaris, Akshay Gadre, Anhong Guo, Boyuan Guo, Queenie Kravitz, Toby Li, Tianshi Li, Xieyang 'Micheal' Liu, Hong Shen, Prasoon Patidar, Akarsh Prabhakara, Rituparna Roychoudhury, Sophie Sacks, Sai Ganesh Swaminathan, Vaibhav Singh, Yiwen Song, Xu Wang, Jingxian Wang, Koji Yatani, Yaxing Yao, Kuang Yuan, Zheng Yao, Siyan Zhao, Han Zhang, Junbo Zhang, Yang Zhang, Diana Zhang.

I also thank my family for their love and support. I am forever grateful to my parents, Zhonghui Jin and Haiying Mo, for your unconditional love and limitless support. And thank you to my girlfriend, Yulan Feng, who has provided both emotional and intellectual support. She proposed the name "Peekaboo" for the main project in this dissertation.

It has been a long journey that I did not anticipate when I began. I moved to Pittsburgh on August 21, 2016, and left on September 13, 2022. Other than attending conferences, I rarely traveled outside of Pittsburgh. I spent most of my time reading, writing, and coding at Newell

Simon Hall. The work I've written in this document is the culmination of six years of learning and discussion with my mentors, collaborators, friends, and colleagues. While the past six years have been challenging, they have also been incredibly enjoyable and enlightening.T

"This chapter of my Ph.D. life is coming to a wistful close, but I have many more snowfalls to see, many more friends to meet, and many more dreams to chase. I'm fortunate to have all of you to share it with."

Publications

Chapter 2 revises a previous publication [183]: Haojian Jin, Minyi Liu, Kevan Dodhia, Yuanchun Li, Gaurav Srivastava, Matthew Fredrikson, Yuvraj Agarwal, and Jason I. Hong. "Why are they collecting my data? inferring the purposes of network traffic in mobile apps." Proc. ACM IMWUT (Ubicomp), 2018.

Chapter 3 & Chapter 5 revises a previous publication [181]: Haojian Jin, Gram Liu, David Hwang, Swarun Kumar, Yuvraj Agarwal, and Jason Hong. "Peekaboo: A Hub-Based Approach to Enable Transparency in Data Processing within Smart Homes." Proc. IEEE S&P (Oakland), 2022.

Chapter 4 & Chapter 6 revises an in-submission manuscript: Haojian Jin, Gram Liu, Prasoon Patidar, David Hwang, Sophie Sacks, Swarun Kumar, Yuvraj Agarwal, and Jason Hong. "MapAggregate: Putting Data Control Back into Users' Hands through Structured and Isolated Data Aggregation." In submission.

Chapter 7 revises a previous publication [186]: Haojian Jin, Hong Shen, Mayank Jain, Swarun Kumar, and Jason I. Hong. "Lean privacy review: Collecting users' privacy concerns of data practices at a low cost." ACM TOCHI, 2021.

Chapter 8 revises a previous publication [187]: Haojian Jin, Boyuan Guo, Rituparna Roychoudhury, Yaxing Yao, Swarun Kumar, Yuvraj Agarwal, and Jason I. Hong. "Exploring the Needs of Users for Supporting Privacy-Protective Behaviors in Smart Homes." Proc. ACM SIGCHI, 2022.

Contents

Chapter 1

Introduction

Principle of data minimization - General Data Protection Regulation (GDPR), Article 5 (1) (c): *"Personal data shall be adequate, relevant, and limited to **what is necessary** in relation to the **purposes** for which they are processed."*

Data minimization is one of the most influential and straightforward principles in mitigating privacy risks. Many current privacy regulations, such as General Data Protection Regulation (GDPR), California Privacy Rights Act (CCPA), and Fair Information Practice Principles (FIPPs), have adopted it. Yet, engineering this high-level principle into actual systems is surprisingly complex [147]. Developers often struggle to translate this high-level principle into technical requirements [72, 148, 294, 333]. Further, they have to build everything around data minimization from scratch [181, 333], requiring significant resources and resulting in inconsistent interfaces for end-users to manage.

An essential part of implementing data minimization is knowing what data can be minimized. We performed three empirical studies to characterize developers' data collection behaviors, illustrating that most developers do not need raw data but rather a transformed version. For example, imagine a "HelloVisitor" app installed on a video doorbell that identifies visitors using faces in images. Today's video doorbells often send captured raw photos to the cloud when they detect a scene change. However, by applying a pre-processing pipeline (i.e., face detection and image cropping), HelloVisitor can avoid sending images of people or vehicles simply passing by, thus reducing data egress to just what the app needs to operate.

We then introduce a new **generic** design pattern, called Modular Privacy Flows (MPF), for designing systems in allowing developers to collect data on a need-to-know basis. At the high level, MPF wants to replace the conventional all-or-nothing data access with operator-based APIs. Most systems today often use a set of all-or-nothing APIs (or permissions) to govern user data access. However, this approach can hardly support fine-grained data accesses, such as face-only images and user identities, since it would be onerous for system builders to implement, unwieldy for users to configure, and complex for developers to learn.

Instead, MPF proposes that future systems may offer a small set of reusable stateless operators and allow developers to connect these operators into data transformation pipelines (similar to Unix Pipes) to customize desired data access.

In doing so, system builders can now support numerous fine-grained APIs by implementing a small set of reusable operator implementations. Developers only need to learn the semantics of a few operators to customize their data access. Since the operators have clearly defined semantics and the manifests are non-proprietary, MPF can facilitate many independent privacy features to help users manage their privacy in a centralized and unified manner. Further, MPF also allows third-party privacy advocates (e.g., consumer reports) to analyze manifests programmatically and alert users of bad practices.

MPF shifts the data collection from a black box, late-binding process to a transparent, modular, early-binding process. Traditionally, developers collect users' raw data to their proprietary servers, declare how they will use them in comprehensive and vague privacy policies, and then analyze these data in a black box for users and auditors. In contrast, MPF asks developers to declare how they want to use users' data first, then reflect on what data they actually need, and use the operators to compose desired APIs.

This shift has two critical implications. First, developers must decide what fine-grained data they should collect and how to use it. Making the process transparent implies that developers are more likely to receive backlash over imprudent data collection. Second, asking users to approve all individual data requests manually is no longer necessary. The modular nature allows system builders and third parties to build privacy management features to alleviate users' burdens. Towards these two implications, we introduce two design methods to help developers navigate data minimization's design space, including data collection decision-making and designing independent privacy features through MPF protocols.

Thesis statement: This thesis contends that most developers do not need raw data but rather a transformed version and introduces a design pattern called Modular Privacy Flows (MPF) to implement it. Software architectures based on MPF can help minimize data collection by app developers, while also making it easier for users and auditors to verify and interact with developers' data collection behaviors.

1.1 A Data Minimization Example

This section uses a calendar application as an example to illustrate the problems around existing solutions. For example, Google calendar is a popular calendar service provider which offers six types of data access for third parties (Fig. 1.1). Zoom, a popular video conferencing application, now wants to access users' calendar data, detect the Zoom join links in users' calendar events, and offer users an easy way to start or join these Zoom meetings [407]. In the conventional paradigm, developers at Zoom would request the permission of "read/write access to calendars". Google then shows users a consent interface (Fig. 1.2) and asks them to grant the requested data access to Zoom. Many other applications adopt similar privacy

paradigms, such as iOS/Android apps, smart home applications, browser extensions, and social network apps [139].

Scope	Meaning
`https://www.googleapis.com/auth/calendar`	read/write access to Calendars
`https://www.googleapis.com/auth/calendar.readonly`	read-only access to Calendars
`https://www.googleapis.com/auth/calendar.events`	read/write access to Events
`https://www.googleapis.com/auth/calendar.events.readonly`	read-only access to Events
`https://www.googleapis.com/auth/calendar.settings.readonly`	read-only access to Settings
`https://www.googleapis.com/auth/calendar.addons.execute`	run as a Calendar add-on

Figure 1.1: Google calendar offers six types of data access for third parties to access users' calendar data information. The design of read/write/execute permissions is possibly inspired by the Unix file permissions.

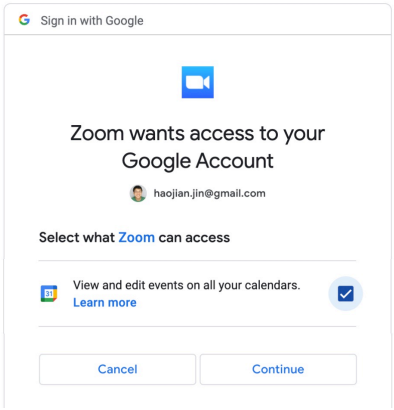


Figure 1.2: If the user consents, Google would allow Zoom to access all her calendar data.

This paradigm comes with two important drawbacks. First, Zoom needs only a small portion of calendar events containing a zoom join link in the event description, but Google requires users to grant Zoom full access (Fig. 1.2) to all events. A natural question emerges from this: "why not ask Google to offer more fine-grained APIs, such as read access to only Zoom events"? The issue is that systems like Google calendar need to support numerous applications, each of which may require some niche APIs. For example, a productivity tracking application, interested in knowing how busy the user is, may want to request the number of daily business meetings. Enumerating these fine-grained accesses would lead to an explosion of accesses that would be onerous for system builders (e.g., Google) to implement, unwieldy for users to configure, and complex for third parties (e.g., Zoom developers) to learn [181].

Second, users often do not have the expertise and time bandwidth to make informed decisions based on these consent interfaces [246]. Prior research shows that people are demonstrably bad at assessing the risk of future harms that may flow from the piecemeal collection of their private data [66]. Further, this

interface puts users in an all-or-nothing situation, forcing users to accept the request if they want to use the service [103].

1.2 Modular Privacy Flows (MPF)

We argue that the conventional design of all-or-nothing binary permissions (Fig. 1.1), initially designed for security management, has become increasingly languid in managing privacy. First, the binary permission approach works best with a small number of options (e.g., Unix file permissions). But, managing privacy requires many fine-grained data access to capture the nuanced data collection/use contexts (e.g., what data is being collected and why, how the data is being used). Second, data analysis can subvert these binary permission systems easily. For example, Zoom can potentially use calendar data to infer many insights about the user, such as income and education. Instead of always offering raw data, systems should seek to process the raw data into less privacy-intrusive forms to reduce these unintentional information leaks.

Modular Privacy Flows replaces these all-or-nothing permissions with data collection manifests, which has three key ideas. First, developers must declare all intended data collection behaviors in a text-based **manifest** (Fig. 1.3). Second, to specify the data collection, developers choose from a small and fixed set of open-source, chainable **operators** with well-defined data transformation semantics, authoring a stream-oriented pipeline similar to Unix pipes. Third, a trusted **runtime** enforces the declared behaviors in the manifest, by running all of the pre-loaded, open-source operators specified in the manifest (Fig. 1.4).

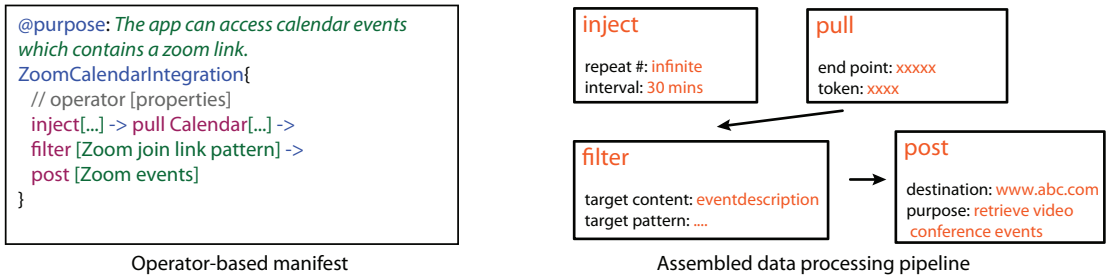


Figure 1.3: Zoom developers can export the data processing pipeline as a text-based manifest (left) and upload it to Google. Given a manifest, Google then assembles and executes a pre-processing pipeline (right) using operators pre-loaded on the cloud, processing users' calendar data before sending it to Zoom.

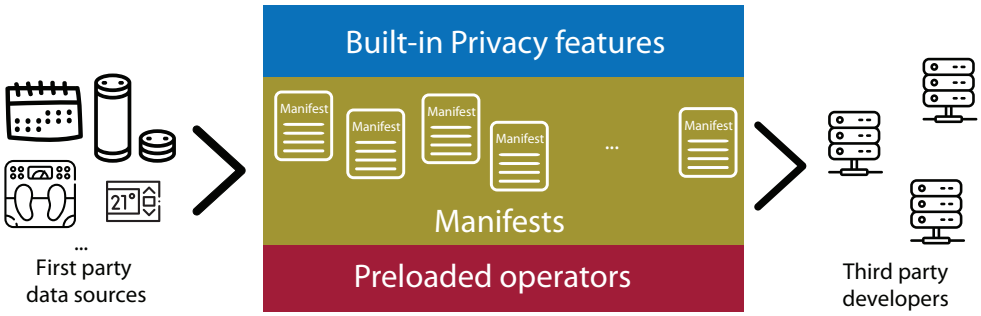


Figure 1.4: The runtime (middle) assembles data transformation pipelines using pre-loaded operators and processes users' data before sending it to third parties. In doing so, developers can collect users' data on a need-to-know basis; third-party auditors can verify data collection behaviors; and the runtime itself can offer several centralized privacy features to users across applications without additional effort from app developers.

Combined, developers can reduce data collection by running data transformation tasks before accessing users' data. System builders can implement fewer APIs by reusing a small set of operators. This design also makes the API easier for developers to learn. In addition, since each operator's semantics is known and the manifest is public, outsiders (e.g., regulators and privacy advocates) can quickly analyze developers' data collection behaviors.

Further, MPF can enable many independent end-user privacy features without support from app developers. For example, a future calendar application may allow users to noisify the data provided to developers by inserting additional data transformations into the pipelines specified by developers. Or the calendar application can analyze the manifest and generate natural language statements to make it easier for lay people to understand what data will be sent out, when, and to where. Finally, if services/apps/devices share the same MPF protocol, we may reuse these privacy features, enabling a unified and centralized privacy management interface for end-users.

Alternative solutions: Besides the fine-grained permission approach and MPF, there are two potential alternatives: GraphQL and remote code execution. Table. 1.1 compares MPF with these three approaches. In the database approach, the system may allow developers to author SQL-like queries (e.g., GraphQL) and create a SQL database engine to interpret the declarative queries. MPF has two important advantages over this design: flexibility and auditability. First, it is more flexible for the system builder to extend supported data transformations by adding/removing/updating operator implementations, while modifying a database engine is non-trivial for most organizations. Second, the pipeline approach is more verifiable than the database approach. The runtime simply connects data transformation functions into a pipeline. Outsiders can verify the runtime by verifying each operator's pre-loaded implementation (open-sourced). In contrast,

the complexity of a database engine makes it hard to analyze and verify the engine's behavior.

Another alternative design is that system builders may allow developers to upload code to transform data (e.g., [81]). While this approach offers great flexibility for adding new data accesses, it is hard to enforce (i.e., analyze and control) the data transformation behaviors of these arbitrary programs. In contrast, MPF only allows developers to specify data collection behaviors using a set of high-level operators with well-defined semantics. So MPF can easily infer the program behaviors by analyzing the manifest and help users control how their data would be collected by stopping undesired data flows.

Table 1.1: Tradeoffs of different data minimization implementations

	Fine-grained permission	MPF	Database (e.g., GraphQL)	Remote code execution
Flexible granularity	✓	✓	x	✓
Easy development	x	✓	✓	x
Enforceable collection	✓	✓	✓	x
Auditability	✓	✓	x	x

1.3 Thesis Outline

This dissertation introduces a family of empirical studies, system architectures, and design methods around Modular Privacy Flows. Table. 1.2 outlines the main idea of each chapter.

1.3.1 Understanding data collection behaviors

Data minimization, as a legal concept, is defined vaguely [148, 294]. While legal scholars and philosophers have proposed taxonomies of privacy violations [339] and a holistic framework for evaluating appropriate flows of information based on contextual social norms [271], engineers still struggle to translate this high-level principle into technical requirements [333].

We first built MobiPurpose (Chapter 2), a large-scale mobile app network analysis system that can classify the data collection purposes of network requests made by an Android app using a combination of supervised learning and text pattern bootstrapping. We used MobiPurpose to intercept and analyze over 2 million unique traffic requests from 15k apps. We found that fine-grained data collection purposes often only require transformed data rather than raw data.

We then use this insight to explore the potential privacy-sensitive implementations for 200+ smart home (Chapter 3) and 80+ smart city applications (Chapter 4). Our analysis suggests that (1) most cloud services do not need raw data; (2) most data transformations share similar data-agnostic data actions; (3) many of the

Table 1.2: The outline of this dissertation.

Part 1: **Understanding** data collection behaviors	
Chapter 2	We introduce a framework to inspect developers' data collection behaviors. The framework contends that fine-grained data collection purposes often only require transformed data rather than raw data.
Chapter 3 & 4	We use the framework in Chapter 2 to examine the data minimization opportunities in 200+ smart home applications and 80 smart city applications.
Part 2: **Implementing** data minimization	
Chapter 5	We introduce a new MPF software architecture for smart homes that leverages an in-home hub to pre-process and minimize outgoing data in a structured and enforceable manner before sending it to external cloud servers.
Chapter 6	We introduce a new MPF software architecture that only allows developers to query insights, on data aggregated across multiple homes, in a manner that is controlled and configured by homeowners in the interest of privacy.
Part 3: Navigating the **design** space	
Chapter 7	We introduce a fast, cheap, and easy-to-access design method to help developers decide what fine-grained data they should collect and how to use it appropriately.
Chapter 8	We introduce a low-cost design method for developers to prioritize the independent features that can be implemented through MPF and identify the critical design factors.

data transformations are lightweight and non-proprietary. These findings motivate and inform the design of MPF.

1.3.2 Implementing Data Minimization

Conventionally, implementing data minimization requires significant expertise since developers must work with multiple stakeholders.

Development burden. It often takes developers a great deal of effort to incorporate data minimization into their systems. Developers must implement edge data transformation programs, privacy management interfaces, and fine-grained server APIs. Further, another constraint is that if developers only collect necessary data, it implies that developers need to constantly update the edge programs if they have a new data collection need.

User management overhead. If all developers overcome the development overhead and build data minimization individually, users would have to face distributed management interfaces with inconsistent seman-

tics. For example, when a user may choose to "mute" a smart speaker, there can be multiple interpretations across different products, such as powering off, stopping always listening, and halting the current conversation sessions. Further, as the number scales, users can scarcely manage across devices/services.

Limited transparency. Finally, how can developers assure users that they will only collect necessary data? For example, how can a video doorbell developer assure users that they only collect the face images and use these face images in a visitor album? In contrast to security access, privacy-related code snippets are often scattered across the programs and tangled with application logic. To audit the actual data practice, developers have to either open source their programs or invite third-party auditors to study their codebase privately. Neither approach is practical, as the former breaks most business models, and the latter is expensive and slow. Unfortunately, even if we have access to all the source code, verifying the actual code behavior from the arbitrary code is a daunting task.

The goal of Modular Privacy Flows is to offer a systematic and reusable design pattern to replace ad-hocism around data minimization, scaffolding the process of developing privacy-sensitive systems. We introduce two software architectures, Peekaboo and MapAggregate, illustrating two examples of implementing MPF (i.e., "Operators, Manifest, and Runtime") (Table 1.3).

Peekaboo (Chapter 5) is a new privacy-sensitive architecture for smart homes that leverages an in-home hub to pre-process and minimize outgoing data in a structured and enforceable manner before sending it to external cloud servers. Peekaboo contributes an example operator design for implementing data pre-processing and designs an example runtime using an in-home hub. Peekaboo also demonstrates that developers can collect smart home data on a need-to-know basis; third-party auditors can verify data collection behaviors; and the hub itself can offer a number of centralized privacy features to users across apps and devices, without additional effort from app developers.

MapAggregate (Chapter 6) is a first-of-its-kind software architecture that only allows developers to query insights, on data aggregated across multiple homes, in a manner that is controlled and configured by homeowners in the interest of privacy. MapAggregate contributes an example operator design for implementing data aggregation and designs an example runtime using serverless cloud infrastructure. Modular Privacy Flows is unique in that users can sign up for contributing data only after developers provide an explicit and enforceable plan on how they will aggregate user data.

Table 1.3: Operators, Manifest, and Runtime in Peekaboo and MapAggregate.

	Peekaboo	MapAggregate
Operators	A small and **fixed** set of chainable operators that abstract common **data pre-processing** functionalities	a small set of chainable operators that abstract common **data aggregation** functionalities
Manifest	A text-based manifest that declares all intended data collection behaviors, including under what conditions data will be sent outside of the home to cloud services, where that data is being sent to, and the granularity of the data itself.	A text-based manifest that declares intended data aggregation, including how data will be aggregated, stored, and pre-processed.
Runtime	A trusted hub running locally inside a home	A trusted serverless provider that pulls data from multiple smart homes

1.3.3 Navigating the Design Space of Data Minimization

While MPF scaffolds the development process for data minimization, there remain two crucial design challenges. First, developers must decide what fine-grained data they should collect and how they can use it. Making the process transparent implies that developers are more likely to receive backlash over imprudent data collection. Traditionally, they rely on formal privacy reviews in identifying potential customer acceptance issues with their organizations' data practices. However, this process is slow and expensive, and practitioners often have to make ad-hoc privacy-related decisions with little actual feedback from users.

We introduce Lean Privacy Review, a fast, cheap, and easy-to-access method to help developers decide what fine-grained data they should collect and how to use it appropriately. Lean Privacy Review is a fast, cheap, and easy-to-access method to help practitioners collect direct feedback from users through the proxy of crowd workers in the early stages of design (Chapter 7). Lean Privacy Review takes a proposed data practice, quickly breaks it down into smaller parts (Fig. 1.5), generates a set of questionnaire surveys, solicits users' opinions, and summarizes those opinions in a compact form for practitioners to use. Our results show that (1) the discovery of privacy concerns saturates as the number of evaluators exceeds 14 participants, which takes around 5.5 hours to complete (i.e., latency) and costs 3.7 hours of total crowd work ($80 in our experiments); and (2) LPR finds 89% of privacy concerns identified by data practitioners as well as 139% additional privacy concerns that practitioners are not aware of, at a 6% estimated false alarm rate.

Second, developers need to prioritize the end-user privacy management features to implement. While researchers and practitioners actively develop new privacy-enhancing technologies, most of these proposals are studied independently and evaluated with a relatively low baseline. As a result, it is unclear what types of features users prefer and why.

Chapter 8 introduces a new design method called Privacy Speed dating, a design method for evaluating and critiquing independent end-user privacy management features. We extend the operator-based graph in

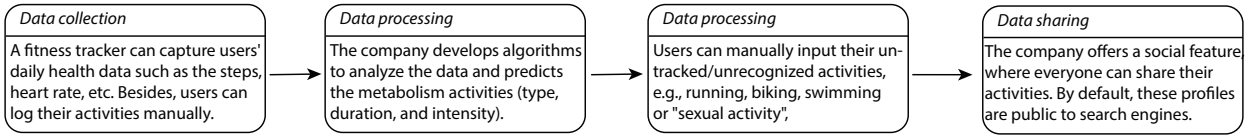


Figure 1.5: An example Lean Privacy Review privacy storyboard of "#9 Fitness tracking". This storyboard describes how a wearable technology company collects users' intimate behavior data and makes them public by default. Lean Privacy Review shares the same philosophy with Peekaboo and MapAggregate, where each node is a stateless data action. Practitioners offer textual descriptions about data actions, such as what data is being collected, where it is being sent to, and why. Given a storyboard, Lean Privacy Review generates a set of questionnaire surveys, solicits users' opinions, and summarizes those opinions s in a compact form for practitioners to use.

MPF into a privacy storyboard to help developers collect feedback from users. We storyboard 11 privacy protection concepts (e.g., Fig. 9.1) that can be built upon Peekaboo in Chapter 3 and ask another 227 participants to compare them head to head, offer a list of the pros and cons for each concept, and rank these design concepts.

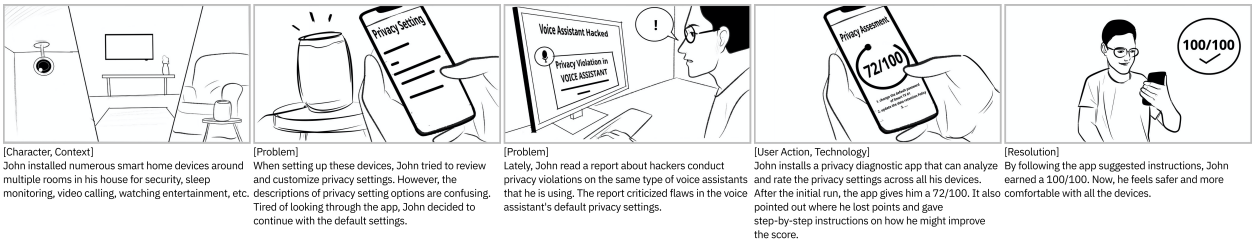


Figure 1.6: Out of 11 different storyboards we created based on participants' wishlist features, *Privacy Diagnostics* was ranked as the most favored feature in 63% of the comparisons, when participants ranked three randomly selected storyboards. *Privacy Diagnostics* is a mobile app that can assess the smart home configurations across all the smart home devices and offer step-by-step instructions on how to improve the score. Participants praised it for "being simple," "offering privacy control," and "requiring little management efforts."

Part 1

UNDERSTANDING DATA COLLECTION BEHAVIORS

Chapter 2

MobiPurpose: A Framework for Understanding the Purposes of Data Collection

Many smartphone apps collect potentially sensitive personal data and send it to cloud servers. However, most mobile users have a poor understanding of why their data is being collected. We present MobiPurpose, a novel technique that can take a network request made by an Android app and then classify the data collection purposes, as one step towards making it possible to explain to non-experts the data disclosure contexts. Our purpose inference works by leveraging two observations: 1) developer naming conventions (e.g., URL paths) often offer hints as to data collection purposes, and 2) external knowledge, such as app metadata and information about the domain name, are meaningful cues that can be used to infer the behavior of different traffic requests. MobiPurpose parses each traffic request body into key-value pairs, and infers the data type and data collection purpose of each key-value pair using a combination of supervised learning and text pattern bootstrapping. We evaluated MobiPurpose's effectiveness using a dataset cross-labeled by ten human experts. Our results show that MobiPurpose can predict the data collection purpose with an average precision of 84% (among 19 unique categories).

2.1 Introduction

A major privacy concern of mobile apps is that they can potentially access a great deal of sensitive personal information [4, 163]. Here, we focus on one dimension of mobile privacy, namely the point when sensitive data leaves the device and is sent to remote servers over the network. Many privacy researchers have adopted this network perspective, studying the raw attributes of privacy-sensitive mobile data sharing [189, 314, 340, 342, 399, 403], namely which app is sharing data, what data is being shared, and where

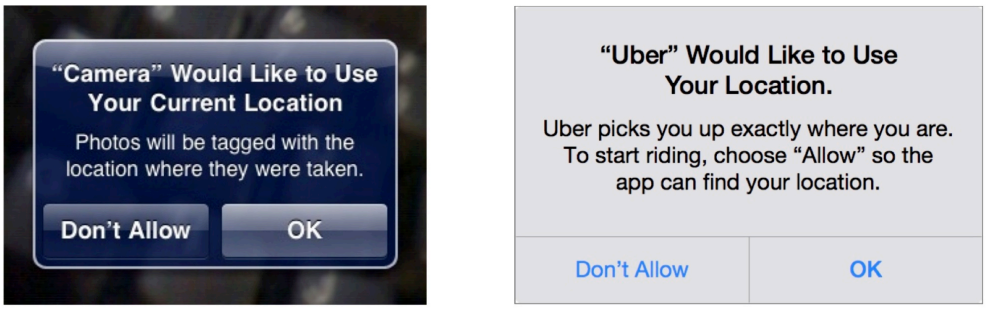


Figure 2.1: Android and iOS ask developers to offer explanations about why an app is accessing sensitive user data by way of a purpose string or a "usage description". However, these purpose strings are only shown at the user interface layer, and can be arbitrary text. There is no easy way to verify if the purpose strings are accurate. Furthermore, these purpose strings are a relatively recent addition to Android and iOS, and are not yet widely adopted.

that data is going.

Currently, though, there is little research looking into *why* an app is requesting access to sensitive data (we also refer to this as the *purpose* of data collection). "Why" is a fundamental component of contextual definitions of privacy [241, 272]. For example, users might be more willing to provide their locations to make search results more relevant, but less for targeted advertising. Past work has also found that surfacing what data an app is using without explaining why can raise privacy concerns [225, 241, 363]. For example, Lin et al. [225] found that when end-users were told that the Dictionary app accessed their location, they were very concerned about privacy. However, when told that location data was only used to search for trending words that people nearby are looking up, they felt much less concerned. As another example, Brush et al. [43] found that people were very willing to share their location data to help cities plan bus routes or to get traffic information, but reluctant to share the same data for ads or for maps showing one's travel patterns.

A major challenge here is that end-users currently have little support for understanding the purpose of data use in smartphone apps [122, 225]. Android and iOS now offer the capability to explain why an app is accessing sensitive user data by way of a purpose string or a "usage description" [94]. Figure 2.1 illustrates two example app permission modal dialog boxes, which depicts that Camera/Uber (who) is requesting location (what) to tag photos or locate passengers (why). However, these purpose strings are only shown at the user interface layer, and can be arbitrary text. There is no easy way to verify if the purpose strings are accurate. Furthermore, these purpose strings are a relatively recent addition to Android and iOS, and are not yet widely adopted [348].

There has been some past work looking at how to infer purposes, for example based on static text analysis of executables [367] or by considering the libraries used by the app [61, 205]. Instead, we look at a

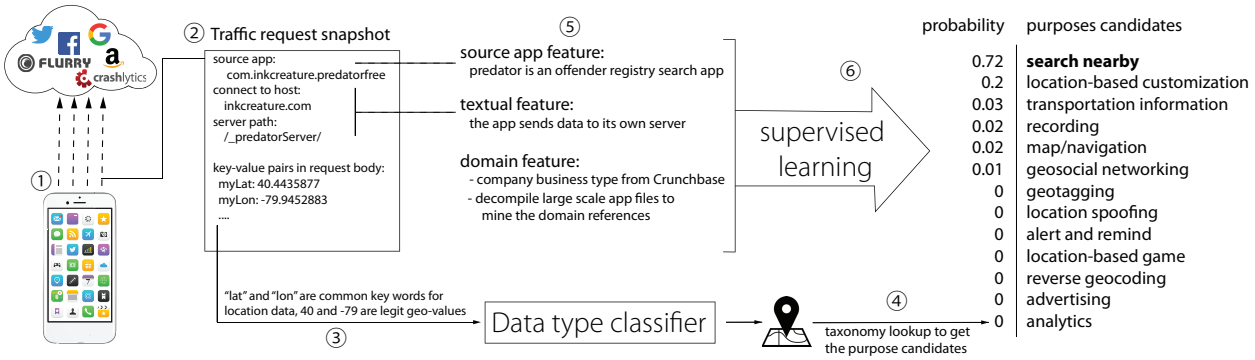


Figure 2.2: Overview of MobiPurpose's workflow (running on in-lab devices). Numbers indicate the order of the methodological steps. (1) MobiPurpose uses a UI automation tool to simulate user interaction with an app, allowing us to navigate an app and generate network traffic. (2) MobiPurpose uses a VPN that acts as a Man-in-the-middle (MITM), intercepting all outgoing traffic requests from a smartphone and parsing the body of requests into key-value pairs. (3) A bootstrapping algorithm automatically classifies each key-value pair data type based on the textual patterns. (4) Based on the identified data type, we find associated purpose candidates through a taxonomy lookup. (5) Based on internal and external features extracted from the traffic snapshot, (6) we use a supervised machine learning model to predict the data collection purpose.

new approach for inferring purpose, namely based on network traffic, since looking at data egress provides a better vantage point in understanding the specific data that is flowing out of the phone.

In this chapter, we present *MobiPurpose*, a novel technique to automate the inference of mobile traffic purposes. MobiPurpose takes a network request made by a smartphone app and classifies the purposes of each of the key-value pairs, to help explain to non-experts why their data is being collected.

MobiPurpose is designed to run on in-lab devices instead of end-users' smartphones. We built tools to automate downloading and installing apps onto a smartphone, interacting with those apps using an Android Monkey Script [345], and capturing synthetic network data from in-lab devices. We assume that the network service API collects nearly the same types of data from different devices. While we are intercepting data using in-lab devices, the learned knowledge can be transferred to real users as well. MobiPurpose is a key step towards our broader goal of offering a public, large-scale database (similar to PrivacyGrade [205]) that can show not only what data is being collected by different services around the world via smartphone apps, but also help explain why. This will allow us to analyze the privacy practice in mobile apps as well as API service providers in an unprecedented way.

2.1.1 MobiPurpose: Inferring the Purposes of Network Traffic in Mobile Apps

MobiPurpose uses a VPN approach to intercept all outgoing traffic requests and parses the HTTP(S) request body into Key-Value (KV) pairs (see Fig. 2.2 ②). HTTP(S) is considered the most common protocol

in smartphone-server communication [118] and recent studies show that more than 80% of apps have structured responses [342]. Using a combination of supervised machine learning and natural language processing (NLP) techniques, *MobiPurpose* inspects the data disclosure context of each KV pair. Figure 2.2 illustrates an example of MobiPurpose inferring the data type of *"myLat:40.44; myLon:-79.94"* as *LOCATION* and the corresponding purpose as "search nearby".

At the core of *MobiPurpose* is our purpose taxonomy (§2.2), in which we enumerate the potential data collection purposes for each data type explicitly. That is, rather than having to generate text describing the purpose, our goal is to instead select the appropriate purpose from this taxonomy. Given any traffic request, we first infer what data types are involved using a bootstrapping NLP approach (§2.3), and then find the associated purpose candidates through a taxonomy lookup. Then, we use a supervised machine learning approach to predict the data collection purposes (§2.4).

Our machine learning approach emulates how reverse engineering experts [91, 287] use various cues in network traffic (e.g., variable names, the URL path, etc.) to infer API details (e.g., data types, data collection purposes). We first observed a group of experts examine a large set of network traffic requests and label data types and purposes (§2.6), and then asked the participants to describe their reasoning processes (§2.4). Based on these observations, we propose a set of computational features (§2.4) to model these data patterns by leveraging two intuitive facts:

- Developer naming conventions often make identifiers self-explainable, in both KV pairs and URLs. For example, a KV pair *"userAdvertisingId : 901e3310-3a26-487e-83c7-2fa26ac2786c"* is likely a unique ID, because the keyname contains keywords such as "advertising" and "id", and the value is a machine generated UUID. As another example, a network request connecting to reports.crashlytics.com is likely for crash reports generation, since the domain is a neologism of "crash" and "analytics", and the subdomain is "reports". We also discuss potential obfuscations in §2.9.
- External knowledge, such as the app description and the domain owner, can be meaningful cues to infer the data collection purpose [225]. For example, suppose we have a game app that sends location data to admob.com. Since AdMob is a mobile advertising company, we can infer that the location data is probably used to tailor what advertisements to show based on the user's location.

We used automated tests to scale the traffic data collection. We developed a test harness[1] that can install apps on an Android device and then explore those apps using an interface automation monkey [345] that randomly clicks on UI elements. Over 50 days, we used 8 Android devices to collect data on 15k apps, each running the monkey for 3 minutes, and intercepted 2 million unique traffic requests (§2.5). We then sampled 1059 traffic requests and had 10 human experts manually label the behavior of each traffic request using our What&Why Taxonomy. Since data collection purposes can be subjective and ambiguous, we collected 3 independent labels for each traffic request.

[1] Open sourced at https://github.com/CMUChimpsLab/MobiPurpose

We used the labeled data to evaluate our data type inference algorithm and train a purpose classifier. In our experiments, we found that MobiPurpose achieved an average precision of 95% in predicting "what" (among 8 unique data type categories) and 84% in predicting "why" (among 19 unique purpose categories).

As noted earlier, our approach is intended to replicate what developers already do when attempting to reverse engineer APIs, so our method does not account for things like deliberate obfuscation of variable names or server names (See §2.9), or account for additional hidden purposes of how the data is used once it is on a remote server. However, we believe that MobiPurpose can be an effective starting point in improving the transparency of the smartphone app ecosystem, by mapping out what data is being shared with why it is being collected.

2.1.2 Contributions

Our contributions lie in the technical implementation of purpose inference. MobiPurpose is the first work to address automatic mobile network purpose inference. To achieve that, we first make the machine inference feasible by turning the purpose description generation task into a classification task. We then propose practical computational methods to emulate how reverse engineering experts use various cues in network traffic (e.g., variable names, the URL path, etc.) to infer data collection purposes. Our specific contributions are as follows:

- We present the design and implementation of the first system that can automatically categorize data collection purposes of sensitive data in mobile network traffic. To achieve this, we developed an extensible taxonomy of purposes, and turned the purpose description generation task into a classification task.
- We propose a set of practical computational features and patterns for inferring purposes. We investigate the effectiveness of different kinds of features, showing that text-based features and domain features offer high gains, while source app features offer marginal improvements.
- We evaluate our framework using a human-labeled data set of 1059 labeled instances of network data from 815 different apps, which contact 636 distinct domains. We found that MobiPurpose can achieve an average accuracy of 95% for data type inference and 84% for data purpose inference.

The remainder of this chapter is organized as follows. The system design of MobiPurpose is discussed in §2.2-2.4. We present our experiment and evaluation in §2.5-2.7, followed by a discussion of the related work in §7.2. We conclude with the limitations and design decisions of our system in §6.10-7.13.

2.2 Building a Purpose Taxonomy

This section describes the design and development of our purpose taxonomy, which turns the purpose description generation task into a classification task. We present the full taxonomy in Fig. 10.14 and

Table 10.3, 10.5, 2.4, 2.5. The goal of this taxonomy is to offer a collection of purposes that is comprehensive (covers the vast majority of use cases of sensitive data), have a meaningful granularity (a given purpose should not be too narrow in only characterizing very few apps, nor too broad), and understandable (developers and end-users can understand each purpose with minimal explanation). Below, we describe our process for developing this taxonomy of purposes.

2.2.1 Existing Purpose Taxonomies

We first conducted a broad survey of existing purpose taxonomies (Table 2.1 enumerates a partial list). We found that two distinct types of purposes were used mixedly in the past work. Some of the purposes are proposed to describe the specific context explicitly (e.g., nearby search, map navigation), while the rest are more general (e.g., legitimate, primary).

Our goal is not to argue if the privacy-sensitive data disclosure is legitimate. Instead, we want to communicate to non-experts the data collection contexts and help them understand what they should expect. So we opted to focus on purposes that better characterize the functionality, e.g. advertising, SNS, etc, but exclude purposes such as "primary" [149], "internal use" [226], "legitimate" [113], or "core functionality" [363]. We discuss the detailed rationale behind that choice in §2.10.

Table 2.1: A partial list of employed purposes in past work and MobiPurpose. MobiPurpose taxonomy does not capture if a data disclosure is legitimate or malicious. Rather, the goal is to characterize the traffic requests and explain why and how they can cause trouble potentially.

	Data collection purposes (Why)
Lin et al. [226]	utility, advertising, UI customization, content host, game, SNS, analytics, payment, internal use
Wang et al. [367]	10 fine-grained purposes (e.g. search nearby places, map/navigation) for two permissions (location, contact list)
Han et al. [149]	primary, advertising, content server, analytics, API, don't know
TaintDroid [113]	legitimate or non-legitimate
Kleek et al. [363]	core/non-core functionality, marketing
Martin et al. [241]	game, weather, social networking, navigation, music, finance, shopping, productivity
MobiPurpose	76 fine-grained purposes (e.g., ad, analytic) for 16 data types (e.g., device info, tracking ID)

2.2.2 Methodology

We iteratively developed our taxonomy by examining a large number of smartphone apps, in terms of the sensitive data they used (Android permissions) and network traffic behaviors.

Permission access. Android classifies all permissions into three protection levels: normal, signature,

Figure 2.3: Taxonomy card sorting sessions. Left: participants first sort the raw cards (white cards) and proposed their categorizations (post stickers in assorted colors) independently, in which each cluster (single color) is the individual result. Right: after the individual sorting session, we have group discussions to reach an agreement and use an affinity diagram technique to form a hierarchical taxonomy.

dangerous. Requesting the 9 "dangerous" permissions require users' explicit consent [97]. We indexed Android apps (see dataset description in §2.5) based on their required permissions, and sampled 100 apps for each "dangerous" permission, resulting in a total of 900 apps. We then printed the app name, app description, and permission on 900 index cards, which would be used for the later card sorting session.

Network traffic. We installed 20 most popular free apps across all categories on two smartphones and then actively used each app for 3 minutes, with the goal of invoking major pieces of functionality in these apps. We used a MITM (man-in-the-middle) VPN app [46] to intercept 5504 unique HTTP(S) traffic requests connecting to 321 unique domains. We then sampled 400 of these traffic requests, each contacting a unique URL (domain + path). We printed the destination host name, app name, app description, and data body on 400 index cards.

We conducted four card sorting [378] sessions to create and refine the taxonomy. In each iteration, participants first independently created their own taxonomy (Fig. 2.3 left) and then discussed with other participants to reach an agreement (Fig. 2.3 right). The converged taxonomy would then be the starting point of the next iteration. Overall, we had 12 participants across these four sessions, involving a mix of mobile developers and UX designers.

In our earlier iterations of the taxonomy, one main question that arose was how to organize the purposes into meaningful categories.

- We started with ***independent purpose categories*** similar to [149, 226], such as Social/Communication, Advertising, Game, Multimedia consumption, Content, Analytic, Personalization, and Map/Navigation. This structure is intuitive for both developers and users since apps are organized through these categories in the App Stores. However, this purpose granularity mainly stays at the app-category level, only providing little privacy insights.

- We then switched to a ***hierarchical structure*** similar to [241, 363] where the end leafs were purposes, and the parent nodes were app categories. However, participants in later iterations found this hard to use, in terms of being able to look up purposes quickly. Apps in any category can make traffic requests for any purpose. Besides, this structure can help users understand how privacy friendly the app is, but not the disclosure context of the specific network request.
- Our final taxonomy is inspired by the app permission modal dialog boxes (Fig. 2.1). We still use a hierarchy in our final taxonomy (see Fig. 10.14, Table 10.3,10.5,2.4,2.5), but the parent nodes are now data types, with the end leafs being purposes (same as before). The purpose candidates listed in Figure 2.2 are the leaf nodes of "*LOCATION*" node. We also organize the data types into four groups (see Fig. 10.14): *PHONE ID*, *PHONE STATUS*, *PERSONAL DATA*, and *SENSOR*.

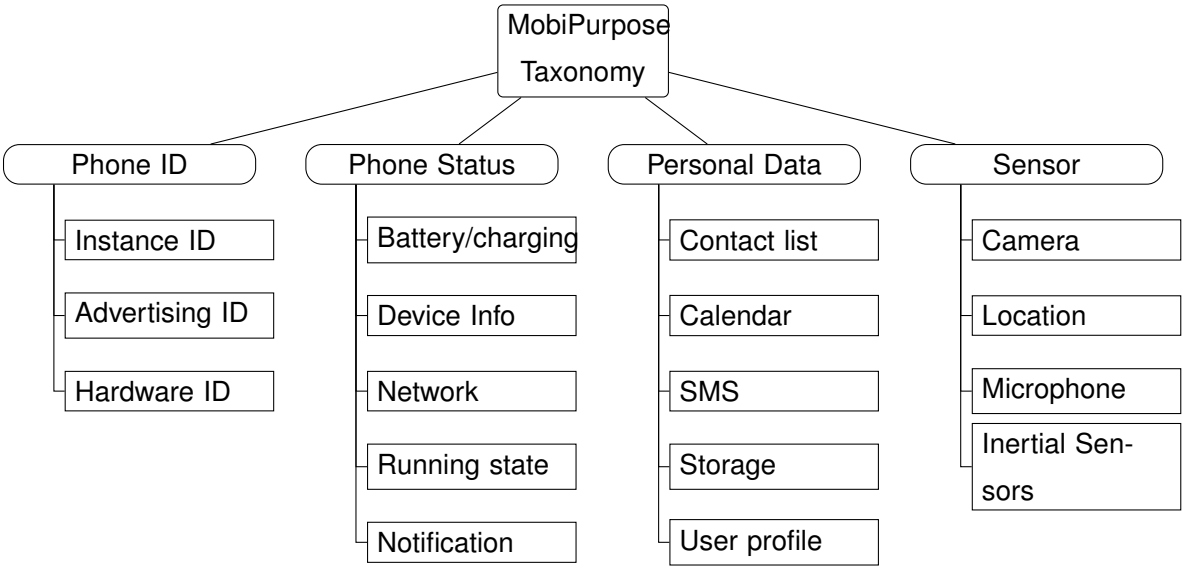


Figure 2.4: Taxonomy overview for Data Type Groups

[2]MobiPurpose runs apps on in-lab devices so that MobiPurpose can identify the unique identifier (ID) types (e.g., Ad ID, MAC address, Instance ID). Official development guidelines [99] often recommend developers to use different types of ID in different contexts; however, our dataset shows that developers use different types of ID mixedly. If future developers better adopt these best practice guidelines, the taxonomy of ID purposes would be listed separately.

Table 2.2: The purpose taxonomy for *Phone ID*.

Data types (what)	Purposes (why)	Example usages
Instance/hardware/ advertising ID[2]	Tracking for advertising	Support ad targeting/evaluation
	Tracking for data analytics	Avoid redundant device counting in marketing.
	Signed-out personalization	Personalize news for sign-out users
	Anti-fraud	Enforce free content /advertisement limits
	Authentication	Relogin a user with a cookie

Table 2.3: The purpose taxonomy for *Phone Status*.

Data types (what)	Purposes (why)	Example usages
Battery/charging	Battery-based event trigger	Show charging/low battery notifications
	Power management	Adapt the phone settings to save battery
	Data collection for analytics	Analyze the battery assumption of apps
Device Info	Interface customization	Customize the interface based on the resolution
	Data collection for ad	Collect data for ad personalization
	Data collection for analytics	Collect data for marketing analysis
Network	Network switch notification	Show wifi/lte switch notification
	Network optimization	Download low resolution images when on LTE
	Geo localization	Use IP to infer the geo location
	Data collection for ad	Collect data for ad personalization
	Data collection for analytics	Collect data for marketing analysis
Running state	Cross-app communication	Support cross-app communication
	Task management	Detect foreground tasks

Notification	Interface customization	Customize lock screen notifications
	Interruption management	Delay notification to manage interruption

Table 2.4: The purpose taxonomy for *Personal Data*.

Data types (what)	Purposes (why)	Example usages
Contact list	Backup and Synchronization	Backup contacts to the server
	Contact management	Merge duplicate contacts
	Blacklist	Block unwanted calls
	Call and SMS	Make VoIP/Wifi calls using Internet
	Contact-based customization	Add contacts to the personalized dictionary of input typing apps
	Email	Send Email to contacts
	Find friends	Find common friends who use the same service
	Record	Display call history
	Fake calls and SMS	Select a contact to fake calls
Calendar	Context prediction	Predict if the user is commute to workplace
	Schedule	Manage schedule conflicts
	Alarm	Notify calendar events ahead
SMS	Send messages	Send messages through apps
	Organize messages	Cluster/delete/re-rank SMS messages
	Extract message content	Extract the verification code in SMS
	Block messages	Block unwanted SMS
	Schedule messages	Delay the messaging sending
	Backup/sync messages	Backup messages to the server
Storage	Access photo album	Modify or upload photos
	Manage downloaded files	Save photos to local storage
	App data persistent storage	Save configuration files
Account/user profile	Third-party login	Login through Facebook/Google accounts
	Data collection for analytics	Collect data for marketing analysis
	Data collection for ad	Collect data for ad personalization

Table 2.5: The purpose taxonomy for *Sensor*.

Data types (what)	Purposes (why)	Example usages
Camera	Flashlight	Turn on/off flashlight
	Video streaming	Steam the video capture
	Code scanning	Scan QR code
	Document scanning	Scan document
	Augment reality	Capture videos
	Text recognition	Recognize the text in the live video capture
	Photo taking	Take photos
Location[3]	Nearby Search	Search nearby POIs/real estates
	Location-based Customization	Fetch local weather/radio information
	Query Transportation Information	Estimate the trip time through Uber API
	Recording	Track the running velocity
	Map and Navigation	Find the user location in Map apps
	Geosocial Networking	Find nearby users in the social network
	Geotagging	Tag photos with locations
	Location Spoofing	Set up fake GPS locations
	Alert and Remind	Remind location-based tasks
	Location-based game	Play games require users' physical location
	Reverse geocoding	Use the GPS coords to find the real world address.
	Data collection for analytics	Collect data for marketing analysis
	Data collection for ad	Collect data for ad personalization
Microphone	Voice Authentication	Authenticate users using voices
	Audio streaming	Make VOIP phone calls
	Voice control	Use voice to send the command
	Speech recognition	Turn the speech audio into text
	Audio recording	Record voice messages
	Acoustic event detection	Sense users' health using microphone
	Acoustic communication	Decode audios to receive messages
	Music	Record songs

Accelerometer/ gyroscope/ magnetometer	Step-counter	Count users' steps
	Game input controller	Detect device movements to control game input
	Map/Navigation/Compass	Use dead reckoning to improve localization
Proximity sensor	Speaker/display activation	Turn off screen if the phone is near the users' ear

2.3 Data Type (What) Inference

The hierarchical structure of our taxonomy allows us to narrow the purpose candidates down significantly by inferring the data type first. This section describes how MobiPurpose can predict the data type of *"myLat:40.44; myLon:-79.94"* as *LOCATION* (Figure 2.2③).

Data type inference in the network traffic has been studied extensively in the past. Past projects use either hard-coded regular expressions [210, 340] or supervised machine learning approaches [314] to classify data types. However, the regular expressions cannot be generalized to unseen patterns, and labeling training data for our taxonomy would be a daunting task. Instead, we adapt a traditional generative NLP technique to avoid data labeling and scale the automation to fine-grained data types. More specifically, we first use a bootstrapping method to build a key-value text pattern corpus, and then derive a Bayesian probabilistic model from the corpus to classify the data type.

2.3.1 Using Pattern Bootstrapping to Build a Corpus

Our bootstrapping approach is inspired by prior work in information extraction tasks, which requires *minimal human participation* [6, 167, 182]. The key idea is leveraging the key-value text pattern redundancy. We noticed that many key-value pairs in our data set were combinations of constrained text patterns, owing to two reasons. First, developers share similar naming conventions, which limits the key name variations. For example, typical patterns for latitude key names are: "lat," "***_lat," "lati," "latitude," etc. Second, our devices also share similar configurations (e.g., physical location, device models), which limits the value variations as well. If a developer tries to collect some GPS data, the devices will send nearly the same values to the server.

[3]SENSOR.LOCATION in MobiPurpose only refers to the location information of the device, but not the location data in the search query.

More concretely, if we manually start with a handful of patterns for LOCATION key names, we can then find more key-value pairs containing unknown value patterns. These new value patterns can then be used to find more key-value pairs again, this time with new key name patterns. In each iteration, the algorithm will only keep the most reliable ones for the next iteration. This iterative process can collect a large text pattern corpus with minimal human efforts. We describe the approach in more detail below.

Table 2.6: Extracted text patterns for key-value examples. We extract two types of text patterns for any key/value string: bag-of-words & special string, and then use these text patterns for the later bootstrapping process.

Key: Value	bag-of-words	special string
devid: 99349319-a6c7-4657-a3bc-6929c52090e1	dev, id	UUID
ip_addr: 128.237.175.242	ip, addr	IP
device_model: Nexus_6P	device, model, nexus, 6p	
pickup_lat: 40.4431531	pickup, lat	LATITUDE_NUM

Extracting text patterns. A strawman solution is to use the exact string match as the text pattern. However, it doesn't fit well with the developer variable contexts. For example, common latitude key names include "dev_lat," "device_lat," "my_lat," "mylat," etc. Values of UUIDs are universally unique but share a common string pattern. We need a more generalizable representation to capture these repetitive text patterns. Through experimentation, we extract two types of text patterns for any key/value string:

- *Special-string check*: we first use regex expressions and bi-gram models to recognize common special strings, such as MAC address, IP, Email, URL address, UUID [98], Advertising-ID [96], MD5 Hash, package name, timestamp, isnumber, latitude/longitude, developer version number, randomly generated strings [264], etc.
- *Bag-of-words*: If the key/value string is not a special string, we parse the string into a bag-of-words representation using an open source English WordSegmentation model [275]. To better handle technical jargons (e.g., lte, gsm) and casual abbreviations (e.g., ad for advertise), we manually add these common terms into the open source model.

Bootstrapping the text patterns. Algorithm 1 shows pseudo-code for our bootstrapping process. The method is initialized with some manual seed rules to provide initial rules for learning. For example, if the key contains "lat" or "lon" and the value is a legitimate latitude/longitude string, the data type would be *LOCATION*.

We (1) use the initial rules to find an initial *LOCATION* key-value set (denoted as L-KV), (2) extract all the text patterns that appear in the L-KV set, (3) compute the frequency of different text patterns and identify the text patterns that commonly occur (if their frequencies exceed a threshold), (4) use these text patterns to

search the whole data set to recognize more *LOCATION* key-value pairs, and (5) update the L-KV set and then go back to (2). This process is repeatedly performed, each time with higher frequency thresholds to ensure high reliability, until the *LOCATION* KV set converges.

Identifying the initial seed set is the necessary first step for snowball bootstrapping algorithms [6]. Choosing different seeds should only impact the results slightly if the frequency thresholds are carefully tuned. In our implementation, the initial rule set contains an average of 7.5 seed rules per data type. The data type of Device has most seed rules (14) as the text patterns are diverse, while data type of IP has the least seed rules (4).

Input: A large collection of key-value pairs R and a list of data types $D = \{d_1, d_2, ...\}$

Define: For any key/value string, we extract text patterns $T = \{t_1, t_2, ...\}$.

Initialization: For each data type d_i in D, we initialize a set of seed rules to extract a initial key-value set E_i.

Bootstrapping Algorithm:

Step 1: Traverse all the keys and values in E_i, and extract the patterns as T_k and T_v.
Count the frequency of unique patterns in T_{key} and T_{value}, and remove less common patterns.
Annotate the results as T'_k and T'_v.

Step 2: Search key-value pairs in R using T'_k and T'_v.
If the key matches T'_k **or** the value matches T'_v, we add that pair to E_i.
Go back to Step 1 to find new patterns, and repeat the iteration until the size of E_i does not grow.

Output: A text pattern corpus where key-value pairs are labeled with different data types.

Bayesian classifier:

Given any key-value pair, we estimate the likelihood of different data types using a bayesian method:

$$P(c|k,v) = \frac{P(k,v|c)*P(c)}{P(k,v)}$$

Determining the data type for that key-value pair is equivalent to finding c^* that maximizes $P(c|k,v)$:

$$c^* = \arg\max_c P(c|k,v)$$

ALGORITHM 1: MobiPurpose's bootstrapping algorithm for data type inference

2.3.2 Bayesian Classification

It is possible that the bootstrapping approach might miss some key-value pairs for each data type. Furthermore, running the bootstrapping iteration can be slow. To address these problems, we used the bootstrapped results as a corpus to quantify the weights of different text patterns, and built a probabilistic Bayesian classifier to infer the data type. Let $C = \{c_1, c_2, ..., c_n\}$ be the n data types (n = 16 in our taxonomy), $E = \{E_1^*, E_2^*, ..., E_n^*\}$ be the converged KV set for each data type. Given any key-value pair (k,v), we can extract the text patterns $X = x_1, x_2, x_3, ...$ to represent that pair. The conditional probability that the data type of (k,v) is c is $P(c|k,v)$. Determining the data type is equivalent to finding c^* that maximizes $P(c|k,v)$. From Bayes'rule, $P(c|k,v)$ can be formulated as:

$$c^* = \arg\max_c P(c|k,v), \qquad where \qquad P(c|k,v) = \frac{P(k,v|c)*P(c)}{P(k,v)} \tag{2.1}$$

in which $P(c)$ represents the prior probability of selecting c without the observation of the (k,v), $P(k,v|c)$ is the likelihood function, which expresses how probable the text patterns can be seen in data type c, and $P(k,v)$ is the normalization constant.

We model $P(c_i)$ as the percentage of each data type in all key-value pairs. For example, if MobiPurpose identifies 1,000 privacy-sensitive pairs and 30 of them are *LOCATION* pairs, $P(LOCATION)$ is 0.03. We use text patterns to represent each key-value pair; therefore the likelihood function is interpreted as the product of the possibility $P(x_i|c)$ each text pattern happening in class c:

$$P(k,v|c) \sim P(X|c) = \prod_{1}^{n} P(x_i|c) \tag{2.2}$$

In our traffic data set, not all data fields are necessarily privacy-sensitive (e.g., timestamp, version number). We further empirically determine a threshold. If all the likelihood predictions are below that threshold, we classify the key-value pair into the *NON-PRIVACY* category.

2.4 Data Collection Purpose (Why) Inference

Based on the data type prediction, we can find a small list of associated purpose candidates through a taxonomy lookup (Fig. 2.2④), and then use machine learning techniques to infer the likely purpose. To do so, we leverage our ten participants from the data labeling step (see details in §2.6). Specifically, after the labeling, we discussed the data patterns (DP) they observed and asked them how they determined the purpose that they labeled. Based on this discussion, we derive a number of computational features from three separate data sources: traffic requests (G.1,2,3), app descriptions (G.4) and the host domain information (G.5,6) (Table 2.7).

2.4.1 Data Patterns Observed by Labeling Participants & Features Extraction

Here we report the data patterns which our participants mentioned for inferring purposes, and how they informed our proposed features.

Table 2.7: The features used in classification model can be classified into three groups: embedded textual features, source app features and domain features.

Group	Feature	Details
Embedded textual features	G1: *url path bag-of-words*	A bag-of-words representation of the URL path and sent data (160∼350 dimensions)
	G2: *package-endpoint similarity*	A 3 dimension vector, each value represents if the URL components share a common string with the package name.
	G3: *co-sent data types*	The length of k-v pairs and involved data types.
Source app features	G4: *app category*	The app category can be queried from Google Play using the package name (15 binary dimensions).
Domain features	G5: *owner business type*	The business categories from Crunchbase.
	G6: *domain occurrences in app corpus*	We first decompile the app file, then find the apps that contains the domain inside their source code, and count the app distribution across different app categories. We use the top app categories to represent the business type.

DP.1: *The keywords in the host path (e.g., searchnearby, fetchads) and key-value pairs (e.g., accesstoken) often describe the API behavior directly. Nearly all the participants actively looked for special keywords, such as "ad" and "analytic", during the labeling process.* Given any endpoint address (host+path), we parse the address into components by the URI protocol (i.e., *scheme :// subdomain.domain / path / document . extension ? query # fragment*), segment each component (except scheme) into words using the probabilistic model described in Section 2.3, and encode the results into a "bag-of-words" representation (i.e., number of word occurrences in the URL). We maintain a stopword list, such as "com", "www", "android", "google", to filter out some unnecessary features. We also apply the same approach to the key-value pairs and merge the results into one "bag-of-words" representation.

DP.2: *The string similarity between the app package name and the domain name can indicate if the app is connecting to third party services. For example, the app "net.passone.gwabangeng" contacts "api.passone.net/ ggwabang/questionApi.php", suggesting that the endpoint is not a third party service.* We segment the endpoint address into three parts (subdomain, domain, path), and compute the longest common substrings between the package name and the three components separately. We then use a vector to describe this data pattern, with each item in the vector representing if the corresponding component contains a non-stop-word common string longer than four characters.

DP.3: *The co-sent data can help determine the purpose. For example, advertising services often collect ID as well as screen size, since they need to determine the visual appearance of the advertisement. Analytic services often collect key-value pairs (10+) more aggressively than the rest.* We use a numerical vector

to represent the number of total key-value pairs as well as the number of different data types of key-value pairs.

DP.4: *Participants often read the app description in Google Play to understand the network request context.* In this feature group, we only use the app category in Google Play. To reduce unnecessary features, we merge all the game categories to reduce the feature dimensions [2].

DP.5: *Understanding the service provider (domain owner) business can help the purpose inference. For example, ID collected by "doubleclick.com" are most likely for advertising purpose, since 'DoubleClick' is a subsidiary of Google which provides ad services. Participants often check Google Search, WHOIS to infer the business type.* We developed a set of scripts to automate the domain owner lookup using WHOIS API[4] and Crunchbase API[5]. We use the corporate categories in Crunchbase to represent the organization business type.

However, most international companies do not have a profile on Crunchbase and many domain owners choose "private registration" to hide their WHOIS registration information. We were only able to identify 22% domains using that approach. To address this issue, we leverage the app source code to represent the business types. The intuition is that if an app contacts a domain, the app category can indicate the business type of the domain owner. For example, *uber.com* is contacted by five apps in our network tracing dataset, three in the "Maps & Navigation" category and two in the "Travel & Local" category. We can use these two categories to describe Uber's business. This approach can also identify the domains providing third party services since they are contacted by apps across all the categories.

We extract this domain feature as follows. We first decompile all the 185k apps using Androguard [92] and index the domains in the decompiled source code. For each domain, we find the apps that contain the domain inside their source code and count the app distribution across different app categories. We use two heuristic rules to extract features representations: 1) if the domain is contacted by 10+ apps across 5+ categories, we set the business type to "third party library"; 2) otherwise, we use top 3 app categories to represent the business type.

2.4.2 Feature Selection

A simple bag-of-words model produces too many features which may lead to overfitting. We assume the low-frequency features are less generalizable, so we filter out all the features with a document (i.e. traffic request) frequency below 3%, resulting in feature vectors with a dimension between 160 and 350 for different data types.

[4]https://www.whoisxmlapi.com/

[5]https://data.crunchbase.com/docs

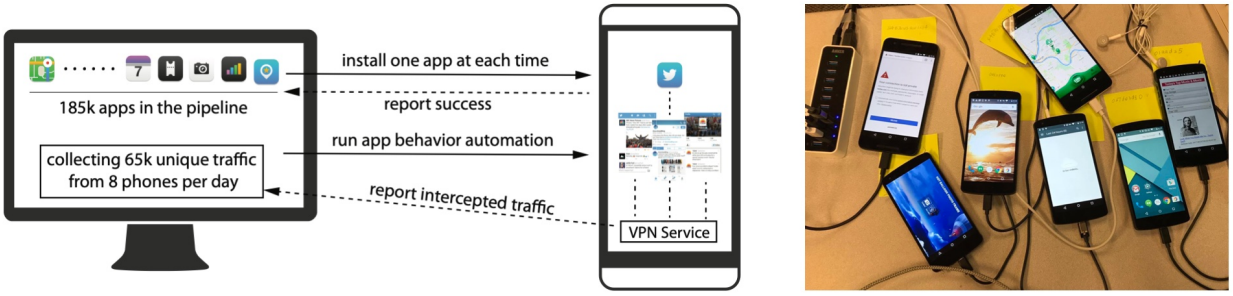


Figure 2.5: Scalable Network Tracing. Left: the pipeline of the network tracing. Right: the hardware configuration. We install the app one by one on eight Android devices, connecting to a PC running app interface real-time analysis. For each app, we will explore the app behavior for 3 minutes and uninstall the app afterward. This configuration can intercept around 65,000 unique traffic requests every day.

2.4.3 Supervised Machine Learning

We use supervised machine learning to train a purpose classifier based on the proposed features. We assume the heterogeneous data sources contribute similarly to the classification process, so we set all the weight components to 1.0. We maintain an independent classifier for each data type and experimented three different classification algorithms: Support Vector Machine using a linear kernel (SVM), Maximum Entropy (ME), and C4.5 Decision Tree (C4.5). In §2.7.2, we compare the performance of different algorithms and different feature combinations.

2.5 Data Collection

In this section, we describe the design and implementation of our network tracing system (Fig. 2.5). The goal of this network tracing system is to make it fast and easy for us to collect diverse network data from a large set of apps in the lab.

Our tracing system is comprised of two main components: a UI automation test tool and a network sniffing app. We implemented the UI automation tool using DroidBot [222], a lightweight UI-guided test input generator, which analyzes the user interface on-the-fly and randomly traverse the UI elements on the screen. We then built our network sniffing app by using the internal Virtual Private Networking (VPN) service provided by Android OS. We manually added a trusted root certificate to our test devices and performed a Man-in-the-Middle (MITM) SSL injection to decrypt SSL/TLS traffic. Our architecture made it easier for us to capture which app initiated the network request, which would have been more difficult if we had used a server proxy approach (e.g. [363]). For each traffic request, we recorded the destination domain, path, source app, and the network request body.

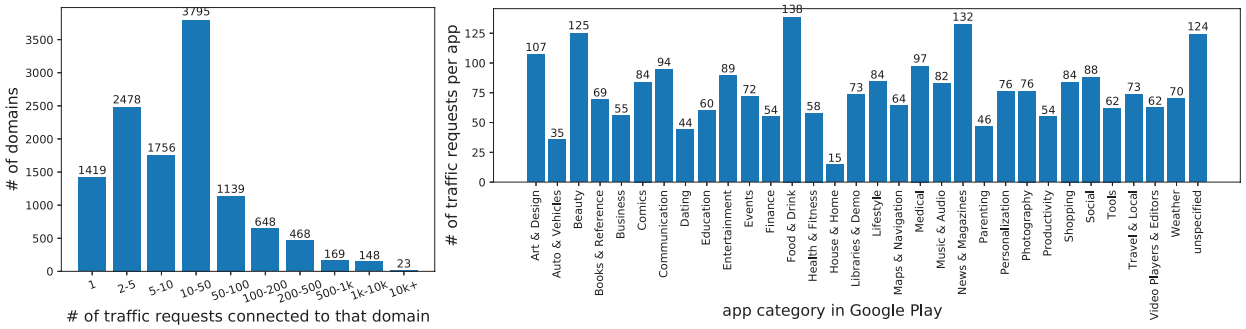


Figure 2.6: Data stats of the 2 million outing traffic requests. The left figure illustrates the long tailed distribution of domains in the data set. The right figure shows that the apps in different categories generate similar numbers of traffic requests.

We tested a large collection of Android apps using this tracing system. We crawled the Google Play web pages in November 2016 to create an index of all the apps that were visible to US users, among which 1.5 million of them were free apps. We decided to only focus on free apps updated after 2015, resulting in 185,173 apps.

For each app, our automation tool first explored the app behavior for 3 minutes and then uninstalled the app afterward. The 3 minutes duration is a tradeoff between our computation resources and our goal to collect as many unique traffic requests as possible. Our hardware configuration (Fig 2.5 right) consisted of 1 PC and 8 Android phones. Roughly, using this set up, we could intercept around 65k network requests and 3k apps every day.

Descriptive statistics. Our data collection process took 50 days to traverse the 185k apps. Due to OS compatibility (our devices were running on Android 7), we were only able to install 30,075 apps. In total, we intercepted 2,008,912 traffic requests from 14,910 apps, contacting 12,046 unique domains (302,893 unique end-point URLs), sending 6,376,833 key-value pair data to a remote server. The top 3 active domains were *doubleclick.net* (261048 requests), *googlesyndication.com* (158672 requests), and *startappservice.com* (124289 requests). The number of requests across domains followed a long tail distribution: 9320 (77.3%) domains were only contacted by one app, and 5653 (46.9%) domains were contacted less than 10 times. Figure 2.6 illustrates a more comprehensive view of the traffic distribution.

Preprocessing the dataset. If we study the individual request directly, the final system would be biased to top domains like *doubleclick.net* and less adaptive for the wild traffic requests. So we first filtered out traffic requests that sent an empty data body and then merged repetitive traffic requests. We grouped requests based on the end-point address (i.e. host + path) using manually crafted regex expressions, resulting in 60,753 unique network API traces. For example, we use `raph.facebook.com\/v\d.\d\/\d{15} ~to` roup similar traffic requests sent to facebook graph API, and merge their parameters (i.e. key-value pairs).

2.6 Labeling the ground truth behavior of traffic requests

Our goal of data labeling was threefold: 1) to evaluate the accuracy of data type classification, 2) to test the taxonomy completeness, and 3) to prepare a data set for purpose inference. We designed each labeling task as a short set of questions about a specific traffic request, with a focus on one key-value pair (Fig. 2.7). We presented participants raw traffic requests as well as the data type classification results. We then ask them to judge if the classification is correct, and label the data collection purpose.

In the pilot test, we found judging the data type was straightforward for engineers, while labeling the data collection purpose required more contextual knowledge [91, 287]. To accommodate that, we incorporated multiple shortcuts (e.g., Google Play, WhoIS) to help participants access relevant external information quickly.

In the example illustrated in Figure 2.7, a participant can find that the app is a shopping coupon app named “The UOL Club” from the Google Play link, infer that the traffic is sent to a UOL server. Combining with the host path "/coupon/category/nearby" and the data body, she can then infer that the traffic request is to "Search Nearby" using longitude and latitude information.

Procedure. To avoid test data leakage [191], we randomly sampled 5k API traces from the preprocessed dataset for evaluation. We developed our bootstrapping algorithm based on the remaining 56k API traces, and run the final data type inference algorithm on the 5k reserved API traces. The labeling instances were from the 5k reserved API traces. Our labeling strategy was similar to Wang et al. [367]. We tried to label at least 30 instances for each purpose manually. However, some purpose instances are scarce (<5%) in our traffic data set. For example, we only observed 5 instances of "reverse geocoding" among 250 location instances. We stopped once we got 30 instances for common purposes (>=5%).

Participants. We recruited ten engineering graduate students with prior Android app development experience. All participants had over three years of experience in Android/iOS development, six were Ph.D. students in computer science, and four are familiar with reverse engineering and mobile privacy research. Participants are also aware of the GPS location of the lab, so they can recognize if the set of coordination is the nearby location.

We introduced our taxonomy in a five-minute labeling tutorial. Since the purpose interpretation can be subjective and in some cases ambiguous (e.g. due to insufficient or incomplete information), we collect three independent labels for each task and allow participants to label multiple purposes for each entity if needed. The purpose labeling is quite time-consuming. Each labeling task takes between 30 seconds to 2 minutes since the participant often needs to generate search queries and check the facts on external websites. We organized 2 group data labeling hackathons with free food, each lasting around 5 hours. No monetary remuneration was paid.

Labeled data stats: We collected labels from 3,177 (1,059 x 3) labeling tasks, where each entity

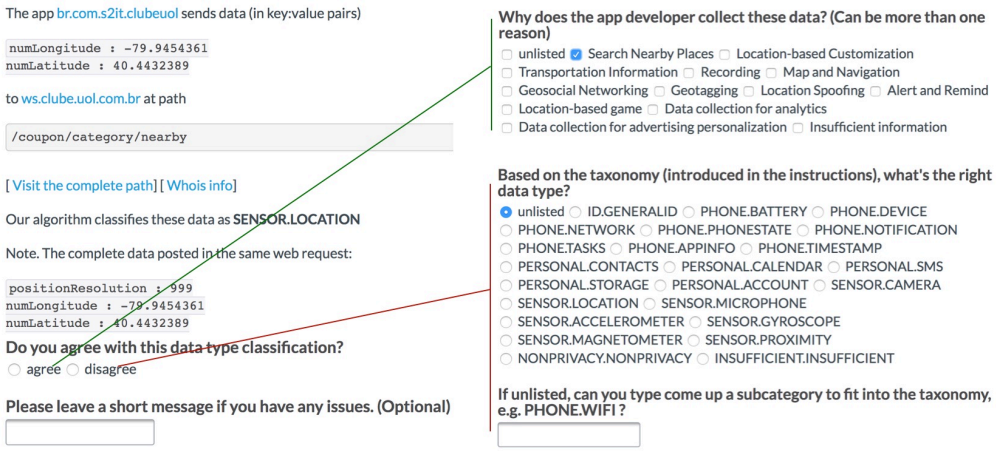


Figure 2.7: An example labeling task. We classified the "numLongitude" and "numLatitude" key-value pairs as *LOCATION* data. Participants determined if our automated classification is correct. If correct, participants continue by labeling the data collection purpose. Otherwise, participants are asked to select the correct data type. In this example, “The UOL Club”, a shopping coupon app, sends *LOCATION* data to the host path "/coupon/category/nearby" on a UOL server. So the purpose is "Search Nearby Places."

was labeled by three independent human experts, covering 7 data types and 34 data purposes (Table 2.8). The data type annotations had a very high inter-annotator agreement: only 23 traffic requests (23/1059 <2%) received inconsistent annotations, giving us confidence that the quality of labels is high and mostly consistent.

Table 2.8: The examples of different data type and the # of labeled instances.

data type	#labels	examples	data type	#labels	examples
ID	400	MAC address, uuid	Battery	7	battery level, if charging
Device	150	phone model, screen size	Network	100	IP address, wifi, lte, RSSI
Running state	2	foreground tasks	Account	150	email, age, gender, zip
Location	250	GPS coord, place name			

Total: 1059

The purpose labeling results further illustrate the feasibility of inferring data collection purpose from the embedded attributes of the app and external factors such as where it contacts. In the labeling tasks, the participants can choose to annotate it with “Insufficient Information” (II) if they find there is not enough information to support their judgment. 123 (123/3177 < 4%) labeling tasks were marked as such, among which only 3 entities were marked as II by all three participants, and 20 entities were marked by two participants.

We merged the purpose labels from different participants using majority voting. If at least two inde-

pendent participants believe the key-value pair is associated with a specific purpose, we accept that label. Based on this standard, there were 118 traffic requests (118/1059 < 12%) that are too ambiguous to reach an agreement. We excluded these traffic requests in the later purpose inference evaluation.

Taxonomy in Practice: The labeling interface also allows participants to add a new purpose label if they find the behavior is not covered by the current taxonomy. Our participants nominated new purposes in 30 labeling tasks (1.2%), resulting in 4 new categories after re-coding: "Network Info - Advertising", "Location - Reverse Geocoding", "Location - Malicious", "ID - Malicious". We incorporate the first two purposes in our final taxonomy and have a separate discussion regarding malicious purposes (Sec. 2.10).

Based on the labeling tasks, we believe the current taxonomy has a good coverage for most regular network traffic. The described process can be repeated from time to time if we need to add/update the taxonomy.

2.7 Evaluation

MobiPurpose uses a two-step procedure to infer purposes. MobiPurpose first infers the data types, using it to find the associated purpose candidates through a taxonomy lookup. Then, MobiPurpose uses a supervised machine learning approach to predict the data collection purpose. Here, we evaluate both the data type inference and purpose inference, and found that MobiPurpose can achieve an average accuracy of 95% for data type inference and 84% for data purpose inference.

2.7.1 Data Type Inference Performance

Methodology. To evaluate the performance of data type inference, we measure the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) for each data type, and present our results regarding precision, recall, and f-score.

We define precision as the fraction of true predictions (TP) among all the predictions (TP+FP) regarding one class. In the labeling tasks, participants determined if our data type inference was correct and provided the correct label if the inference was wrong. We define recall as the fraction of true predictions (TP) among all the true instances (TP+FN) regarding one class. We manually inspect all the fields classified as *NON-PRIVACY* to determine if any privacy-sensitive key-value pairs are missed. We label obfuscated key-value pairs as *NON-PRIVACY*. F-score is the harmonic mean of precision and recall.

To measure the overall correctness of our approach, we use two standard multi-class accuracy metrics: *micro-averaged* (equal weight for each instance) and *macro-averaged* (equal weight for each class) [234, 367]. For micro-averaged metrics, we first sum up the TP, FP, FN for all the classes, and then calculate precision and recall using these sums. So classes that have many instances are given more importance. In

contrast, macro-averaging is the mean of the metrics for all the individual classes. Since we do not have enough data samples in *Battery* and *Running State*, we exclude them in computing the macro-average.

Results. Our results for classifying data types are shown in Table 2.9. Our approach has precisions above 93% for all the classes and an overall precision (micro-averaged) of 95.9%. For recall, our approach identifies over 86% of the privacy-sensitive entities for all the categories with an overall recall (micro-averaged) of 89.9%.

Most FP instances are due to the similar text patterns shared between different classes. For example, *gps_adid* is classified as a location data type but is actually an ID. Reasons for TN are diverse: some personal data are nested in a serialized dictionary; several unusual variable names (e.g., using a token to name ID data) have not been captured by the bootstrapping algorithms, etc.

Table 2.9: Data type inference accuracy

	ID	Battery	Device	Network	State	Account	Location	**Macro-avg**	**Micro-avg**
Precision	97%	100%	93%	94%	100%	96%	96%	95.6%	95.9%
Recall	86.8%	100%	87.5%	92.1%	100%	92.3%	95.2%	90.8%	89.9%
F-score	0.925	1.00	0.902	0.930	1.00	0.941	0.956	0.931	0.928

2.7.2 Purpose Inference Performance

Evaluation Metrics & Methodology. We use 10-fold cross validation to evaluate the performance of purpose inference. For the purpose predictions of each key-value pair, we measure the accuracy, precision, recall and f-score as we did in the data type inference evaluation.

To measure overall performance, we aggregate the individual metrics using micro-average and macro-average for both precision and recall. Since we do not have enough data samples for the Battery and Running State data categories, we only cover the other five data categories (i.e., ID, Device, Network, Account/Profile, and Location) in the purpose evaluation. Due to the imbalance of purpose distributions, some purposes are not common in our dataset. For example, we only have 5 SENSOR.LOCATION.ReverseGeoCoding instances in the labeled data set. We opt to run the classification only among the purpose categories with more than 20 instances.

Results. Our results in classifying the purpose of different data types are shown in Figure 2.8. The Maximum Entropy algorithm performs the best, with an overall accuracy (macro-average) of 84% for all the categories, while SVM outperforms ME in Account/Profile and Device categories. Overall, all the algorithms perform reasonably well across all the data types. MobiPurpose has a comparable accuracy as [367] (85%), which infers the Android permission purposes by analyzing the source code, though MobiPurpose

treats the app as a black box.

	SVM	**ME**	C4.5
Precision	0.85	**0.88**	0.83
Recall	0.84	**0.85**	0.82
F-1 Score	0.84	**0.86**	0.82

(a) ID.

	SVM	ME	C4.5
Precision	**0.82**	0.82	0.81
Recall	**0.81**	0.79	0.80
F-1 Score	**0.81**	0.80	0.80

(b) Device.

	SVM	**ME**	C4.5
Precision	0.82	**0.86**	0.84
Recall	0.81	**0.86**	0.84
F-1 Score	0.81	**0.86**	0.84

(c) Network.

	SVM	ME	C4.5
Precision	**0.85**	0.84	0.84
Recall	**0.85**	0.83	0.83
F-1 Score	**0.85**	0.84	0.84

(d) Account/Profile.

	SVM	**ME**	C4.5
Precision	0.79	**0.82**	0.80
Recall	0.79	**0.82**	0.80
F-1 Score	0.79	**0.82**	0.80

(e) Location.

	SVM	**ME**	C4.5
Precision	0.83	**0.84**	0.82
Recall	0.82	**0.83**	0.82
F-1 Score	0.82	**0.84**	0.82

(f) Overall (Macro-avg).

Figure 2.8: Performance of different machine learning models (using all the features) for classifying purposes

Figure 2.9 shows more details about misclassifications across all the data types. The micro-average accuracy of 82.3% (Fig. 2.9f) is similar to the macro-average accuracy. However, the specific classification errors across different categories vary greatly.

For ID data (Fig. 2.9a), the category "Ad" achieves the best result, with precision and recall both higher than 90%, which is due to the domain name as a feature in recognizing the business types of the domain owner. The category of "Authentication" purposes have 94% precision but recall under 60%. The main misclassifications are in classifying "Sign-out personalization" and "Authentication" into "Analytics," which may because these three purposes collect similar types of data.

For Device data (Fig. 2.9b), the main classification errors come from the confusion between "Ad" and "Analytics": 11 "ad" (35%) instances are classified as "analytics". Similar situations also happen to the Account/profile data (Fig. 2.9d): 8 "ad" (23%) instances are classified as "analytics". In the location matrix (Fig. 2.9e), the major confusion happens between "Location-based personalization" and "Search nearby," which may be due to the semantic similarity between these two purposes.

Through inspecting these individual misclassifications, we categorize errors into two types. First, developers use diverse vocabularies for purpose instances such as "sign-out personalization" and our model didn't capture the statistical patterns of these less common keywords. For example, one term in a "sign-out personalization" instance is "visualdna." Continuing to increase size and diversity of the labeled data set will help here. Second, the semantic correlation between different purposes, such as "Ad" & "Analytic," or "Location-based personalization & "Search nearby," also imposes extra challenges in classifying the purposes. This semantic noise is not introduced only from the data labeling bias, but also the developer's

		P1	P2	P3	P4	P5	Total
Anti-fraud	P1	26	-	-	1	4	31
Authentication	P2	-	16	1	3	7	27
Personalization	P3	3	1	8	1	11	24
Ad	P4	-	-	-	162	15	177
Analytics	P5	-	-	1	11	114	126

(a) Confusion Matrix for ID

		P1	P2	P3	Total
Ad	P1	19	11	1	31
Analytics	P2	3	62	2	67
Interface	P3	3	6	19	28

(b) Confusion Matrix for Device Information

		P1	P2	P3	Total
Ad	P1	20	3	3	26
Analytics	P2	4	29	4	37
NetOptimization	P3	3	1	24	28

(c) Confusion Matrix for Network

		P1	P2	P3	Total
Ad	P1	26	8	1	35
Analytics	P2	5	59	3	67
Login	P3	2	6	17	25

(d) Confusion Matrix for Account/Profile

		P1	P2	P3	P4	Total
Ad	P1	102	4	3	-	109
Analytics	P2	4	26	4	3	37
Personalization	P3	2	3	16	7	28
Search Nearby	P4	-	4	4	25	33

(e) Location

		Prediction	
		Pos	Neg
Truth	Pos	770	166
	Neg	166	2683

(f) Overall (Micro)

Figure 2.9: Detailed confusion matrices using all the features with Maximum Entropy algorithm. Each column represents the instances in a predicted class, while each row represents the instances in an actual class. Note, the P1-Pn shown in each table denote the different purposes, and they vary for each feature (a)-(e). For example, P1 is "Anti-Fraud" for the ID feature, but is "Ad" for the Network feature.

naming decisions.

The classification results in Figure 2.9 is biased to "Analytics", since there are many "Analytics" instances. We also run an evaluation based on a sampled dataset, where each purpose has the same number of instances. This experiment shows similar performance results with an average accuracy of 81%. We report the original data as it fits better the real-world situation.

Feature Comparison. We perform component evaluations to find the efficacy of different feature groups (Table 2.7). We use the bag-of-words features (G.1) as the baseline and experiment with different feature combinations. All the models are trained with Maximum Entropy algorithm. The overall experimental results are summarized in Table 2.10. We can see the bag-of-words features (G.1) alone can achieve an accuracy above 74% and the rest of the features play supporting roles.

Different supporting features illustrate different performance gains in different data categories. The domain features (G.6) improve the performance by 12% in the "ID" category but has little impact on the performance in "Device" and "Account" categories. The co-sent features (G.3) do not perform well in general but is the most valuable feature in "Device" category.

Inspecting the individual results, we find that different categories suffer from different ambiguities. For example, it is hard to differentiate "ad" instances from "analytics" instances in the "ID" category using just the URL. Domain features would be the primary differentiator in this case. In contrast, "ad" services may also collect "Device" information for "analytics" purpose as well. In this case, domain features do not help much in resolving that confusion. Instead, "analytic" services usually collect more key-value pairs than "ad" service, so G.3 (co-sent data types) is the most helpful feature in "Device" category.

Table 2.10: F1-score performance across different feature combinations (Maximum entropy algorithm). Bag-of-words features (G.1) achieve good baseline accuracy alone, with domain features (G.6) offering significant improvements. Using only G1,6 can have a similar performance as using all the features.

	G.1	G.1,2	G.1,2,3	G.1,2,3,4	G.1,2,3,4,5	G.1,2,3,4,5,6	G.1,6
ID	0.74	0.74	0.74	0.75	0.78	**0.86**	**0.86**
Device	0.76	0.77	**0.81**	**0.81**	**0.81**	**0.81**	0.80
Network	0.81	0.81	0.77	0.74	0.76	0.80	**0.86**
Account	0.83	**0.84**	0.81	0.81	0.81	**0.84**	0.82
Location	0.77	0.79	0.78	0.76	0.78	0.82	**0.83**

2.8 Related Work

MobiPurpose is mainly related to three major approaches to characterize mobile data access and disclosure: static analysis, dynamic analysis, and network analysis. We organize the relevant work in Table 2.11 & 2.12, and discuss these areas in detail below.

Network Analysis. Many commercial tools [83, 125, 338, 388] have been developed to support analysis of network flows and identification of leaks of potentially sensitive data. For example, VIP Defense [83] and Forcepoint [125] can detect and block leaks of personal information (e.g., credit card number and social security number) automatically.

Table 2.11: Network analysis research at different granularities using different techniques

		Traffic request	Key-value Pair
Network Analysis	Differential test	Zhang et al. [403], AntMonitor [210]	PrivacyOracle [189], PrivacyProxy [342]
	Text-based	PrivacyGuard [340], Zang et al. [399]	Recon [314], **MobiPurpose**

Past projects have mainly looked at two granularities of network flows: individual traffic requests [304, 403] or specific data fields inside a request. Zhang et al. [403] is an example of the first type, which labels the entire network request as legitimate if the request is associated with a UI event. In contrast, projects like Recon [314] parse the request body into key-value pairs and check if the request contains some specific types of privacy-sensitive data.

While looking at data egress is a good vantage point, searching for privacy-sensitive data in large quantities of network data is still challenging. Black box differential testing is commonly used to tackle this issue [71, 189, 342]. It first establishes a network behavior baseline of an application. It then modifies some specific data on the device and detects leaks by observing deviations in the resulting network traffic. However, this approach is mostly limited to recognizing personally identifiable information and does not generalize to other kinds of sensitive data (e.g., device model, screen resolution, etc.). An alternative choice is leveraging text patterns in traffic requests, which use either hard-coded regular expressions [210, 340] or supervised machine learning approaches [314] to classify network flows.

In contrast, MobiPurpose analyzes the key-value pairs inside a traffic request by leveraging text patterns [314] as well as external knowledge (e.g., app description, domain owner business, etc.). More fundamentally, we not only categorize the data types in a fine-grained manner, but more importantly, classify the purpose of each key-value pair.

Dynamic and Static Analysis of Mobile Apps. Dynamic analysis tracks the flow of privacy sensitive data at runtime. By modifying the device OS, solutions like TaintDroid [113] can label (taint) data from privacy-sensitive sources and transitively apply these labels as the data propagates through program variables, files,

and interprocess messages. In this way, the system can track the data flow, namely which app transmits the data and where the data is being sent to. However, this approach suffers from large false positives (due to coarse tainting granularity) and false negatives (e.g., the data does not leave the device) detection issues. The significant runtime overheads and the compatibility of modified OS are also barriers to widespread use.

Table 2.12: Using dynamic and static analysis to analyze mobile privacy at different granularities.

	App	Library	Permission	Data flow
Dynamic Analysis		Chitkara et al.[61] Han et al. [149]		PmP [4], TaintDroid [113] AppsPlayground [302]
Static Analysis	PrivacyGrade [205] CHABADA [138]	Amandroid [373]	Pscout [23], WHYPER [288] AutoCog [299], ASPG [368] Wang et al. [367]	PiOS [107] DroidJust [56]

Static analysis detects *potential* privacy leaks through analyzing the source code (e.g., permission file analysis [23, 205, 225], data flow analysis [56, 107], or text pattern analysis [367]). For example, PrivacyGrade [205] inspects the permission files and app descriptions to identify unnecessary permissions, since these permissions may lead to privacy leaks eventually. This approach is designed to be more scalable than dynamic analysis since it can avoid runtime overhead without code execution [217]. However, it is still hard to infer the privacy context using static analysis, e.g., where the data is sent to. Besides, running symbolic execution is very time-intensive, and loading code dynamically is increasingly common [232].

Past research has used these two approaches across a range of different granularities (see Table 2.12). For example, PrivacyGrade [205] grades the privacy behavior of each app by measuring the gap between people's expectations and the app's actual behavior. Chitkara et al. [61] use code injection to infer whether the data access is by a third-party library or by the app itself for its functionality. To justify if the data access is legitimate, DroidJust [56] uses static taint analyses to link each data flow with certain application function.

Both approaches can be complementary to the network analysis approach [210, 314]. For example, AntMonitor [210] uses the dynamic analysis to cross-check with the result of network analysis. MobiPurpose leverages the idea of static analysis in feature extractions (Sec. 2.4). We decompile the app file into source code, and count the number of occurrences of domain names across different apps to infer the information of the domain owners.

2.9 Limitations

Network Tracing Coverage. Gathering large-scale comprehensive data from mobile apps is a challenging task [218, 304]. The UI monkeys approach used in our paper can collect a large dataset with relatively small computing resources [48, 223]; however, it also suffers from the issue of lacking comprehensiveness.

For example, text entry box validation (e.g., log in screens) can impede the monkey's progress through an app [304, 316]. In our experiment, we also find monkeys cannot parse the Unity[6] interface in some game apps because it's hard to parse the UI element tree.

We have implemented several heuristics[7] to mitigate these issues. First, we manually login into the popular apps (e.g., Yelp, Twitter, Google) to avoid the login screens in these apps. Second, we detect third-party login options (e.g., log in with your Google account) and select these options if available. A recent study shows that more than 40% mobile login screens support third-party login functionalities [236]. Third, our monkey clicks different coordinates randomly if the app interface is written in Unity.

While developing a better monkey alternative is beyond the scope of this paper, we conducted a preliminary experiment to understand the network tracing coverage. We selected the top 20 popular free apps and manually logged into the apps before the study. We then tested the apps under 3 conditions: manual interaction (3 minutes), monkey interaction (3 minutes), and monkey interaction (15 minutes). We tested the monkey interaction at a longer duration since the monkey duration can be scaled up with little extra resources in practice.

Table 2.13: The Mean (SD) of detected unique network API traces per app under different conditions.

	Manual interaction (3 min)	Bot interaction (3 min)	Bot interaction (15 min)
# of unique requests	45.2 (20.3)	33.1 (16.7)	40.8 (23.5)
# of unique purposes	5.4 (2.2)	3.1 (1.8)	3.9 (1.9)

Table 2.13 illustrates the statistics of unique network API traces detected across different conditions. In general, manual interaction has the best coverage regarding unique requests and unique purposes. The coverage of monkey interaction will increase if we extend the test duration (15 min vs. 3 min). The network traces observed for the same app in 3 conditions are not fully overlapped, so we cannot compare the recall directly. We empirically noticed that monkey interaction generates less diverse traffic than manual interaction, and we observed more ad/analytic traffic requests in the monkey interaction dataset. Future works may consider incorporating app interface semantic knowledge [84, 217] to improve the bot implementation.

Apps Requiring Login. To further understand the login limitation, we studied how much the login blocking will impact the bot interaction. We manually installed the top 80 free apps in US Android stores, found 41 of them support login and 22 of them support Google/Facebook/Twitter login. Among the remaining 19 apps, 15 apps (e.g., Chase Mobile, Robinhood) requires a critical login to interact with core functionalities. Most critical login apps appear in the top 40 apps (13/15), while only 2 apps between 40-80 require a critical login. Modern mobile app design guidelines often suggest registration is a road-block to adoption [26].

[6] https://unity3d.com/

[7] Open sourced at https://github.com/CMUChimpsLab/MobiPurpose

Forcing registration too early can cause more than 85% of users to abandon the product [318]. We expect to see fewer apps require a critical login in "lower-ranked" apps.

Beyond the technical perspective, automated logins with fake profiles can be a legal grey zone, which may raise two legal concerns: 1) packet sniffing while accepting terms and conditions and 2) login with fake profiles. First, logging into a service generally requires accepting terms and conditions, which may ban the packet sniffing analysis. The banning of packet sniffing appeared in the early End-User License Agreement (EULA) and received strong opposition [265]. Recent EULA templates [351] and Terms of Service employed by major apps (e.g., Google, Facebook, Yelp) only prohibit source code reverse engineering. Second, login with fake profiles has been heavily used in web/mobile data scraping [78]. Most recently, a US court ruled that creating fake profiles can be protected by The First Amendment [242]. Future work should pay attention to these potential legal implications as well.

Traffic Obfuscation. Our premise is that mobile network traffic textual data, the app source, and the domain information reflect underlying developer purposes. However, developers might deliberately or unintentionally have obfuscated names. For example, we noticed some apps have unclear key names like "v2", "c12", and these key names might not be consistent across different traffic requests. We found 218 out of 12,046 (1.8%) domains and 405 out of 14,910 (2.7%) apps have some form of obfuscation.

Certificate Pinning. Our current implementation of MITM SSL injection cannot intercept certificate pinned traffic. Using modified OS that disables SSL certificate checking at the system level can potentially mitigate that issue [375]. In our early experiments, we found that this approach only works for the apps developed with Android SSL libraries. So we decided to use the original OS for better compatibility. To quantify the problem of certificate pinning, we selected the top 500 free apps from Google Play retrieved on March 13th, 2018 and found only 22 apps (<5%) used certificate-pinning on all network requests [342].

2.10 Discussion

Theoretical Framework: Contextual Integrity. Our work can be thought of helping contribute to the growing body on privacy as contextual integrity [271], which argues that a data access or disclosure is legitimate based on the specific context and norms in which the information flow happens. In 2012, the White House further espoused this framing in the Privacy Bill of Rights [165]: "*consumers have a right to expect that companies will collect, use, and disclose personal data in ways that are consistent with the context in which consumers provide the data.*"

Some of the ideas behind contextual integrity have started to diffuse into research in mobile computing (either deliberately or through convergent evolution). One example is the app permission system (Fig.1). Researchers have also started to look at ways to incorporate users' privacy expectations [19, 215, 225, 288, 299, 367]. For example, WHYPER [288] and AutoCog [299] build a machine learning model to determine

if the purposes of permissions are consistent with the app description. ASPG [368] and CHABADA [138] identify outlier permissions by clustering similar apps based on app descriptions. Beyond binary anomalous detection, Lin et al. [225] and Wang et al. [367] framed mobile privacy in the form of people's expectation about the data collection purpose.

There has been a rich literature in studying the benefits of understanding "why" [241, 272, 363] in network traffic. For example, Van Kleek et al. [363] manually labeled the network traffic purposes and found purposes can help users make confident and consistent choices. In contrast to prior work, MobiPurpose is the first solution that can automate the mobile traffic purpose inference.

Taxonomy Motivation & Completeness. Our taxonomy does not capture if a data disclosure is legitimate or malicious. The question of whether a traffic request is legitimate or not can only be answered in each specific context in which the question arises. But attempts to answer this question are challenging because of confusion about defining the troublesome activities that fall under the rubric of privacy. Our taxonomy will aid us in analyzing various privacy problems so we can better address them and balance them with opposing interests.

We have created a taxonomy of 16 privacy-sensitive data types and 76 purposes, and 10 participants used this taxonomy to label 1059 instances. While our taxonomy is good enough for our purpose, it is possible that there are other purposes that we did not find. Further, depending on how purposes are used, our taxonomy might be too fine-grained or too coarse-grained. For example, we can further branch the data analytics purpose into developer analytics (e.g., crash reports) and marketing analytics (e.g., marketing attribution analysis). However, we believe that the proposed approach should generalize for new purposes and data types.

Apps from Non-English Speaking Developers. Though we crawled apps from US Android app market, we noticed that many apps are from non-English speaking developers. During data labeling, participants used Google Translate to understand the app functions and the developer's business type. One challenge of the inference pipeline is that developers with different language backgrounds may have different naming conventions. The bootstrapping algorithm works across different languages since it only identifies repetitive patterns [258]. For example, different traffic requests will send the same GPS decimal degree values (e.g., 41.40338, 2.17403) with some common key names. These key names will become parts of the bootstrapping extraction. In fact, "weidu", the Chinese pinyin romanization of "latitude," was captured in our experiment.

2.11 Conclusion & Future work

In this chapter, we present the design and implementation of the first system that can automatically categorize data collection purposes of sensitive data in mobile network traffic. The core of our solution is our

data-type dependent purpose taxonomy, in which we enumerate the potential purposes associated with each data type. Given any key-value pair in the traffic request, we first infer the data type using a bootstrapping NLP approach and then use a supervised machine learning approach to predict the data collection purpose. We evaluated our approach using a dataset cross-labeled by ten human experts. Our experiments show that our approach can predict "what" with an average precision of 95% (among 8 unique categories) and "why" with an average precision of 84% (among 19 unique categories).

Past research shows that the privacy indicators that surface the purpose of data collection can help users make privacy decisions [205, 225, 363]. As such, our next step is to scale up our analysis even further and to build a public resource that can help developers, end-users, journalists, and policy makers better understand which app is collecting data about us, what data, where that data is going, and why it is being collected.

Chapter 3

An Analysis of Smart Home Scenarios

To inform the design of MPF, we first studied smart home apps for two reasons. First, users' privacy will suffer more in smart home privacy contexts since these data are more privacy intrusive than web browsing data and mobile data. Second, there is no standard in the smart home space yet, making it more likely to have MPF adopted. We collected over 200 smart home use cases and examined the feasibility and trade-offs of doing pre-processing on a hub. Overall, our analysis suggests that, while it cannot support all smart home scenarios, pre-processing data locally before it leaves a smart home via a small set of operators can indeed support a wide range of smart home apps.

3.1 Method

Since smart home applications are not yet as popular as those on smartphones [202], we chose to study use cases beyond today's commercially available products, sourcing applications through three approaches. First, we conducted a literature review of sensor-based scenarios for smart homes (e.g., [207]). Second, we surveyed the mobile privacy research literature (e.g., [120, 183, 221]) to understand common data use patterns by mobile app developers since many of them apply to IoT scenarios. Third, we conducted design fiction interviews [106] with 7 participants (3 female, 4 male) across different age groups (min=21, max=60, avg=32) to broaden our set of use cases.

Design fiction interview apparatus. Design fiction is a speculative design method, where participants are asked to sketch diegetic prototypes in an imaginary story world. All participants had interacted with multiple smart home products before. For the interview, we presented each participant with a hypothetical scenario[1] and a floor plan of a home, asking each participant to play the role of different family members and brainstorm different smart home functionality they desired. We also asked participants to focus on the

[1] Adapted from [101]

role' s needs, rather than feasibility of sensing or hardware availability.

The hypothetical scenario: *Imagine a family of five members living in a single-family house. Jeff and Amy are husband and wife respectively. Dave and Rob are their eight-year and seven-month-old kids. Jeff' s mother, Jen, helps look after the baby, and Jeff' s brother Sam occasionally visits the house. The family plans to purchase a set of devices to enable different smart home applications.*

The role playing questions: *If you are [Jeff\|Amy\|...], what smart home usages do you want in the [basement\|living room\|...]?*

In total, we gathered over 200 unique smart home use cases. We clustered similar use cases, resulting in 37 smart home apps spanning different sensors and locations in a home (Table 3.1). We then analyzed what data these apps needed to operate. Specifically, we enumerated why these apps need to collect data (e.g., sensing, analytics, system diagnostics), analyzed potential requirements and constraints in using this data (e.g., compute load, proprietary algorithms, business models), and concluded with what we felt were reasonable outgoing data granularity for each data collection purpose. For example, suppose that the developer of a video doorbell app wants to build an online album for visitors to the home. A reasonable data requirement of this app is face images. We enumerated these data requirements for different purposes in Table 3.2, organizing them by data type.

3.2 Results

We make the following key observations. First, **most cloud services do not need raw sensor or log data**. For example, an app that uses images to detect potential water leaks likely only needs the parts of the original image showing the floor. A smart lighting app which needs to infer the brightness of a room from a camera only needs derived brightness values rather than the raw image. Table 3.2 offers a quantitative view of data granularity requirements across various scenarios. Only 4 out of 37 camera usages require raw data, while the rest only need either derived or partial data. Beyond cameras, we found that only 19 out of 61 cloud services require raw data, while 8 were scalar values (e.g., a binary door status sensor).

Second, **most pre-processing functions share similar data-agnostic data actions**, suggesting the feasibility of using a small set of reusable operators to replace these repetitive pre-processing implementations. At first blush, the number of pre-processing functions appears huge, since we need to cover various data types, output data granularity (e.g., face, objects, audio events, abnormal data), and filter/transform operations (e.g., image cropping, audio spectrum extraction). However, by enumerating the 200+ smart home uses, we find that there exists consistent, simple, and data-agnostic semantics behind most of the functions. In Table 3.2, we categorize all the output data granularity into two classes. *Partial original data* is a subset of the raw sensor data in the original data representation, such as the parts of an image with faces extracted, audio recording segments containing human speech only, and a column in a table. *Derived data* is computed

from the raw data, often resulting in a new data representation (e.g., the keypoints of a human pose). We use two verbs, "select" and "extract," to summarize all the functions that output "partial original" and "derived data," respectively.

Third, **many of these pre-processing functions are lightweight and non-proprietary**, suggesting that they can be run on a low-end smart home hub. For example, pre-trained machine learning models like face detection, functions like inferring brightness, as well as database-like operations on tabular data (project, select, sum, average) are relatively straightforward, openly available, and fast to compute.

Table 3.1: A summary of collected smart home use cases drawn from the research literature and design fiction interviews we conducted. We obtained the initial list of use cases from the design fiction interview, which asked participants to brainstorm use cases in each room using a single-family house floor plan. We then augmented this list by enumerating different sensors supporting these applications and potential data collections that developers may need. This list is not exhaustive, but our goal is to have a large sample covering many common smart home use cases.

ID	Description (Location)	Relevant sensors
#1	Water leak detection on the floor (Basement, Kitchen, Garden, Bathroom)	Camera, humidity, microphone
#2	Home inventory tracking (Storage room, Closet)	Camera, RFID
#3	Toxic Gas Alarm (Basement, Kitchen)	Specialized sensors
#4	Temperature tracking across rooms (The whole house)	Themometer
#5	Garage usage detection (Garage, garden)	Camera, occupancy, microphone
#6	Street parking spots detection (Driveway)	Camera, occupancy, radar
#7	Home arrival recording/prediction (Entrance doors, Garage)	Camera, RFID
#8	Garden irrigation tracking (Garden, driveway)	Camera, humidity
#9	Soil health tracking (Garden, driveway)	Specialized sensors
#10	Bath room activity (e.g., toileting for elderly) tracking (Bathrooms)	Humidity, occupancy, camera, pressure
#11	Smart Speaker & Voice control TV (Living rooms, Bedrooms)	Microphone, proximity
#12	Room occupancy statistics (The whole house)	Camera, occupancy, microphone
#13	Automatic temperature control based on the occupancy/identity (The whole house)	Camera, occupancy, microphone
#14	Office productivity tracking (Office room)	Camera
#15	Personalizing welcome message for visitors (Entrance doors, garage)	Camera
#16	Package delivery detection (Entrance doors)	Camera
#17	Fall detection for the elderly (The whole house)	Camera, microphone
#18	Automatic photography [5] (Living room, Kitchen, Garden)	Camera
#19	Automatic lighting based on occupancy and light intensity (The whole house)	Occupancy, light sensors, camera
#20	Wanted criminal search on the street (Entrance doors)	Camera
#21	Detecting strangers when no one is at home (The whole house)	Camera
#22	Ice detection (Entrance stairs, driveway)	Camera, specialized sensors
#23	Sunshine tracking for plants (Terrace, garden)	Camera, light sensors
#24	Detecting messiness of the home and calling a cleaning service (Living room, kitchen, closet)	Camera
#25	Smart cooking (Kitchen)	Smart appliances
#26	Freezer ice cleaning reminder (Kitchen)	Smart appliances
#27	Appliances electricity consumption statistics (The whole house)	Smart appliances and plugs
#28	Sleep tracking and sleep quality measurement (Bedrooms)	Camera, microphones, pressure sensors
#29	Smart toilet recognizes butt and analyzes poop for diseases [289] (Toilets)	Camera, pressure sensors

Table 3.2: What our team felt were reasonable outgoing data granularities for different smart home scenarios. We cluster these scenarios by their input and the output. If the output is Original, it implies no pre-processing functions can be done on the hub for these scenarios. The "(#scenario)" is the number of scenarios for each corresponding category, and the numbers in the "scenarios" column refer to the scenario id in Table 3.1. Since a use case may have multiple reasonable app designs, the use case may be counted more than once.

Data type (# scenarios)	Pre-processed output (# scenarios)	Example usage	Scenarios
Image/Video (27)	**Original**		
	- raw video 1	Enabling online security camera album	#21
	- raw image 3	Enabling automatic photography (e.g. Google Clips [137])	#18
		Recognizing special activities (e.g., garden irrigation)	#8, #29
	Partial original		
	- only person/face 10	Recognizing human relevant activities (e.g., fall, sleep)	#7, #12, #13, #14, #17, #20, #21, #28, #30, #32
	- excluding person 7	Determining home messiness	#1, #2, #5, #19, #23, #31, #34
	- particular objects 3	Detecting certain objects (e.g., package)	#16, #22, #33
	Derived		
	- identity 2	Personalizing the welcome message	#7, #15
	- pose 2	Quantifying work activities	#14, #17
	- light intensity 2	Determining the lightness of the environment	#19, #23,
Audio (9)	**Original**		
	- raw audio 2	Supporting signal processing at per-frame level	#36, #37
	1	Supporting phone call, audio diary	#11
	Partial original		
	- voice audio 1	Supporting speech interaction	#11
	- activity sound 3	Detecting garage events	#5, #28, #30
	Derived		
	- speech text 1	Recognizing command intents	#11
	- FFT/MFCC 2	Detecting occupancy	#12, #13
	- audio events 3	Sending notification to owners when the dog barks	#12, #13, #33
Tabular (e.g., Channel state information) (5)	**Original**		
	- complete CSI 1	Enabling fine-grained sensing tasks using the Wi-Fi signals	#32, #34
	Partial original		
	- only UUID 3	Tracking the food/cloth inventory	#2, #7, #30
	Derived		
	- distance/positions 1	Helping users find the item in home	#34

Chapter 4

An Analysis of Smart City Applications

We also studied smart city applications to generalize our findings in Chapter 3. Overall, our analysis suggests similar findings: (1) most cloud services do not need raw individual data, but aggregated data; (2) most data transformations share similar data-agnostic data actions; (3) many of the data transformations are lightweight and non-proprietary. These findings motivate and inform the design of MPF.

4.1 Method

We first collected 80 unique smart city use cases (see Table 4.1) through a literature review [85, 224, 260, 290, 293, 325]. We then examined the feasibility of realizing these use cases through distributed personal sensors. For example, personal video doorbells can potentially count the number of available street parking spots, and built-in microphones in doorbells or smart speakers may measure the noise level in the residential area and detect gunshot events through acoustic trilateration. Finally, we analyzed potential privacy risks with each application and explored solutions that can potentially mitigate these risks.

4.2 Results

We make the following key observations. First, **many smart city applications only need collective insights rather than insights about an individual.** For example, a user may want to know the general availability of street parking spots near a restaurant rather than specific spots. Or, the city council may want to know how busy the parking spots are, so they can decide whether to allocate more parking space. Parking occupancy apps that only output some approximate aggregated information (e.g., whether the street parking is busy) can offer significant value to the society. Meanwhile, such an app can mitigate privacy risks for individual sensor owners and bystanders on the streets. This finding motivates us to explore data aggregations that can balance utility with privacy.

Second, across applications, **we can divide the data aggregation process into two stages: `Map` and `Aggregate`.** Real-world sensor networks involve many multimedia sensors (e.g., cameras, microphones) and various forms of analysis over accumulated data (e.g., the TV search queries in the past month), offering numerous application possibilities. To derive collective insights from these data streams, developers often need to first **map** high dimensional data from one individual source to a set of intermediate key/value pairs and then **aggregate** intermediate results from spatially distributed sources to generate meaningful and privacy-sensitive insights. This abstraction is similar in spirit to the well-known MapReduce programming framework [82], which has proved to be expressive for many real world tasks.

Third, **a small set of `Map` programs can cover a surprisingly large number of scenarios, while `Aggregate` functions are mostly repetitive and lightweight.** For example, a simple `Map` that counts available parking spots can enable numerous applications, such as pulling real-time parking availability, generating availability histogram, and predicting future parking availability. Meanwhile, *Aggregate* functions (e.g., sum, average, tilateration, heatmap, histogram) mostly aggregate data across spatial and temporal domains. Another way to view these *Aggregate* functions is that they are similar to SQL extensions in spatial-temporal databases [115], where database researchers found that a constrained set of programming APIs can support many spatial-temporal analyses.

Fourth, **a well-articulated data aggregation task should specify how a system should compute the intermediate results and store them in the database.** Suppose we implement the parking occupancy histogram app using the MapAggregate abstraction. We need to articulate that the system should pull raw images from video doorbells every 60 minutes, `Map` the images to occupancy key-value pairs, and then `Aggregate` the key-value pairs. Further, the system should delete data older than a year since the city council only wants to know the parking occupancy in the past year. This means that both computation and data storage/retention policies must be provided as part of the data aggregation specifications. In MapAggregate, developers only specify two data transformation functions (i.e., `Map` and `Aggregate`). The transformations in the manifest are written as if the entire pipeline where stateless – i.e. no stored/retained data anywhere. In practice, however, data is indeed stored within the private clouds of users to cache outputs of `Map` however, only with the intention of improving latency and avoiding repeated computation. The runtime manages data retention and application logic automatically by analyzing the manifest, while developers remain oblivious to any data storage/caching, giving them an illusion of a system that is overall stateless.

Table 4.1: A summary of collected smart city use cases drawn from the research literature [85, 224, 260, 290, 293, 325]. We clustered these applications by the aggregation types and enumerate a few non-aggregtable apps.

Aggregation types	Applications
Count/Sum/Average	Electricity usage (sum/average), water consumption (sum/average), heat consumption (sum/average), water leaks (sum/average), fire alerts (average), CO2 emission tracking (sum), power-efficient appliances (sum/percentage), indoor vs. outdoor water usage (group sum), building vs. home energy efficiency (group sum), electric vehicle usage (sum), waste management (sum), renewable energy (sum), rainwater collection (sum), public transport tracking (count), intersection traffic (count), residents physical activity (average), pedestrian entrance tracking (count), pedestrian foot traffic (count), user mobility (count), time spent asleep (average), number of users that pass by a billboard (count), DMV visitor counting (GPS), fire department delay (average), traffic light wait time (average), irrigation tracking (sum/average), electromagnetic field level monitoring (sum/average), food consumption (sum/average), indoor home temperature (average), parking occupancy (count/average/sum)
Rank/Median/Top X	Snow plowing fairness, public space noise level monitoring
Histogram/Heatmap	Network speed monitoring (group average), water quality monitoring (group average), food consumption (group average), street light brightness (group average), noisiest neighborhoods (group average)
Route	Delivery route with best parking availability, Smart garbage pickup route, least-polluted biking path
TF-IDF	Determine restaurant type for region, trending TV search queries
Trilateration	Gunshot localization, air pollution source localization
Time-series forecasting	Parking occupancy prediction
Non-aggregatable apps	Searching for full trash cart, On-demand rides with autonomous vehicles, DMV visitors Counting through a camera, Garbage collection prediction

Table 4.2: A small set of `Map` programs can cover a surprisingly large number of scenarios. Note that this table is not an exhaustive list of `Map` functions an app store should support. In practice, an app store can incrementally add new `Map` functions, expanding the platform's capability.

Map functions	**Input**	**Output (Key:Value)**	**Example Applications**
Visual object counting	Image	Object: count	Counting people, vehicles, parking spots.
Audio event detection	Audio	Event: high precision timestamp	Gunshot localization, Fire alarm tracking
Term frequency	Text	Word: Count	Trending Smart TV Search Queries
Tabular temporal aggregation	Tabular	Duration: Number	Water/heat/electricity consumption

Table 4.3: `Aggregate` functions are mostly repetitive and lightweight. We clustered these applications by the aggregation types and identified the following categories. Note that this list may not be exhaustive as well.

Aggregation functions	Example applications
Count/Sum/Average	How frequently are the parking spots used in this region in the past year?
Rank/Median/Top X	Which roads are cleared first after a winter storm?
Histogram/Heatmap	How does the noise distribute in the neighborhood?
Route	How to optimize the delivery route, so drivers are more likely to find parking spots?
TF-IDF	What are the trending TV search queries in this region in the past week?
Trilateration	Where is the gunshot?
Time-series forecasting	Would there be parking spots around restaurant X if I arrive in one hour?

Part 2

IMPLEMENTING DATA MINIMIZATION

Chapter 5

Peekaboo: Modular Privacy Flows on a Smart Home Hub

We present Peekaboo, a new privacy-sensitive architecture for smart homes that leverages an in-home hub to pre-process and minimize outgoing data in a structured and enforceable manner *before* sending it to external cloud servers. Peekaboo's key innovations are (1) abstracting common data pre-processing functionality into a small and fixed set of chainable operators, and (2) requiring that developers explicitly declare desired data collection behaviors (e.g., data granularity, destinations, conditions) in an application manifest, which also specifies how the operators are chained together. Given a manifest, Peekaboo assembles and executes a pre-processing pipeline using operators pre-loaded on the hub. In doing so, developers can collect smart home data on a need-to-know basis; third-party auditors can verify data collection behaviors; and the hub itself can offer a number of centralized privacy features to users across apps and devices, without additional effort from app developers. We present the design and implementation of Peekaboo, along with an evaluation of its coverage of smart home scenarios, system performance, data minimization, and example built-in privacy features.

5.1 Introduction

For many smart home products - such as smart speakers, cameras, and thermostats - the "brains" of these systems are typically in the cloud. However, a key concern with cloud-based software architectures is data privacy [366]. It is challenging for users to have any assurances of privacy *after* their sensitive data leaves the confines of their home [243]. Further, it is challenging for companies to *legitimately avoid collecting unnecessary smart home data while also re-assuring users and independent auditors that this is indeed the case*.

For example, imagine a developer of a smart TV claims to only send aggregated viewing history data

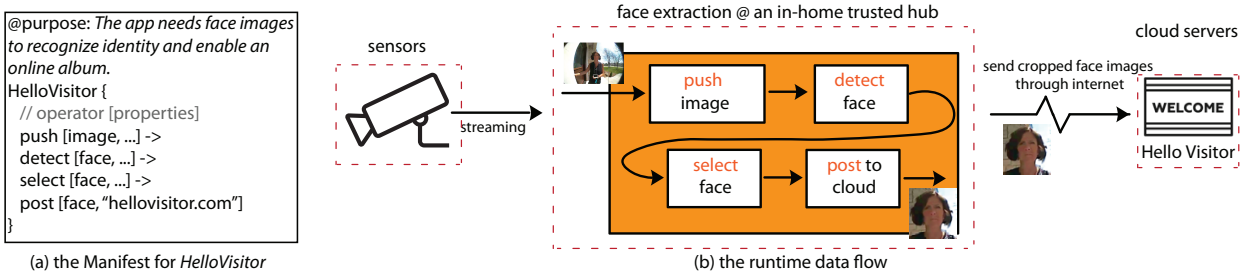


Figure 5.1: The architecture of the "HelloVisitor" app. A developer creates a manifest making use of four chainable operators (i.e., *push*, *detect*, *select*, and *post*) and adds a text string to specify the purpose. Once deployed, a camera streams the video to the hub, which pre-processes the video so that only portions of images with detected faces are sent to cloud servers. Since the semantics of each operator is known, it is easy to analyze the privacy-related behavior of apps (e.g. "this app only sends images of faces to HelloVisitor.com") and modify it if desired.

to their servers once a week. **How can outsiders validate this claim, since the hardware, firmware, and backend servers are proprietary black-boxes?** Today, an independent auditor would need to use arduous reverse engineering techniques to validate devices' data collection behavior [285]. One alternative is for the device to do everything locally without sending any data [11], though given that many devices only have basic CPU and storage capabilities, this approach severely limits the functionality they can provide. Doing everything locally also limits many kinds of rudimentary analytics that users might find acceptable, e.g. developers may want to know how many hours the TV is on per week. As another alternative, the device can aggregate or denature the data itself before sending it out [369, 398], but again there is no easy way to verify this behavior. Yet another alternative is to leverage new mechanisms for trusted cloud computing [133, 198, 212]. However, these approaches are challenging for users and auditors to understand and verify, and do not necessarily perform data minimization before sensitive data leaves one's control.

This chapter introduces Peekaboo, a new privacy-sensitive architecture for developers to build smart home apps. Peekaboo has three key ideas. First, app developers must declare all intended data collection behaviors in a text-based manifest (see Fig. 5.1a), including under what conditions data will be sent outside of the home to cloud services, where that data is being sent to, and the granularity of the data itself. Second, to specify these behaviors, developers choose from a small and **fixed** set of operators with well-defined semantics, authoring a stream-oriented pipeline similar to Unix pipes. This pipeline pre-processes raw data from IoT devices in the home (e.g. sensor data or usage history) into the granularity needed by the cloud service. Third, an in-home trusted Peekaboo hub mediates between all devices in the home and the outside Internet. This hub enforces the declared behaviors in the manifests, and also locally runs all of the operators specified in these manifests to transform raw data before it is relayed to any cloud services. Combined, these ideas make it so that developers can reduce data collection by running pre-processing tasks on the in-home

trusted hub, and users and third-party auditors can inspect data behaviors by analyzing these manifests as well as any actual data flows. Our approach also facilitates a number of privacy features that can be supported by the hub itself, such as adding additional conditions or transformations before data flows out, or transforming parts of the manifest into natural language statements to make it easier for lay people to understand what data will be sent out, when, and to where.

For example, the "HelloVisitor" app (Fig. 5.1) identifies visitors using faces in images. Today's video doorbells often send captured raw photos to the cloud when they detect a scene change. By applying a pre-processing pipeline (i.e., face detection and image cropping), HelloVisitor can avoid sending images of people or vehicles simply passing by, thus minimizing data egress to just what the app needs to operate.

Peekaboo's architecture is based on an analysis (§**??**) we conducted of 200+ smart home scenarios drawn from the research literature and design fiction interviews we conducted. This analysis led to two insights. First, many apps do not need raw sensor or log data, but rather a transformed or refined version. Second, while it cannot support all smart home scenarios, our approach of using a small and fixed set of operators can support a surprisingly large number of scenarios.

We implemented a Peekaboo manifest authoring tool on top of Node-Red, a mature, web-based, visual programming platform [170]. We developed the operators as a set of Node-Red compatible building blocks. Further, we built the Peekaboo runtime, which parses a manifest, retrieves raw sensor or log data from IoT devices, sets up a data pipeline by assembling a chain of operators, and streams the data through this pipeline. Note that Peekaboo operators are device-agnostic, and rely on runtime drivers to handle heterogeneous device APIs (similar to HomeOS [101]). We deployed the Peekaboo runtime using a Raspberry Pi connected to a TPU accelerator.

We conducted detailed experiments to validate the design of Peekaboo. To understand the range and limitations of its architecture, we first implemented 68 different manifests to cover over 200 use cases and analyzed the types of pre-processing in these manifests (§5.6.1). We then used three example manifests to demonstrate the feasibility of using simple algorithms to reduce privacy risks while having little impact on utility (§5.6.2). We built five end-to-end Peekaboo apps, covering 5 data types (video, image, audio, tabular, and scalar) and used these apps to evaluate system performance. We also evaluated the scalability of our low-cost Rasberry Pi setup (≈$100), showing it can support more than 25 inference tasks and 100 filtering transformations per second (§5.6.3). Furthermore, we conducted a user study to test the usability of Peekaboo APIs with 6 developers (§5.6.4). Our results show that developers can understand how to use Peekaboo quickly and complete programming tasks efficiently and correctly. Finally, we demonstrated how Peekaboo's architecture, and the hub specifically, can support three kinds of privacy features across all apps (§5.6.5). Specifically, we show how a static analysis tool can generate natural language descriptions and privacy nutrition labels [111] of behaviors based on operator pipelines, how an arbitrary manifest can be extended with time-based scheduling features, and how rate limiting can be easily added to a data flow.

We make the following contributions in this project:

- A novel software architecture, Peekaboo, that helps developers collect sensitive smart home data in a fine-grained and flexible manner, while also making the process transparent, enforceable, centrally manageable, and extensible.
- A study of over 200 unique smart home use cases to design a taxonomy of reusable pre-processing operators that can feasibly be implemented at the Peekaboo hub to enforce privacy requirements.
- An end-to-end open-source prototype implementation[1] of a Peekaboo hub on a Raspberry PI platform with a TPU accelerator.
- A detailed evaluation of Peekaboo's expressiveness based on coverage of smart home scenarios, system performance, data minimization, and application-independent privacy features.

5.2 Peekaboo Design Overview

Peekaboo has three main components. The first is developer-specified **manifests** that declare **all** data pre-processing pipelines. Note that the simplest manifest can just describe getting data from IoT devices and sending it to the cloud if no pre-processing is needed. The second is a **fixed** set of reusable and chainable **operators** to specify these pipelines. The third is an in-home trusted **hub** that enforces these manifests and mediates access between edge devices and the wider Internet. We present more details about our design rationale and tradeoffs below.

Peekaboo's manifest is a natural evolution of Android permissions [122, 263] and Manufacturer Usage Description (MUD) whitelists [63] for IoT devices. In Android, developers must explicitly declare permissions in an app's manifest so that it can access protected resources (e.g., location or SMS messages). Similarly, MUDs allow IoT device makers to declare a device's intended communication patterns. The rationale is that many IoT devices are expected to communicate with only a few remote servers known *a priori*, and so declaring the device's behaviors allows the network to blacklist unknown traffic requests.

However, a weakness with Android's permission system is that it is binary all-or-nothing access. For instance, an app developer might need to access SMS messages from just one phone number for two-factor authentication, but Android only offers access to all SMS messages or none. Furthermore, while the developer can display text in the app or in a privacy policy that the app will only access messages from one source once, there is no easy way for a user to verify this behavior. One solution is to offer many fine-grained permissions for every potential use case, but this would lead to an explosion of permissions that would be onerous for developers to program, unwieldy for users to configure, and complex for platform builders to support.

Peekaboo proposes an alternative approach, requiring developers to declare the data collection behavior

[1] `https://github.com/CMUChimpsLab/Peekaboo`

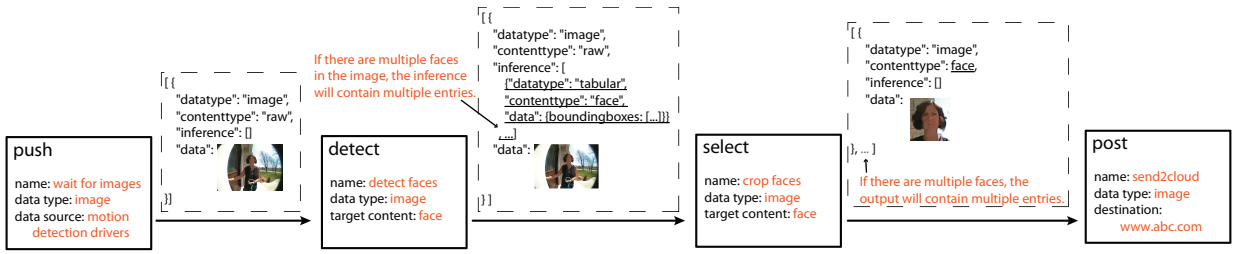


Figure 5.2: The hub program for "HelloVisitor" (Fig. 5.1). Operators come with a few configurable parameters and have clearly defined ways to interact with a uniform data model (dashed boxes). For example, the *detect* operator only modifies the "inference" field. The *select* operator modifies the "data" field based on the "inference" field. We highlight the affected data fields with underscores. Operators are open source to help with verification of their behaviors.

inside a text-based manifest using operators, with the hub only allowing declared flows. With Peekaboo, a user can install a new smart home app by simply downloading a manifest to the hub rather than a binary. This approach offers more flexibility than permissions, as well as a mechanism for enforcement. It also offers users (and auditors) more transparency about a device's behavior, in terms of what data will flow out, at what granularity, where it will go, and under what conditions.

Threat model: We envision that future third-party developers can build and distribute sophisticated ubiquitous computing apps through smart home App Stores, similar to today's app stores for smartphones. However, these smart home app developers, similar to mobile app developers, might deliberately or inadvertently collect more data than is necessary. Peekaboo's goal is to limit data egress by such developers, while also making it easier for users and auditors to verify the intended behaviors and data practices of their IoT apps.

We make the following assumptions: (1) *Trusted Hub:* The Peekaboo Hub, developed by platform providers (e.g., Apple, Google), is trusted and uncompromised. (2) *IoT Devices:* We assume that the Peekaboo hub can isolate IoT devices and only allow whitelisted outgoing data flows, similar to the MUD [63]. All actual IoT Devices are required to send data through the Peekaboo hub, and do not circumvent the Peekaboo hub, either through an independent or covert side-channel. (3) *Operators:* We assume that the operators that we have created are themselves secure and do not have vulnerabilities. The source code of operators will be made open source and allow for verification, audits, and updates as needed. We note that while our threat model assumes that devices within the home are trusted, it does *not* make a similar assumption about cloud services.

5.3 Programming Manifests using Operators

We discuss the design of Peekaboo's manifest and operators, plus the rationale and tradeoffs behind our design. The manifest and operators need to be expressive enough to support a wide range of scenarios,

easy to author for developers, easy to comprehend for auditors, and beneficial for privacy protection.

Peekaboo uses a Pipe-and-Filter software architecture [131], modeling a hub program as a set of connections between a set of stateless operators with known semantics. Figure 5.2 presents a data pre-processing pipeline from the *HelloVisitor* app, which can be abstracted into a directed acyclic graph (DAG) of operators. The first operator, a *push* operator named "wait for images", specifies the raw image retrieval behavior (e.g., pull v.s. push, frequency, resolution) and gets raw image from the Peekaboo runtime when there is a motion event. The second operator, *detect*, annotates the bounding boxes of faces inside the raw image. Next is "crop face", a *select* operator that crops the image to output a set of face images based on annotated bounding boxes. Finally, "send2cloud" is a *post* operator with outgoing network access, which posts cropped faces to a remote cloud service.

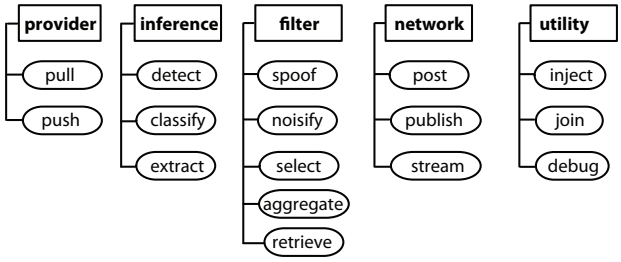


Figure 5.3: The taxonomy of Peekaboo operators. Each operator corresponds to a "verb" statement relative to the operator itself. For example, the *pull* operator pulls data from the hub runtime.

5.3.1 Abstracting Reusable Operators

At a high level, pre-processing pipelines do three things: collect raw data from edge devices, transform that data into the targeted granularity, and send the processed data to external servers. Although the exact desired data actions vary across use cases, the high-level data pre-processing semantics are surprisingly similar across data types. For example, a *noisify* data action denatures the original data slightly by imposing some noise, without changing the original data representation format, such as blurring an image, changing pitch/tempo of audio, or distorting numerical values by a small amount.

We used best practices in API design (e.g. [34]) to guide the design of these operators. We started with a few use cases, programmed them using our initial API, and iterated on the API as we expanded the supported use cases. Based on the data transform behaviors, we created sixteen operators grouped into five categories (Fig. 5.3): `provider`, `inference`, `filter`, `network`, and `utility`. Developers can specify the behavior of an operator by configuring its associated properties.

Inferring and filtering target content. In contrast to the binary all-or-nothing data access control, Peekaboo aims to enable a new fine-grained semantic-based data access control, which requires developers to declare when the app collects data (e.g., when a baby is crying) and what data content would be collected

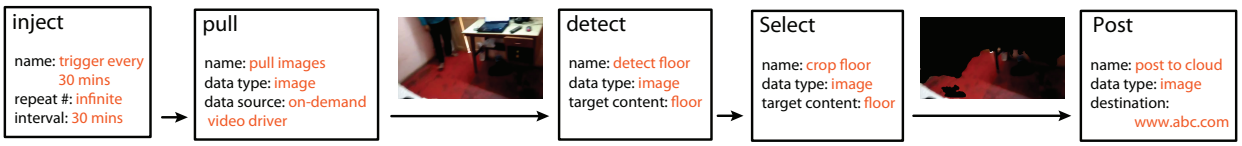


Figure 5.4: A water leak detection app pulls an image from the camera every 30 minutes, detects the floor area using image segmentation algorithms, and sends the image containing only the floor to the server. The app protects users' privacy by only sending the floor pixels to the cloud.

(e.g., face images, speech audios). To achieve this, we introduce two sets of operators: `inference` operators that can annotate data contents (e.g., the bounding boxes of faces, the key points of a body pose), and `filter` operators that filter based on the annotations. For example, *HelloVisitor* first uses a *detect* operator to annotate the bounding boxes of faces and then uses a *select* operator to crop the image based on annotated bounding boxes.

We summarize inference tasks into three primitives: *detecting* instances of objects, *classifying* the dominant content category, and *extracting* derived data (e.g., audio frequency spectrum). We also examined common privacy countermeasures [4, 105, 178, 201] to determine five types of *filter* operators: *spoof* for replacing the payload with an artificial replacement, *noisify* for injecting a configurable random noise, *select* for keeping partial raw data, *aggregate* for summarizing statistics, and *retrieve* for overwriting the payload data with derived data.

The abstractions of operators are somewhat analogous to abstract classes in object-oriented programming, and developers can specify the exact data transformation they want via configuring the properties of each operator. The runtime then maps the manifest specification to the concrete subclass implementations based on the properties. For example, the detect operators in Figs. 5.2 and 5.4 have different target content properties, so they are mapped to two different data transformations. Further, there may be multiple face detection operator implementations, and developers can let the runtime determine which one to use or specify one explicitly. We enumerate supported data transformations in Figure 6.1.

The key property for the inference and filter operators is *target content* (e.g., face in Fig. 5.2 and floor in Fig. 5.4). The target content is a type of semantic annotation supported by integrated inference algorithms, which can then be filtered accordingly using filter operators. Peekaboo currently provides several built-in state-of-the-art inference algorithms using pre-trained machine learning models, which support over 90 visual categories [54, 227] (e.g., face, person, floor, table), 632 audio classes [132] (e.g., baby crying), and many other individual categories (e.g., body pose [228], heart rate [27], audio frequency spectrum, brightness). Developers can choose among these options using a dropdown menu in the authoring interface (Fig. 5.10).

Note that a Peekaboo runtime only supports a fixed set of operators and property options enabled by the pre-loaded implementations. We do not allow operators to be dynamically loaded because we would not

Table 5.1: Supported data transformation across data types.

Data type	Operator	Supported transformations
Video	Extract	Extracting heart rates using [27]
	Retrieve	Keeping inference results but removing the video data
Image	Detect	Detecting the bounding box of 90 Common objects (Microsoft COCO [227]) and faces, Detecting the segmentation areas of 20 common object segmentation (PASCAL VOC2012 [54])
	Extract	Extracting the brightness, body poses [228]
	Noisify	Blurring an image
	Select	Cropping an image based on bounding boxes or a segmentation areas
	Retrieve	Keeping inference results but removing the image data
Audio	Detect	Detecting voice activity windows (i.e., starting time and ending time)
	Classify	Recognizing 632 audio events (AudioSet [132])
	Extract	Extracting frequency spectrum, speech text
	Noisify	Injecting a configurable random variation to the pitch and tempo.
	Retrieve	Keeping inference results but removing the audio data
Tabular	Select	Selecting a column with an optional where clause
	Aggregate	Aggregating (i.e., sum, count, average) the tabular entries by one field and projecting the output to a designated field.
Scalar	Classify	Comparing a value with a threshold
	Aggregate	Computing the sum, count, average of matched scalar items.
	Retrieve	Keeping inference results but removing the original scalar data

know the semantics of that operator, and it may have undesired behaviors.

Collecting raw and relaying processed data. To install a Peekaboo app, users need to bind the manifest to compatible devices, similar to how users install a SmartThings App [323] today. Developers have to specify required device drivers in `provider` operators (i.e., *push* and *pull*), so the hub runtime can determine compatible devices.

Here, *push* and *pull* represent two styles of data access: passively waiting for pushed data from drivers and actively pulling from drivers. For example, *HelloVisitor* (Fig. 5.2) starts with a *push* operator. When there is a significant change in the visual scene, the Peekaboo runtime sends an image to the *push* operator to trigger subsequent operators. In contrast, Fig 5.4 shows the hub program of a *water leak detection* app, which pulls an image from the camera every 30 minutes.

Finally, network operators are the only group of operators that can send data outside. Peekaboo currently has three operators: *post* for HTTP/S post, *publish* for MQTT Pub/Sub, and *stream* for RTSP video streaming. Developers can configure the `provider` and `network` operators to enable SSL connections between the IoT devices and the hub, and between the hub and remote servers, similar to the Network security configuration in Android [93].

5.3.2 Chaining Operators Together

In this section, we present more details on how operators are connected together.

Data model. Peekaboo uses a uniform data structure between operators (see Figures 5.2 and 5.4). The basic data structure (i.e., a Peekaboo data item) is a map, and the message transmitted between operators is an array of such items. An example map is shown below.

$$\begin{aligned}
\{&\text{“}datatype\text{”} : \text{“}video\,|\,image\,|\,audio\,|\,tabular\,|\,scalar\text{”},\\
&\text{“}contenttype\text{”} : \text{“}raw\,|\,face\,|\,person\,|\,dog\,|\,speech\,|\,...\text{”},\\
&\text{“}inference\text{”} : [\{...\}\,|\,\{...\}\,|\,...],\\
&\text{“}data\text{”} : \{raw\,sensor\,data\},\\
&\text{“}process\text{”} : \{device\,information, operation\,history\}\}
\end{aligned}$$

A Peekaboo data item only stores one unit of the data (e.g., an image, audio file, tabular row, or scalar value), while the inference field contains a list of annotations. Suppose the input of *HelloVisitor* is an image with multiple faces. Here, *detect face* will write multiple face annotations to the inference field, and each face annotation is a tabular Peekaboo data item. *crop face* will then generate a list of Peekaboo data items, where each corresponds to one face image (Fig. 5.2).

Composing pipelines. To build a data pre-processing pipeline, developers connect operators' outputs to others' inputs and assemble them into a directed acyclic graph. All operators, except *join* (described more below), take only input from one prior operator. This design avoids synchronization issues since the execution of operators is asynchronous.

In contrast, developers can connect an operator's output to multiple operators' inputs. The preceding operator creates a copy of the output message for each connection to avoid interference and potential multithreading problems.

Supporting more complex logic. Figs. 5.2 and 5.4 show two simple pipelines. However, these linear pipelines are insufficient for many applications. Imagine a baby monitoring app that only sends input audio to an external server when the hub program detects a baby crying (Fig. 5.5). With a linear pipeline this is infeasible since when we use a *retrieve* operator to check if the audio contains baby crying, the subsequent

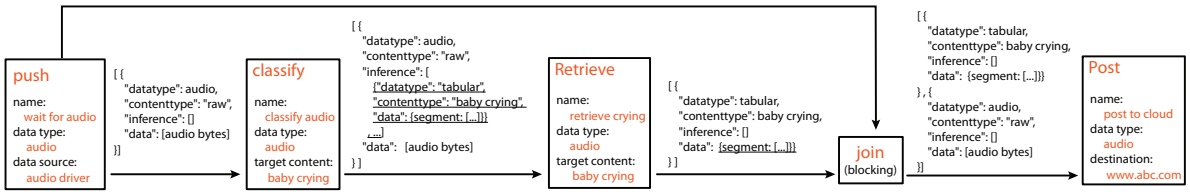


Figure 5.5: A baby monitoring app only sends audio containing a crying sound to the server. The *retrieve* operator replaces the "data" field with the "inference" field. The *join* operator merges the outputs from *wait for audio* and *retrieve crying*, passing its output onward when both input streams arrive. Finally, the *post* operator only sends the audio data item outside, since its data type parameter is "audio".

operators no longer have access to the original data. More fundamentally, this is a common constraint of many dataflow programming models, where the data itself controls the program's flow.

We address this limitation by introducing the *join* operator to support AND, OR, NOT logic. *Join* is the only operator that can take input from multiple operators and fuse asynchronous data flows into one pipeline. A key property for *join* is whether it is blocking or non-blocking. A non-blocking *join* forwards incoming data whenever it becomes available, equivalent to an **OR** logical operator. In contrast, a blocking *join* only merges and forwards incoming messages if they meet specified criteria (e.g., all the incoming messages arrive within a small time window). Note that incoming messages can be from the same prior operator. For example, a blocking *join* can block the data flow until there are multiple "baby crying" events, to confirm that a certain accuracy level is met.

5.3.3 Error Handling & Debugging Support

It is possible for the flow between operators to be Peekaboo data items with different data types due to flow fusion. Without careful design, connecting arbitrary operators together may result in unpredictable errors. An essential property of our API design is that each operator is associated with a target data type. An operator only processes corresponding data items selectively, so the operators will not try to detect faces in a scalar value object.

Another important design issue is that *inference* and *filter* operators handle unmatched data types differently. Inference operators will leave unmatched items untouched, while filter operators will filter them out. This design strengthens Peekaboo's annotation capability since developers now can apply multiple inference operators to the same data item. Meanwhile, it enhances privacy as well: data can flow to the next operator only if the developer matches the data type explicitly.

To help with debugging, developers can append a *debug* operator after any operator, which will print output data to a console window. We also incorporated example data (e.g., test videos, photos, audios, and tabular data) as options of the *pull* operator and designed the *inject* operator to support both manual and interval triggers, so developers can quickly test pipelines within the editor without actual hardware.

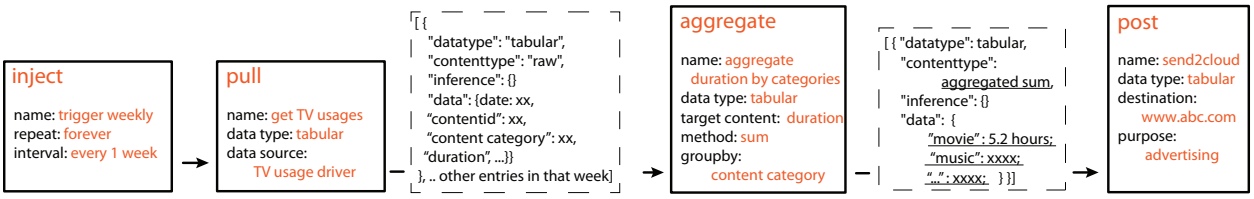


Figure 5.6: A SmartTV usage summary app queries users' watching patterns and uses the data to generate a weekly summary for video consumption across different channels and categories. The *aggregate* operator works similarly to SQL aggregation and replaces the value in the "data" field.

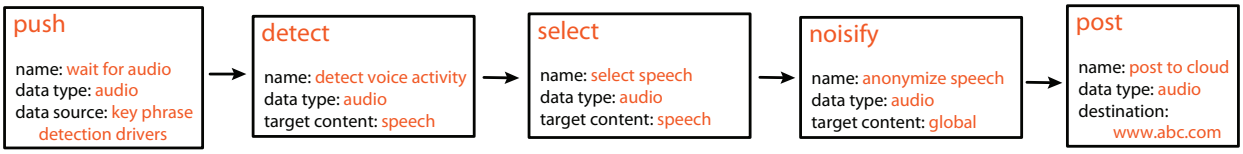


Figure 5.7: The incognito voice assistant only collects short audio segments containing speech and hides the speaker's identity by changing the pitch. This hub program can protect users' identity without breaking the speech recognition functionality.

5.4 Illustrating Pre-processing using Examples

We present three example manifests to illustrate the use of Peekaboo's APIs.

Smart TV logs. A smart TV developer is interested in collecting users' viewing history for advertising purposes. This smart TV stores viewing history in a simple tabular form. Here, we assume developers want to do better with respect to privacy, perhaps for legal compliance, market competition, or because hubs like Peekaboo are widely adopted in the future and companies want to assure customers that they are only collecting minimal data. Fig. 5.6 shows an example pipeline we built to show how a developer can compute an aggregate view of video consumption on the hub, thus sending out a less sensitive summary. This example also shows Peekaboo's support for database-like queries (e.g., SELECT, AGGREGATE, JOIN, WHERE), which also work in other tasks (e.g., counting people in an image).

Incognito voice assistant. Fig. 5.7 presents a manifest of a smart voice assistant that we developed, which can offer a speech anonymization feature that protects users from exposing undesired voice fingerprints. In contrast to Google's "Guest Mode" [136], this manifest can **assure** users that their voice fingerprint identities are protected.

Beyond database-like queries, Peekaboo introduces two important extensions, content-based selection and explicit noise injection, to accommodate the smart home context and privacy-preserving goals. This example illustrates how the combination of inference and filter operators can filter out non-speech audio segments. Further, this example uses a *noisify* operator to hide speaker identity, which changes the tempos

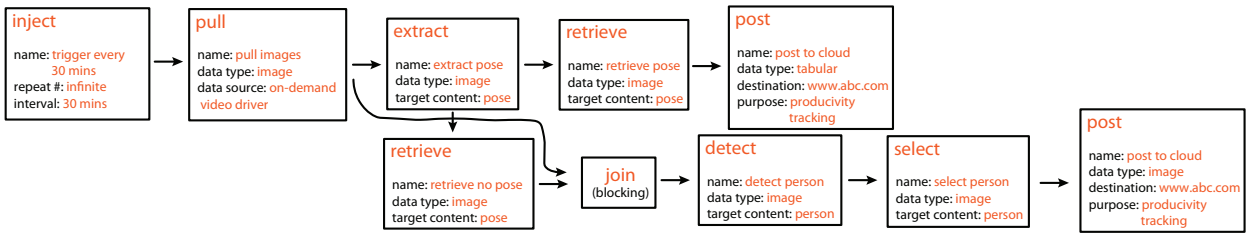


Figure 5.8: The productivity tracking app pulls an image from the camera every 30 minutes, analyzes the pose of the person, and uploads the pose to the server for further analysis. Pose information is usually unavailable when the person sits at a desk. The app then only sends the person pixels to the cloud with more sophisticated algorithms.

and pitch of the captured audio with a configurable random variation (e.g., 5%).

Productivity tracking. PC Applications like RescueTime [315] help users be more productive by helping them understand how they spend their time. As working from home becomes increasingly common, we envision a smart home version that track a person's productivity beyond PCs. Implementing such an app in a conventional cloud-based architecture can be worrisome due to privacy concerns similar to those in the voice assistant app. Fig. 5.8 presents an example pre-processing pipeline of a productivity tracking app using a camera, which transmits extracted poses to the cloud.

This example shows use of Peekaboo's flexible conditional flow control. Peekaboo operators support an intrinsic condition flow control similar to Unix Pipes. The runtime executes an operator only if its input is ready, and the filter operators only forward non-empty processed data to subsequent operators. For example, if the retrieve pose operator (Fig. 5.8) cannot find any extracted poses in its input, the flow stops propagating. This simple design makes it easy for Peekaboo APIs to support IFTTT-like trigger-action programs natively.

This example also shows how the output of a single operator can be forwarded to multiple operators, similar in spirit to Unix's *tee* command. The blocking and non-blocking *join* operators allow Peekaboo to accommodate distributed asynchronous sensors, supporting smart home apps with complex logic across distributed devices within the same home.

5.5 Implementing the Hub Runtime

We implemented Peekaboo by leveraging Node-Red [170], a visual programming platform developed by IBM to wire together devices using a library of customized blocks. The notion of "blocks" and directional "connections" in the visual programming language are well suited to Peekaboo's chainable operators. We implemented Peekaboo operators as a set of Node-Red compatible building blocks, so developers can use a Node-Red web-based flow editor UI as the programming environment and leverage its built-in debugging utilities (e.g., displaying images, playing an audio, printing output data). The source code is available at

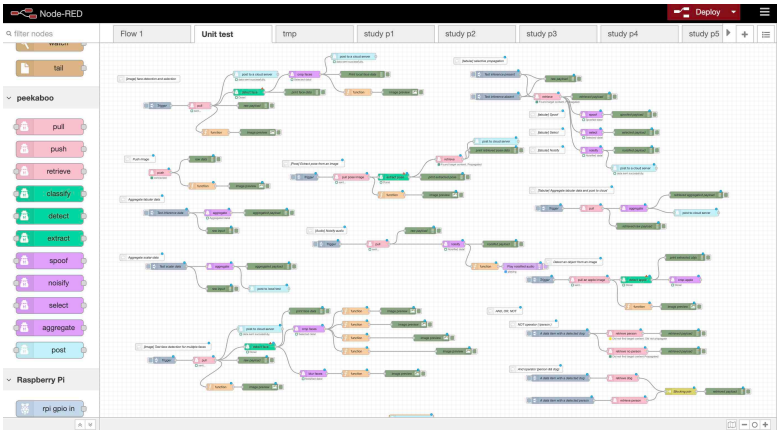


Figure 5.9: The editor interface and the unit test tab presented to the developers. Developers can inspect the unit tests to understand operators' behavior and the connection grammar.

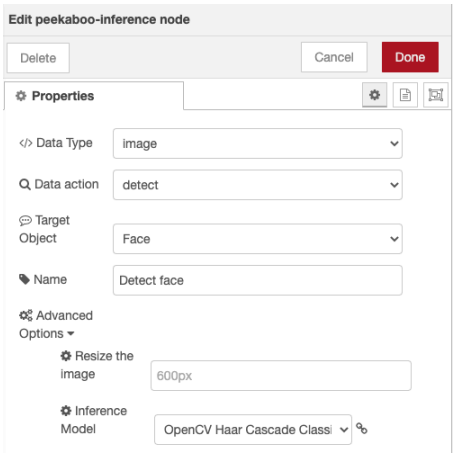


Figure 5.10: The configuration interface of an inference operator.

`https://github.com/CMUChimpsLab/Peekaboo`.

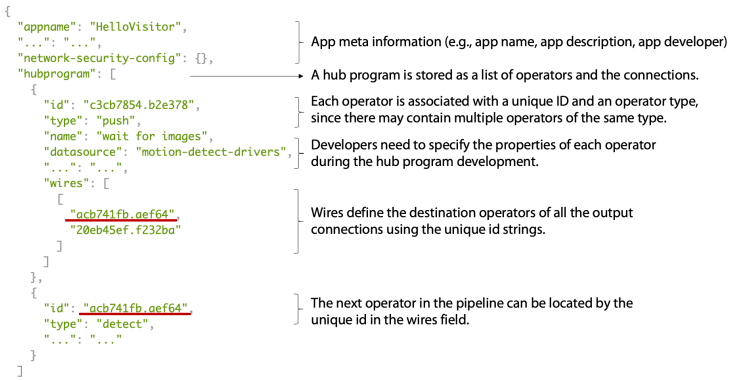


Figure 5.11: The manifest file of a hub program is stored in JSON format. The manifest file contains three types of information: app meta information, security configuration, and the hub program presentation.

Operators: Each Peekaboo operator is written in JavaScript and executed by a Node-red runtime once deployed. One challenge in Peekaboo is the relative lack of support for machine learning algorithms in JavaScript. Our experiments with multiple JS implementations (e.g., opencv4nodejs) showed them to be slow and inaccurate. Instead, we design the Peekaboo runtime using a microservice architecture, where several ML inference algorithms are hosted in containerized services running on the Peekaboo hub, and only Peekaboo inference operators can communicate with these services locally through web sockets.

Drivers to obtain sensor data: Each Peekaboo hub program gets data from hub runtime drivers rather than querying hardware directly. Developers may implement customized runtime drivers in arbitrary code, similar to HomeOS [101]. These drivers retrieve raw sensor data (video, image, audio, tabular, or scalar)

from edge devices, format it (e.g., b64string for images, bytes for video/audio, JSON for tabular data), and publish the data in a local socket. The hub program can either pull the latest data (e.g., retrieving a real-time photo) from the driver or register a push subscription on the driver (e.g., receiving a photo if there is a significant scene change). Note that these runtime drivers handle device heterogeneity and specifics of getting data from them, while the Peekaboo operators are device-agnostic data stream operators.

Runtime: We deployed the Peekaboo runtime (i.e., Node-Red, drivers, containerized inference services) on a Raspberry Pi 4B (4GB), along with a Google Coral TPU and a Zigbee adaptor, with a total cost of $\approx$ \$100 USD. Most of our deep-learning-based inference tasks use pre-trained MobileNet [326] models and are outsourced to the TPU.

End-to-end applications: We developed drivers for four Peekaboo IoT devices, a customized smart camera and smart speaker using Google AIY Kits [135], a simulated RESTFul API that generates tabular smart TV logs, and an Aqara Zigbee humidity sensor, covering the five data types in Table 3.2 (video, image, audio, tabular, and scalar).

We built five end-to-end applications using these devices: video based heart rate measurement [27], HelloVisitor (Fig. 5.1), incognito Speech Assistant (Fig. 5.7), Smart TV logs collector (Fig. 5.6), and humidity-based irrigation reminder. We built customized cloud services and web UIs for all five apps.

Table 5.2: We implemented 68 unique manifests for over 200 smart home use cases, analyzed the types of pre-processing in these manifests, and if the data needs for that use case could not be supported, we examined why.

Pre-processing	#Scenarios supported	Why Peekaboo could not support some of the scenarios
Content selection	64 / 68	e.g., hard to select content of interest, need to be used for online albums
Conditional filtering	57 / 68	e.g., high-stake tasks, proprietary implementations, insufficient computing resources
Explicit noise injection	51 / 68	e.g., high-stake tasks, injected noise may break the intended tasks
Always need raw data	3 / 68	the intersection of the three categories listed above

5.6 Evaluation

This section presents detailed experimental evaluations of the feasibility and benefits of Peekaboo. We first created 68 different manifests for all the 200+ smart home use cases (Section **??**). Note, these manifests

only work with bare-bone server implementations as we do not have proprietary backend implementations from manufacturers. We then analyzed types of pre-processing opportunities that can be enabled by these manifests (§5.6.1). We also investigated the feasibility of balancing the tradeoff between privacy protection and utility for each type of pre-processing (§5.6.2). We later built five end-to-end real-world Peekaboo apps, covering five data types and used these apps to evaluate our system performance (i.e., latency and throughput) (§5.6.3). Finally, we demonstrate several hub privacy features that both developers and users essentially get for free, which work across apps and devices (§5.6.5).

5.6.1 Evaluating Peekaboo's coverage of smart home scenarios

To validate the feasibility of Peekaboo, we authored manifests to cover the use cases described in §**??**.

Method. A smart home use case can be realized through different types of sensors. For example, a dedicated occupancy sensor or a camera can enable an occupancy-based application, although required data collection behaviors can differ. Two authors collaboratively implemented one manifest for each usage-sensor pair to explore different pre-processing opportunities. We found that many manifests could be reused for multiple scenarios, and so this process resulted in 68 manifests.

Results. Of the 68 manifests, we only identified three that always need raw data: a smart cooking device that uses a camera to analyze ingredients, a smart toilet that analyzes poop for diseases, and a microphone that performs spirometry [207]. For these kinds of apps, developers can directly connect a `provider` operator to a `network` operator to send out the raw data. While Peekaboo does not reduce data egress in these cases, it still provides transparency of data collection (e.g., what data has been sent to the service provider, and how often) and a binary on-off control.

For the other 65 manifests, Peekaboo APIs can enable at least one of the following types of pre-processing: content selection (e.g., cropping faces), conditional filtering (e.g., only sending data if a person appears in the view), and explicit noise injection (e.g., changing the pitch of an audio recording). Table 5.2 enumerates the breakdown of supported pre-processing across all scenarios, as well as the unsupported reasons.

Content selection is the most common pre-processing (64 of 68 scenarios). Examples include cropping faces from images, extracting audio frequency spectrum, and aggregating numerical values. We could not apply content selection to 4 apps for two reasons. First, the algorithms to select the content of interest cannot run on a Peekaboo hub. For example, the algorithm to recognize food ingredient is proprietary and may require significant computational resources and frequent model updates. Second, developers need raw data to fulfill the data collection purpose. For example, automatic photography (e.g., Google Clips [5]) needs to collect and store the original photo.

Conditional filtering is also common (57/68). Since many smart home apps are event-driven, adding

a local event filter can significantly reduce data egress. For example, a manifest that filters images based on the presence of faces can reduce the number of outgoing images from a basic motion-activated camera. However, one constraint of Peekaboo's architecture is that the open-source hub algorithms may not be as accurate as their cloud counterparts. As a result, we cannot insert filters for high-stake tasks, e.g., elderly fall detection. Another constraint is that some conditional filters might be proprietary with no current open source equivalent (e.g., water leakage detection). Finally, some conditional filters may require extensive computational power and storage, which cannot run on a local hub (e.g., wanted criminal search using smart doorbells).

While it is possible to inject explicit noise into the data pre-processing pipelines, there is one crucial privacy-utility trade-off: the injected noise may break the intended functionality and reduce the service quality. So we cannot explicitly inject noise to apps for high-stake tasks. Besides, many scenarios collect only coarse data (e.g., binary occupancy) or need raw data, where explicit noise injection is not applicable.

5.6.2 Privacy-utility trade-offs

While some data transformations are unlikely to affect service quality (e.g., many kinds of tabular data aggregation), others might have negative impacts. A main concern for content selection and conditional filtering is that the open-source inference models on the hub may be less effective than large proprietary machine learning models on the cloud. Furthermore, for explicit noise injection, that noise might break intended functionalities. While the actual trade-offs depend heavily on the applications, we used three example apps to demonstrate the **feasibility** of using simple algorithms to reduce privacy risks while having little impact on utility.

Method. We chose three pre-processing tasks to evaluate: a face-only video doorbell (HelloVisitor, Fig. 5.1, content-selection), a person-activated camera (Fig. 5.8, conditional filtering), and an incognito voice assistant (Fig. 5.7, noise injection). We chose these tasks because they represent different types of pre-processing, the availability of labeled ground truth data, and the availability of publicly accessible cloud-based baselines.

The face-only video doorbell can improve privacy by only sending images of faces, although Peekaboo's operator might miss some faces that a more capable cloud algorithm could detect. Similarly, the person-activated camera reduces unnecessary outgoing images, but it may miss images that potentially contain a person. For these two apps, we quantified the privacy benefits using the percentage of data egress reduction in bytes and the impact on utility using F1 scores. Finally, our incognito voice assistant protects speakers' identity by changing the tempo and pitch of the captured audio with small random variations using the *noisify* operator (<10%). We quantify the privacy benefit by measuring speaker recognition error and the utility impact using speech recognition error. We measured accuracy using the Levenshtein function [381] to compare the recognized speech text with the ground truth text.

We ran the experiments with multiple benchmark data sets. First, we used the ChokePoint dataset [389],

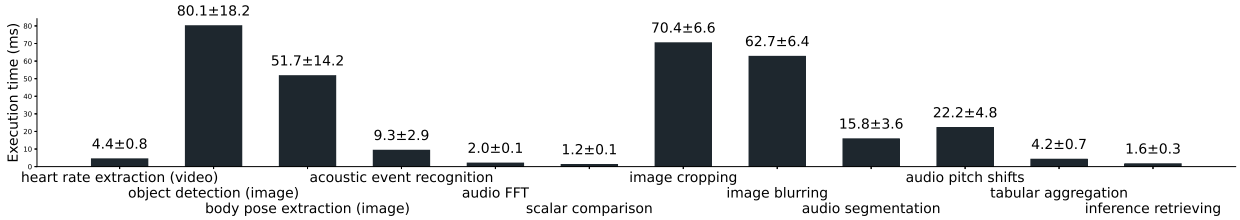


Figure 5.12: The execution times (in milliseconds) for transforming 1-second videos/audios (normalized), 800x600 images, and a 5-column table with 100 entries. Our low-cost setup can support 25 inference tasks and more than 100 filtering transformations per second, which we believe is sufficient to support many smart home scenarios.

a person identification dataset under real-world surveillance conditions, to test the face-only video doorbell. Next, we selected 457 "home office" videos from an in-home activity dataset [335] to evaluate the person-activated cameras. Finally, we used speech recordings from the CMU PDA database [276] to test the incognito voice assistant (6 unique speakers, 112 unique audio files). For each speaker, we used half of the audio files (7-15 files) for speaker enrollment and the other half for speaker recognition and speech recognition. We used state-of-the-art cloud-based solutions from Microsoft Cognitive Services (i.e., face detection, person detection, speaker recognition, speech recognition) as the baseline [25].

Table 5.3: Simple pre-processing algorithms provide significant improvement in the amount of potentially sensitive data sent, while maintaining utility.

	Privacy Metric	Utility Metric
	Outgoing data	F1 Score
Peekaboo face doorbell	**6.0**%	94.3%
Baseline doorbell	100%	**95.3**%
Peekaboo person camera	**4.3**%	93.2%
Baseline camera	100%	**96.2**%
	Speaker Recog Accuracy	Speech Word Error Rate
Peekaboo voice assistant	**27.7**%	11.88%
Baseline voice assistant	100%	**9.27**%

Results. Table 5.3 presents the privacy-utility tradeoffs across our three example apps. The face-only camera reduces data egress from 1 million full-resolution images (12 GB) to 64,000 face images (200 MB), while the F-1 score of the local model is only 1% lower than the Microsoft Azure API. The person-activated app replaces 2868 images (62.8%) with less privacy-sensitive pose key points and removes unnecessary background pixels from 770 images (16.8%), resulting in a data reduction of 95.7% (from 164 MB to 7

MB). Meanwhile, the F-1 score of the local model is only slightly lower than the Microsoft Azure API (93.2% v.s. 96.2%). The incognito voice assistant reduces speaker identification accuracy from 100% to 27.68% (lower is better since it provides more anonymity) while only reducing speech recognition accuracy by 2.61%. Our results show that Peekaboo's pre-processing (e.g., small random pitch shifts, pre-filtering) can significantly reduce data egress with minor adverse impacts on the intended tasks.

5.6.3 System Performance

Peekaboo's architecture has two major constraints. The first is that the hub has more limited computing resources than cloud servers. Resource constraints on the hub may lead to pre-processing of data taking longer, or limit the ability to scale to many simultaneous pre-processing tasks in a smart home. Second, the pipe-and-filter architectural style introduces latency due to repetitive parsing and unparsing across filters. We conducted experiments to quantify the *computation load (i.e., throughput)* of data transformations and the *end-to-end latency* of different pipelines. We note, however, that our prototype is not highly optimized and is running on relatively low-end hardware, so this evaluation is intended to provide a rough lower bound on performance.

Method. We deployed the hub runtime to a low-cost setup and used the four sensors as the edge devices, both described in §6.5. We also set up a cloud server on an AWS p2.xlarge instance, in an AWS region close to the authors institution.

We first characterized the computation load of common data transformations identified in §5.6.1, including both inference and filtering. For inference, we profiled object detection (e.g., face, person, floor), audio event recognition (e.g., baby crying, water dripping), audio frequency spectrum extraction, body pose extraction, video heart rate extraction and scalar value comparison. For filtering, we profiled object-based image cropping, object-based image blurring, audio segmentation, audio pitch random shifts, tabular data aggregation, and inference results retrieving.

We used 3 videos from Intel's IoT Devkit [172] (~2 min) to test video transformations, 3 sound clips from Google AudioSet [132] (~10 seconds) to test audio transformations, 10 images from the ChokePoint dataset [390] (800x600) to test image transformations, and a synthetic dataset containing 100 entries to test tabular data transformations. We then measured the average computation time of 1000 repetitions.

Next, we compared the difference in latency when running pre-processing tasks on the hub vs running them on the cloud. With respect to the former, we measured the pre-processing latency on the hub and the transmission time of sending the processed data to the cloud. With respect to the latter, we measured the transmission time of sending the raw data to the cloud and the pre-processing latency on the cloud. We used the dataset mentioned above to test 3 end-to-end apps described in §5.4. According to prior work [16], the frequency of network events from typical sensors (e.g., sleep monitors, nest cameras, switches) varies from a few per minute to a few per hour. We stress tested the requests at an interval of 0.5 seconds and measured

the average latency of 1000 repetitions.

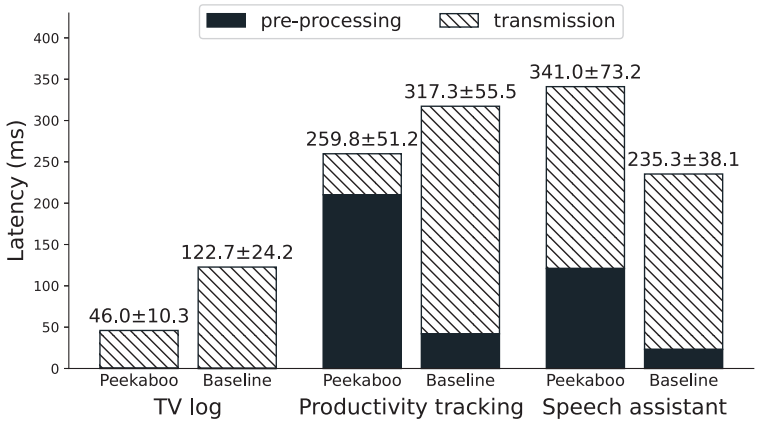


Figure 5.13: The latencies for the three apps are comparable to the conventional cloud-based approaches. The pre-processing times for TV logs are negligible. Although Peekaboo apps spend more time pre-processing data on the hub than on the cloud, they may spend less time on data transmission since pre-processing can reduce outgoing data size.

Results: Figure 5.12 presents the individual completion times on a Raspberry Pi 4B for common data transformations. Most filtering tasks take 5 ms to 80 ms to complete, while the inference tasks on multimedia data are generally more expensive. Although inference tasks (e.g., object detection) take around 80 ms to complete, they consume little CPU resources since the core computation are outsourced to the TPU device, and the operator runs asynchronously. On average, most of the ML models we use (MobileNet [326]) take around 40 ms on the TPU per inference. So we estimate that our low-cost setup can support 25 inference tasks and more than 100 filtering transformations per second, which we believe is sufficient to simultaneously support many smart home scenarios.

Figure 5.13 presents the average latency for different apps, showing that Peekaboo can achieve a latency comparable to conventional cloud-based approaches. The TV log app's hub program spends negligible time on log aggregation but reduces the outgoing data size significantly, thus experiencing lower latency with Peekaboo. Similarly, Peekaboo's productivity tracking app spends 210.2 ms (std=49.6 ms) to extract the pose, reducing the network transmission time from 275.3 ms (std=50ms) to 40.1 ms (std=11.1ms). The Peekaboo speech assistant app takes 106ms more to pre-process and send a 10-second sound clip to the cloud than the conventional approach since the pre-processing in this case does not reduce the outgoing data size. In summary, the latency for pre-processing outgoing data on the hub is comparable to the conventional cloud-based approach. Some Peekaboo apps have reduced data transmission time due to the reduced size of outgoing data.

5.6.4 Developer User Studies

Method: To evaluate the complexity of creating hub programs using operators, we conducted an IRB-approved study with six developers (3 Male, 3 Female, ages 21-26). Participants were asked to author four manifests using an IDE accessible from their web browsers. We configured the inference services and drivers on our hub as well as all cloud services in advance of the study, because we wanted to focus on the usability and understandability of Peekaboo's programming model.

All participants had at least three years of programming experience, 3 have developed mobile apps, and 2 have developed apps for IoT scenarios specifically. None had used the Node-Red IDE before. Each study took around 60 minutes, and we paid each participant $15 upon completing the study. The study was performed remotely over Zoom video conferencing.

During the study, we provided our participants with limited overall assistance – a brief introduction to Peekaboo and the Node-Red interface, some basic documentation on operators (i.e., the taxonomy in Fig. 5.3), many unit tests for each operator with different properties (Fig. 5.9), and the *HelloVisitor* program as a complete example serving as a warm-up task. Participants can inspect and run the unit tests to gain a concrete understanding of the input and output of each operator and the connection grammar.

We presented each participant with four tasks: the first three were the three case studies (§5.4), and the fourth was an additional open-ended task of their choice. We described the context and the desired data granularity (i.e., data content and conditions) for each task and asked them to implement the corresponding hub programs. We did not randomize the order, since we found that ordering the tasks based on difficulty can scaffold participants' learning process in our pilot studies. Finally, we marked the task completed if the hub program could send the right data to a pre-configured server.

Results: Table 5.4 shows the quantitative results of our lab study. All participants completed the tasks successfully. The task completion time across participants are similar: 3-5 minutes for task 1, 4-6 minutes for task 2, and 7-21 minutes for task 3, reflecting the difficulties of these tasks. Most participants picked up Peekaboo's programming model quickly without much explanation. For example, participants #4, and #5 completed Task 1 without using the unit tests.

We also count the total number of operators, and the number of *debug* operators, in the final hub programs. In the first three tasks, most participants actively used the debug operator to help them understand each operator's behavior. When they reached the fourth task, they were quite confident about using these operators in their planned programs. If the target program is relatively simple, participants (P2-6) program it without the help of *debug* operator in a short time (2-3 minutes), suggesting the Peekaboo API is easy to learn and use.

Our results also demonstrate our API's flexibility, which allows developers to easily explore the impact of various design choices on performance and privacy. Multiple participants demonstrated alternate imple-

Participant	Task 1		Task 2		Task 3		Task 4	
	Time	#	Time	#	Time	#	Time	#
#1	5	6 (1)	6	9 (1)	10	17 (4)	7	15 (5)
#2	4	6 (2)	4	10 (2)	12	16 (5)	2	4 (0)
#3	5	5 (1)	4	6 (0)	9	16 (4)	3	8 (0)
#4	2	6 (2)	4	12 (3)	21	9 (4)	2	5 (0)
#5	3	6 (2)	5	9 (1)	12	15 (5)	2	4 (0)
#6	3	4 (1)	4	7 (1)	7	12 (2)	2	5 (0)

Table 5.4: All participants learned the APIs quickly and completed the tasks successfully. The "Time" column shows minutes spent on each task. The "#" column contains total numbers of operators plus *debug* operators (in parens) in the final hub program. After the first three manifests, most participants implemented manifests without using debug operators.

mentations from the ones we show in §5.4. For example, P4 implemented the productivity tracking hub program in a reversed manner: cropping the body image first and then extracting the pose from this cropped image. This alternative design can potentially be more efficient if a space is mostly unoccupied.

While Peekaboo is easy to learn, our user study identified two barriers. Multiple participants felt confused about a few common features, and we pointed them to the corresponding unit tests. First, participants confused *retrieve* with *select*, and expected it to keep the original data in its output. Second, the data communicated between operators is an array of Peekaboo items rather than an individual item, and this led to some confusion when participants intended to process the faces of multiple persons in one image (see Fig. 5.2). After the study, we added additional unit tests and documentation to address these misunderstandings.

Finally, participants offered their opinions on Peekaboo APIs unprompted, such as "much easier to specify data collection behavior through this interface than writing complete code," "Android should also have something similar".

5.6.5 Hub Privacy Features

An advantage of Peekaboo's chained operators is that their structure and well-known semantics make it fairly easy to analyze and modify app behavior. We developed four built-in features to demonstrate the mechanisms.

Table 5.5: Auto-generated explanations for 3 case studies using a simple template. We highlight the properties of the template using underlines: trigger, content data, destination and conditions.

App #	Generated privacy explanations
# 1	For every week, the app sends duration data aggregated by content category to www.abc.com.
# 2	When the microphone detects a trigger phrase, the app sends anonymized speech audios to www.abc.com.
# 3	For every 30 minutes, the app sends extracted poses to www.abc.com. For every 30 minutes, the app sends cropped person images to www.abc.com if the app cannot recognize poses from the raw image.

We built a simple static analyzer to **generate natural language privacy descriptions** based on the manifest automatically, which can support users' decision-making in installing a new Peekaboo app. For any manifest, we can describe its data collection behavior using a four-element template: *[trigger], the app sends [content data] to [destination], if [condition].* We derived a set of heuristics to annotate the above properties for each edge in the pipeline based on the operator behavior. For example, the analyzer annotates [*content data*] as *face images* after processing by a cropping face operator. The analyzer annotates the [*condition*] if there is a join operator in the manifest and [*trigger*] based on the *inject* or *push* operator. The analyzer stops the annotation when it traverses the whole graph and uses the derived properties to explain the data collection behavior of each network operator. Table 5.5 enumerates explanations generated for apps in §5.4.

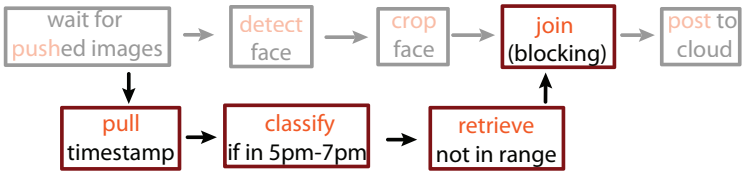


Figure 5.14: By inserting the subgraph of time checking (highlighted in red boxes), the modified HelloVisitor manifest will not send data outside between 5 pm and 7 pm.

We also implemented a **time-based scheduling** feature, which can pragmatically modify a Peekaboo manifest by inserting a subgraph of operators (highlighted in red boxes in Fig. 5.14), so the app cannot send data outside at certain times. Imagine that a family does not want their faces captured by the video doorbell when they arrive home, and most family members usually arrive between 5 pm and 7 pm. Based on users' time specifications, this feature can automatically insert a time-checking branch (i.e., pull->classify->retrieve) to check the time, and merge the two branches using a blocking join operator. In doing so, the data flow can only reach the network operator if the time condition qualifies.

Another feature we built is **pull rate-limiting**, which allows users to control the frequency at which a smart home app pulls data by modifying the operator properties. Figure 5.4 illustrates a water leak detection app, which pulls an image from an existing smart camera every 30 minutes. Using *pull rate limiting*, the runtime can modify the "interval" property of the *inject* operator from *30 minutes* to *120 minutes*, making the app only pull images every two hours.

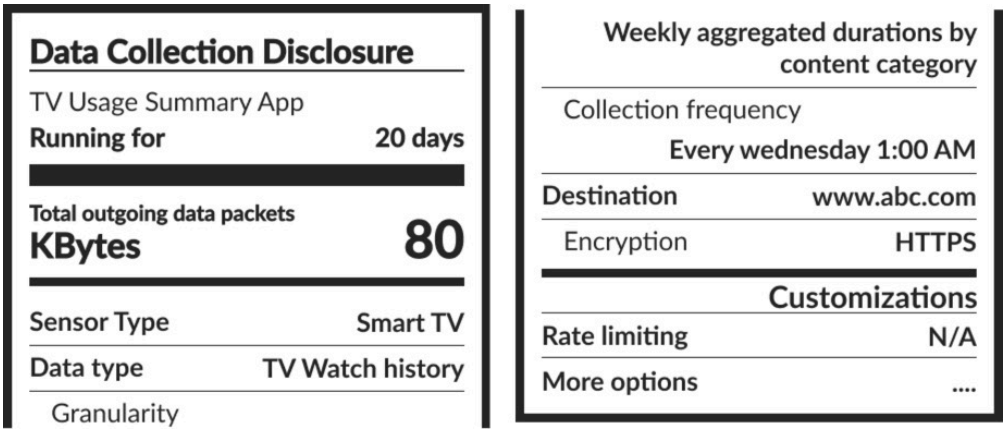


Figure 5.15: A "live" privacy nutrition label generated by the Peekaboo runtime automatically.

Lastly, as a proof of concept, we combined the 3 above features to generate live "privacy nutrition labels" that summarizes an app's behaviors (Fig. 5.15). Apple now requires iOS app developers to fill in a form to create a self-reported "nutrition label" for privacy disclosures. However, it can be hard for developers to accurately fill out these forms. Furthermore, Apple and third parties cannot easily verify these declared behaviors. In contrast, Peekaboo's labels can be auto-generated, thus requiring less effort from developers, and always reflect the actual data practice.

5.7 Related Work

Manifest & operators: There are many examples of using manifests to create a sandboxed environment to contain untrusted applications (e.g., Janus [134], Android permissions [13], MUDs [63]). In contrast to most existing manifests, which often confine applications' access to system resources (e.g., network, storage,

sensors), Peekaboo's manifest confines access to data content (e.g., face images, speech audio) in a fine-grained manner. Due to the vast number of data granularities, the traditional manifest representation (i.e., raw access enumeration) is no longer feasible. Peekaboo addresses this challenge by allowing developers to assemble desired APIs by wiring together a fixed set of operators.

The design of Peekaboo operators is inspired by Unix pipes and PrivacyStreams [221]. PrivacyStreams splits single-pipeline Android data processing into a number of reusable SQL-like operators (e.g., sortby, filter, groupby), which can make data processing more transparent. The key innovations of Peekaboo are the integrated designs (a manifest, a fixed set of operators, and a trusted runtime with pre-loaded implementations) and the demonstration of using a small and fixed set of operators to support a large number of data pre-processing scenarios. These two ideas allow Peekaboo to offer many features that PrivacyStreams cannot offer, such as OS-level enforcement (i.e., developers can only collect data they claimed in the manifest), additional built-in privacy features, and explicit declarations of fine-grained data granularity (e.g., conditions).

Privacy-sensitive software architectures: A number of privacy-sensitive architectures for IoT have been proposed as alternatives to conventional cloud-based designs. One example is to process all sensor data at the "edge", thereby avoiding sending data to the cloud [398]. Another approach is to use trusted cloud computingand perform data minimization through technologies like DIY hosting [284], privacy-sensitive machine learning [58, 133, 212, 239], and federated learning techniques [198]. However, these approaches often come with tradeoffs such as sacrificing computational efficiency [212, 214], development flexibility [133, 214], or service quality [362].

In contrast, Peekaboo offers a hybrid approach that does some pre-processing locally on the hub while also allowing developers to use cloud services in a manner that they are accustomed to [143]. Peekaboo's hub is similar in spirit to cloudlets [328], but the key difference is that the computation running on the hub is structured and enforced using the operator-based manifest. Peekaboo is also inspired by past smart home hub/gateway/firewall projects [57, 86, 196, 238, 402]. Peekaboo has two major differences. First, Peekaboo enables data minimization at the data content level (e.g., only sending face images), while existing projects can only block individual outgoing network requests. Second, Peekaboo requires developers to explicitly declare the data collection behaviors, facilitating auditing and enforcement.

Privacy awareness and control: A complementary approach to privacy is better mechanisms for notice and choice [4, 101, 166, 192, 231], e.g. at runtime for mobile apps [95, 263], often extended to smart home contexts [101]. In the case of IoT privacy, merely allowing or denying sensor access is insufficient and this all-or-nothing access control either exposes sensitive data or breaks the app functionality. In response to this, recent efforts offer finer-grained control and transparency, particularly on the fidelity of the data [3, 178, 303, 369].

However, privacy support in these systems is often built on an individual basis with no common struc-

ture, imposing challenges for both app development and user privacy management. Third-party auditors also cannot easily verify if these features work as claimed [255, 285]. Peekaboo addresses these needs through a novel privacy architecture, which can enable transparent and enforceable data collection, and offer centrally manageable hub privacy features.

5.8 Discussion

Architecture adoption: Many requirements for the Peekaboo hub are well aligned with the roles of recent commercial "hub" products, which may facilitate adoption. For example, many hubs (e.g., Philips Hue Smart Hub) serve as a central in-home gateway to connect devices, mediating access between the internal devices and the Internet. In addition, most hubs (e.g., Nest Hub Pro) have a moderate amount of computing power to pre-process out-going data and offer a centralized hub user interface. A few hubs (e.g., Samsung SmartThings Hub) even behave like an early app store.

Alternative implementations: Peekaboo's manifest is a new program representation that offers more flexibility than all-or-nothing access, while being more structural and verifiable than arbitrary code (e.g., Java). Future work can also implement the manifest in other flow-based programming frameworks (e.g., NoFlo, Pyperator) [206]. We chose Node-Red since it is popular in the home automation hobbyist community, provides many open-source device drivers, and has a mature user interface.

Design pattern adoption: The design of Peekaboo can be generalized as a reusable design pattern for cases where first- and third-parties are trying to access sensitive data, e.g., browser plugins, calendar APIs, and smartphones [300]. Our core ideas of a fixed set of operators, a text-based manifest where all outgoing data flows must be declared, and a trusted computing platform with pre-loaded implementations can thus be useful in these cases. For example, Google Calendar only allows users to grant all-or-nothing access to third-party developers. However, most apps (e.g., Zoom) do not need full access. A future calendar API might offer a set of common operators instead and allow developers to program their access in Peekaboo-like manifests. This design pattern can make data flows transparent, enforce data transformations, and allow third-parties to build independent privacy features.

Role of users, developers, auditors in determining privacy-utility trade-offs: Peekaboo has the potential to disincentivize overcollection of data. We expect Peekaboo manifests to be public, making it possible for app stores and third-party auditors (e.g., Consumer Reports) to analyze manifests programmatically at scale. Users can also see if the required data granularity make sense, and flag items in a review if they do not, block certain outgoing data, or choose not to install an app. Altogether, this kind of transparency has the potential for nudging developers to collect less data.

Hosting proprietary algorithms: Some of the best implementations of inference mechanisms such as

keyword spotting, keypoint tracking, and biometric authentication can be proprietary. Platform/hub builders may implement these algorithms inside their hub drivers in the future, or hardware developers can provide the functionality directly on the edge devices.

Extensibility: Peekaboo assumes a fixed set of operators and a stable data model to support its data pre-processing pipelines. Although the taxonomy in Fig. 5.3 may not be complete, we anticipate that the list and semantics of operators will converge quickly and remain stable for years. Further, platform builders can extend the operator options by expanding supported data transformations (e.g., removing all audible frequencies from the audio [174]) and adding new pre-trained models. Platform builders can also develop new drivers to support more devices.

We expect the runtime to be updated over time, analogous to getting a new version of Linux or Java. Also, similar to Android's manifest, a Peekaboo manifest will need to specify a minimum required runtime version. We do not expect the pre-loaded operators to grow into a large library of data transformation functions. Instead, we believe a few simple and common data transformations (e.g., tabular data aggregation, image cropping/blurring) can cover many common scenarios and go a long way towards improving privacy.

Benefits of Peekaboo: Peekaboo has three important advantages over building data minimization features individually. (1) **Transparency**. Individually built data minimization practices are black boxes to outsiders. Indeed, even if developers open-source their products or allow a third-party auditor to access their code-bases, inspecting the actual data collection behavior is difficult. (2) **Ease of development**. Building data minimization algorithms and user interfaces for privacy requires significant effort. By authoring a Peekaboo manifest, Peekaboo developers can leverage many built-in features for free. (3) **Centralized privacy management**. If all developers build management interfaces individually, users would have to deal with potential inconsistencies between these user interfaces and their different semantics. In contrast, Peekaboo can offer centralized, fine-grained features across devices.

5.9 Future work & Limitations

End-to-end encryption: Peekaboo currently requires two separate encrypted connections (i.e., devices-to-hub and hub-to-servers) for its operation rather than end-to-end encryption from device to server directly. While SSL proxy mechanisms (e..g, [126, 190, 282, 296]) may provide a way to support end-to-end encrypted connections with the ability to verify what data is being sent, it is not clear whether they can support Peekaboo's operators that transform data. This is a current limitation, and we defer this to future work.

More hub features: Beyond the four hub privacy features introduced in §5.6.5, further work may explore many other hub privacy features by analyzing and rewriting the manifest program. For example, the hub may aggregate the installed manifests, make privacy nutrition labels interactive, enable centralized privacy

dashboards, and allow users to query what/when/how data flows out. The hub can also potentially allow users to apply global filters (e.g., blocking outgoing face images) across manifests, e.g. to make guests more comfortable with cameras around the house.

Manifests for other communication patterns: Peekaboo focuses on whitelisting edge-to-cloud data flows to improve privacy. A promising research direction is to generalize the manifest for other communication patterns, such as device-to-device, cloud-to-device, or even physical actuation. Ideally, we may have a set of Peekaboo-like manifests for each IoT device, whitelisting its interactions with the rest of the world in a common, structured, useful, and understandable way. Such an infrastructure can help users establish a correct mental model of device behaviors, allow the hub to offer built-in protection for the physical events (e.g., a bread toaster cannot run for more than an hour), and ease software development.

Unpredictable hub program rewriting. In this chapter, we demonstrate the feasibility of several features that rewrite the hub program written by developers. However, such program rewriting may break the original functionality if users misuse the feature. For example, a user may choose to blur their faces in an unlocking app, which makes her unrecognizable to the unlocking app.

5.10 Conclusion

This paper introduces Peekaboo, a new IoT app development framework to help developers build privacy-sensitive smart home apps. Peekaboo offers a hybrid architecture, where a local **user-controlled hub** pre-processes smart home data in a structured manner before relaying it to external cloud servers. In designing Peekaboo, we propose three key ideas: (1) factoring out repetitive data pre-processing tasks from the cloud side onto a user-controlled hub; (2) supporting the implementation of these tasks through a fixed set of open-source, reusable, and chainable **operators**; (3) describing the data pre-processing pipelines in a text-based **manifest** file, which only stores the operators' specifications, but not the operators' actual implementation.

Chapter 6

MapAggregate: Modular Privacy Flows through Serverless Computing

Would you be willing to share your video doorbell's feed to help find your neighbor's lost cat? Participatory sensing applications such as this one rely on private data from several smart homes to provide useful services, yet pose many privacy risks to home owners and bystanders. We present MapAggregate, a privacy-sensitive software architecture that only allows developers to query insights, on data aggregated across multiple homes, in a manner that is controlled and configured by homeowners in the interest of privacy. MapAggregate requires developers to declare a formal plan of their intended data aggregation behavior as an application *manifest* explicitly and up-front. Data aggregation is specified through a stream-oriented pipeline similar to Unix pipes, each composed of a small number of stateless operators (e.g., summing individual numbers, adding noise). MapAggregate then enforces this manifest in a privacy-preserving manner through the use of ephemeral and isolated serverless functions, that limit the ability of application developers to access individual homeowner data. We implement a prototype of MapAggregate, discuss how MapAggregate mitigates different privacy attacks, and present a detailed evaluation of MapAggregate's cost and latency.

6.1 Introduction

Smart devices such as smart cameras and Internet-connected speakers have been rapidly proliferating in consumers' homes. While most of the use cases around these devices have been geared directly toward personal uses, emerging use cases suggest that these devices also have potential value for many smart city applications [151, 361]. This paper envisions a future where developers can build participatory sensing applications (apps) [44, 145], and users can install these apps to monetize data collected by their smart home devices [184, 361].

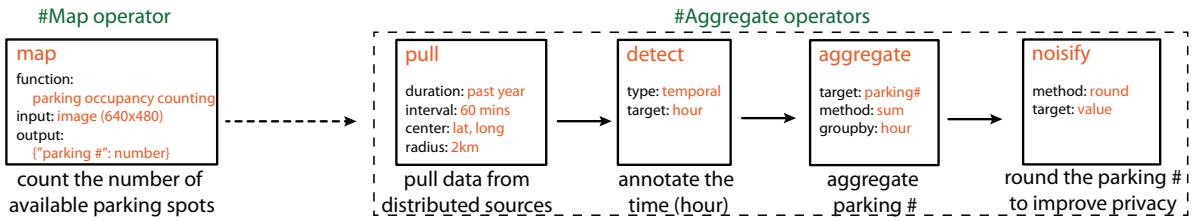


Figure 6.1: Developers declare intended data aggregation behaviors in a text-based application manifest. To specify these behaviors, developers choose from a small set of operators with well-defined semantics, authoring a stream-oriented pipeline similar to Unix pipes. A MapAggregate manifest contains two components: a `Map` program (left) that processes data from an individual source to compute a set of intermediate key-value pairs, and an `Aggregate` program (right) that combines intermediate results from geospatially distributed sources to generate aggregated insights. Both programs work collaboratively to minimize data egress. The `Map` denatures the raw data to avoid oversensing [36], and the `Aggregate` combines the data in ways that are still useful for many smart city apps while also making it hard to re-identify individual denatured data.

For instance, many city councils want to know the hourly street parking usage in the past year to decide whether to build new parking structures [253, 343, 350]. A developer could build an application to query street-facing video doorbells, analyze captured images, and return an hourly parking occupancy map for a designated region [324]. However, these images can easily be misused for malicious purposes, like using parking occupancy to infer if no one is at home. A key challenge to enable multi-home applications is the need to preserve data privacy of individual homeowners. Should apps have direct access to raw smart home data at any stage (whether at the edge or the cloud), malicious developers may exfiltrate private data unrelated to the intended application. Indeed, the classic data broker model [77, 280], where a broker collects users' data at a centralized server, is ripe for such misuse. Even if such a platform promises to obscure privacy-sensitive content prior to further processing (e.g., faces [3, 369], marked regions [303]), it is challenging for homeowners to have any assurance of privacy after their data has been handed over to a developer. Further, consolidation of smart home data in the hands of developers has already received privacy push-back. For example, users have already expressed concerns about the surveillance enabled by Amazon's Ring video doorbells [41].

This paper presents MapAggregate[1], a novel software architecture that only allows developers to query insights, on data aggregated across multiple users[2], in a manner that is controlled and configured by these users in the interest of privacy. Aggregating data in the interest of privacy, like reporting the sum of parking

[1]The name "MapAggregate" is inspired by MapReduce [82]. Instead of reducing data from a centralized data repository, MapAggregate aggregates data streams from spatially distributed data sources on the fly.

[2]We refer to smart home owners as users in this chapter.

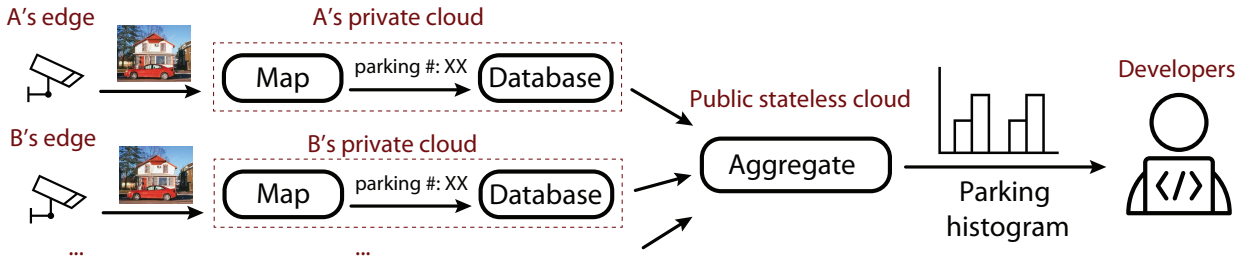


Figure 6.2: Given the manifest (Fig. 6.1), MapAggregate first spawns a public stateless `Aggregate` function by assembling a executable using open source Aggregate operator implementations (i.e., pull, detect, aggregate, noisify). Later, when a user chooses to install the manifest, MapAggregate spawns a stateless `Map` function in an independent private cloud that the user controls. The MapAggregate runtime then infers the application logic and storage requirements automatically: the `Map` function pulls raw images from video doorbells every 60 minutes, computes the occupancy information, saves the key-value output to users' personal cloud databases, and deletes entries older than a year. Upon an aggregation request, the `Aggregate` function pulls key-value pairs from qualified users' private cloud, combines the data in isolated stateless functions, and returns the aggregated results to developers.

spots rather than per-home parking, is a well-known concept. However, MapAggregate is unique in that users can specify enforceable aggregation requirements on how developers can aggregate their data. In other words, users only sign up for contributing data after developers provide an explicit and enforceable plan on how they will aggregate user data. Data aggregation pipelines in MapAggregate are implemented through a series of ephemeral and isolated serverless functions [332]. Users themselves own some of the pipelines. For the rest part owned by developers, exfiltration of private individual user data is limited prior to results aggregation. We implement and evaluate MapAggregate with the Amazon Web Services (AWS) Lambda platform, demonstrate its low cost and system latency, and discuss its resilience to various privacy attacks.

At a high level, in contrast to traditional architectures, which often let developers first obtain data and then analyze it at their discretion, MapAggregate requires developers to declare a formal plan of their intended data aggregation behavior as an application *manifest* explicitly and up-front. The manifest (Fig. 6.1) specifies how the data will undergo two key processing pipelines – a `Map` phase where data is pre-processed (e.g. "count empty parking spots from images" in our parking occupancy application) and an `Aggregate` phase where data across homes is combined (e.g. "report a noisy sum of number of empty spots in a 2 km radius over the past hour"). Both these phases are executed in the cloud through isolated serverless functions prior to data being handed off to the application. The rest of this paper dissects MapAggregate's design in

terms of two key challenges: ensuring privacy and efficiency.

Ensuring Privacy: MapAggregate ensures privacy of smart home data from individual homes with two important design choices. First, developers need to specify a concrete plan of how private user data will be filtered and aggregated. To this end, both the `Map` and `Aggregate` phase, specified in a text-based manifest, are essentially pipelines composed of a small number of chainable stateless operations with well-defined semantics. Fig. 6.1 shows an example of the `Aggregate` phase broken down into a series of simple steps (operations) – such as: pull (read every 60 min), aggregate (sum up number of parking spots), and noisify (round results). MapAggregate supports a small number of such operations that are open-source and easily verifiable, and yet be combined to enable expressive applications. We conducted an analysis of 80 smart city applications (§**??**) drawn from the research literature, and use these insights to inform MapAggregate's library of operations. We note that this approach of stitching together smaller operations to build more complex processing pipelines is loosely inspired by Unix pipes and a prior system for individual smart homes Peekaboo [**?**], with key differences being that the latter does not operate across multiple homes, aggregate data across homes/users or run in the cloud.

A second key design decision considers where and how to perform the `Map` and `Aggregate` phases that faithfully implement the processing pipeline as described by the manifest. Both phases are executed in ephemeral and isolated serverless functions [332], which only ingests, transforms, and outputs data, but cannot access persistent storage and network resources. MapAggregate runs the `Map` serverless function in a private cloud owned by each individual user (see Fig. 6.2). This ensures that raw data pre-processing is performed in a domain that is under the user's control. Next, MapAggregate restricts developers to run the `Aggregate` pipeline through a public serverless function. The public serverless function needs to be configured with a set of public permissions so everyone (e.g. third-party auditors) can inspect and verify its isolation. Combined, an individual user can only see their own data, and developers can only obtain aggregated results but cannot access the raw data or intermediate key-value pairs.

Enabling Efficiency: A key challenge in MapAggregate's design is that performing computation end-to-end every time for each aggregate may be resource and latency-intensive. Consider for example, two programs pertaining to parking spots – one for generating spatial maps and one for counting a grand total, both of which could in principle re-use the results of the same `Map` phase. Re-running `Map` seperately for each of these applications may be wasteful, since `Map` programs (e.g., visual object counting) are typically computationally expensive, while Aggregate programs in contrast (e.g., sum, average) are relatively lightweight. MapAggregate addresses this by caching `Map`'s computation results in a database within a user's private cloud (see Fig. 6.2). Further, MapAggregate can compute the intermediate results within a `Map` pipeline (e.g., parking occupancy information in each image) incrementally to reduce latency and reuse stored intermediate results to save computation resources for future `Aggregate` instances that may benefit from data re-use. We note that given that the private cloud is entirely in the user's control, cached data

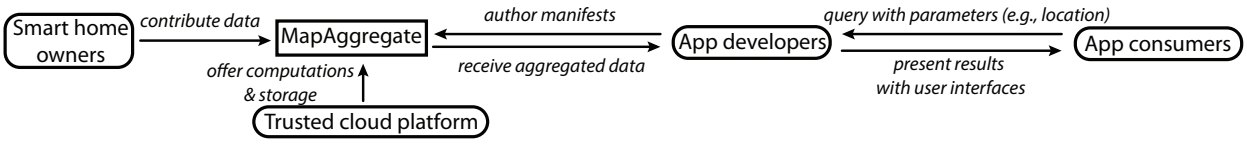


Figure 6.3: MapAggregate involves four stakeholders: (1) smart home owners that contribute data, (2) app developers who author the manifests, (3) a trusted cloud computing provider, and (4) app consumers that query with parameters (e.g., location, radius).

will not directly be accessible by developers. This means that developers are unable to explicitly re-use data without accessing `Map` each time. Further, policies for how long data must be retained in databases are inferred automatically by MapAggregate based on the manifests.

We implemented and deployed the MapAggregate architecture on Amazon Web Services (AWS) Lambda platform [10]. We implemented the manifest on top of NoFlo [273], a JavaScript implementation of Flow-Based Programming. We discuss how MapAggregate mitigates different types of privacy attacks. To demonstrate the ability to deliver expressive applications, we implemented and evaluated 5 end-to-end MapAggregate apps, covering different data types (imagine/audio/tabular), computational costs, and storage requirements (real-time or caching in personal cloud storage). We also showed that most MapAggregate applications can respond to a query between 0.5 second and 3 seconds, at a cost below $0.005, thus demonstrating that structured and isolated data aggregation through MapAggregate is feasible.

This paper makes the following contributions:

- A novel software architecture, MapAggregate, that only allows developers to query aggregated data insights in a manner that is controlled and configured by these users in the interest of privacy.
- A study of 80 smart city applications to characterize the feasibility of leveraging personal sensors for participatory sensing and the implementation of privacy-sensitive data aggregation.
- An end-to-end implementation of five MapAggregate apps and a detailed evaluation of these applications, demonstrating the feasibility of (1) storing users' data in distributed personal storage and (2) aggregating users' data in isolated and structured serverless functions.

6.2 Assumptions and Threat Model

This paper envisions that future developers can access smart home users' data in a privacy-sensitive manner through a new architecture – MapAggregate (Fig. 6.3).

Overview and Stakeholders: We present MapAggregate's workflow with the parking occupancy histogram application as a running example. A developer first declares intended data aggregation behavior in a text-based manifest (Fig. 6.1) and publishes it to a trusted platform (e.g., a MapAggregate app store for

smart homes). Smart home owners can choose to install the app and agree to contribute data collectively to that developer if they are comfortable with the specifications of aggregation provided in the manifest. Later, when a city council wants to obtain a parking occupancy histogram of a specific region, the MapAggregate runtime pulls data from participating smart homes, aggregates users' data in isolated sandboxes, and only releases aggregated results to the developer (Fig. 6.2). The developers then present aggregated data in their user interfaces to the city council.

Assumptions: We make the following assumptions about the capabilities of smart home devices and compute infrastructure. First, we assume smart home devices would have public facing APIs that MapAggregate can leverage [383]. Devices can push captured data at a designated time interval, or allow an authorized program to pull the latest sensor readings. Second, smart home owners, whose privacy we desire to protect, are trusted. Therefore, they do not fake data. We discuss the data integrity issues and potential solutions in §7.10. Third, cloud service providers, who offers isolated serverless functions, are trusted. Future cloud computing will become a utility [20, 188]. Like today' s electricity companies, cloud computing infrastructure would be heavily regulated and audited. We assume that the infrastructure will faithfully execute users' configurations. Recent research identifies a few system security flaws of commercial serverless implementations' isolation mechanisms (e.g., [397]). We assume that cloud service providers will patch potential system security flaws. Finally, we assume that there is an incentive system that properly accounts for users, app developers, and app consumers, which will encourage users to participate in the ecosystem.

Threat model: Allowing developers to access users' private data streams to build apps inevitably introduces more privacy risks for users and bystanders. MapAggregate's design goal is to enable a diverse set of smart city applications, while also mitigating these privacy risks to a reasonable level. Here, we enumerate the main privacy threats caused by different stakeholders.

- **A.1**. Developers might consolidate data from users and save it to their private machines, for potential future analysis and data reselling [80, 304]. For example, developers may use the vehicle brands in captured images to infer the household income and share it with an advertising network.
- **A.2**. Malicious app consumers might track a particular individual. For example, a thief may use various cues to infer whether a house owner is at home, such as lighting at the night, parking information in front of the home. Or, an app consumer may query the network to stalk a celebrity or an intimate partner [354].
- **A.3**. Malicious app developers may disguise their data aggregation by misleading users about an app's functionality. For example, a developer might develop a "stalking" app but label it as a "parking" app, thus deceiving users into installing it.

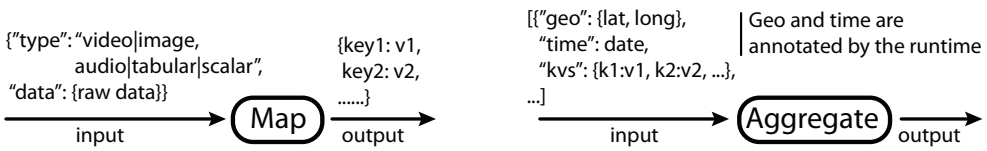


Figure 6.4: MapAggregate uses a generic tabular data structure between `Map` and `Aggregate`. The `Map` program transforms high-dimensional data into intermediate key-value pairs. The runtime merges key-values pairs from distributed sources into a table, annotates spatial-temporal metadata, and feeds it to the `Aggregate` program.

6.3 Programming Data Aggregation Manifest

Each app manifest is a machine-readable contract between users and developers. In the manifest, developers provide an explicit and enforceable plan on how they will aggregate user data. Once users install the manifest, they agree with developers' plan and start to contribute data to the developers' application. This section discusses the design of MapAggregate's manifest, which needs to be expressive enough to support a wide range of scenarios that are beneficial for privacy protection.

6.3.1 Structured stateless pipelines and data model

The key programming elements required to author a manifest are stateless and isolated operators, which only ingest, transform, and output data, but cannot access persistent storage and network resources. The source code of operators will be made open source and allow for verification and audits. This design prevents developers from saving data for other uses.

Operators are somewhat analogous to abstract classes in object-oriented programming [**?**]. MapAggregate clusters data transformation implementations with similar "verb" semantics into the same operator. Developers can specify the exact data transformation they want by configuring the properties of each operator. For example, the two detect operators in Fig. 6.1 and Fig. 6.5 annotate the hour based on the time and the street name based on the geolocation, respectively. In other words, many different but loosely related tasks, (e.g. "annotate the time" vs "annotate street name") could rely on the same operator ("detect" in this example), so long as they share similar semantics (i.e. inputs and outputs). This clustering reduces the API complexity for developers, while also allows third parties (e.g., auditors) to analyze the information flow in ways that are not easily possible if we allow developers to run arbitrary code.

Data model: MapAggregate has two types of operators: `Map` operators running in the users' private cloud and `Aggregate` operators running in a public cloud. `Map` operators take the input of raw data from smart home devices, which can be a video, image, audio file, or table. The `Map` operator transforms these rich high-dimensional data into a set of intermediate key-value pairs. Upon an aggregation request, the runtime

queries key-value pairs from qualified users, associates each with the time and geo-location of where the data was captured, and assembles them as a table (Fig. 6.4). The `Aggregate` operators then analyze this table to generate meaningful and privacy-sensitive insights. MapAggregate does not enforce the data format of the final output of each manifest. Since all operators have well-defined semantics, MapAggregate can infer the output format by analyzing the manifest.

6.3.2 Pre-defined `Map` operator

All `Map` implementations share the same semantics, namely, transforming high-dimensional data into key-value pairs, so there is only one `Map` operator. To specify the `Map` program, developers choose the core function (e.g., parking occupancy counting) and select the input/output data format (Fig. 6.1). Each new `Map` function can potentially unlock numerous application opportunities (Table. 6.1). We expect that the MapAggregate runtime can incrementally add new `Map` functions, expanding the platform's capability.

The `Map` operator is isolated, which only derives information from the input data and does not rely on external knowledge. For example, a parking occupancy counting operator counts the number of available parking spots in a photo. The gunshot event detection operator only detects a relative timestamp of a gunshot event. In other words, `Map` operators are not provided extraneous information such as home location and the current timestamp.

6.3.3 `Aggregate` operators

In contrast to `Map`, there are multiple `Aggregate` operators with different data transformation semantics. To specify an `Aggregate` function, developers choose from a small set of such operators, each with open-source implementations, authoring a stream-oriented pipeline similar to Unix pipes (Fig. 6.1).

Why Pipeline v.s. Database engine?: An alternative architecture design is using an SQL-like database engine. For example, we may allow developers to author `Aggregate` functions as SQL-like queries and create a SQL database engine to interpret the declarative queries. We choose this operator-based pipeline approach for two reasons. First, it is more flexible for the app store to extend supported data transformations by adding/removing/updating operator implementations. For example, the app store may add a pathfinding algorithm to enhance the aggregation capability. In contrast, modifying a database engine is non-trivial for most organizations. Second, the pipeline approach is more verifiable than the database approach since the runtime simply connects data transformation functions into a pipeline. Outsiders can verify the runtime by verifying the pre-loaded implementation of each operator (which are all open-sourced). In contrast, the complexity of a database engine makes it hard to analyze and verify the engine's behavior [355].

Reusable Operators: MapAggregate aims to use a small set of operators to help developers articulate their intended data aggregation behavior and equip them with more privacy support. We used best practices in

API design (e.g., [34]) to guide the design of these operators. We started with a few use cases in Table 4.1, programmed them using our initial API, and iterated on the API as we expanded the supported use cases.

At a high level, a typical `Aggregate` pipeline performs three tasks: querying the intermediate results processed by `Map` functions, annotating the geospatial-temporal information (e.g., street, zip code, day), and aggregating the combined data to derive useful and privacy sensitive insights. To better understand these primitives of `Aggregate`, we consider the road snow clearing application in Fig. 6.5 as a running example. Developers first configure the "duration" and "interval" of the *pull* operator, which informs the runtime on how the personal cloud should compute the intermediate results and store them in the database. The *pull* operator also accepts two dynamic parameters (i.e., "center" and "search radius"), which are required parameters in each aggregation request. The *detect* operator then annotates the geospatial-temporal fields (e.g., street name) and inserts the results as a new column. Finally, the following *select*, *aggregate*, and *rank* operators rank the streets by the cleaning time and return the road that are cleared first after a winter storm. A more comprehensive set of operators to cover various other scenarios for `Aggregate` are enumerated in Table 4.3.

Asynchronous data flows: Many smart city applications need to aggregate asynchronous data flows. Consider a gunshot localization app. Distributed edge devices discover gunshot events from captured audios and send the event information to a centralized server. The centralized server maintains a temporary state to check if these events are in the same time window and then runs trilateration to infer the location. However, this design is not possible in stateless aggregation pipelines. MapAggregate does not have a *push* operator to accept the results pushed by distributed `Map` functions. Instead, developers save this gunshot event information into users' private databases, and the `Aggregate` pipeline synchronously pulls these data from the databases to aggregate them.

Privacy-sensitive aggregation: An important implication of aggregating real-world data is that many values are non-negative integers (e.g., the number of parking spots). For example, consider a parking occupancy counting app that sums the available parking spots of five homes and returns 0. Although the app only discloses aggregated results, developers can easily infer that all five homes have no parking spots available. To mitigate this issue, MapAggregate requires developers to append an *noisfy* operator before disclosing raw statistics (e.g., average, mean, sum, count). For example, the *noisfiy* operator in Fig. 6.1 rounds the number of parking spots to further enhance privacy.

6.4 Executing Data Aggregation Manifest

Given a manifest, the app store then executes the declared data aggregation plan in a privacy-sensitive manner and hands the results to developers (Fig. 6.2). This section discusses the design of MapAggregate's runtime and the rationale and tradeoffs behind our design. Specifically, we focus on three key challenges

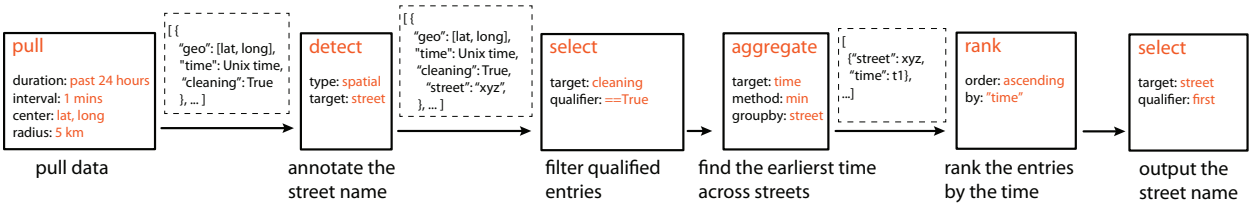


Figure 6.5: An aggregation pipeline that answers "Which roads are cleared first after a winter storm?" Operators come with a few configurable properties (e.g., duration interval in the pull operator) and dynamic parameters (e.g., center and radius in the pull operator). Operators are open source to help with verification of their behaviors.

Table 6.1: Operator descriptions and supported parameters. Developers can specify desired data transformation by configuring the properties of each operator. The runtime later maps the manifest specification to verified open-source implementations based on the properties and assembles them into a data aggregation pipeline.

#	Operator	Description	Main parameters
1	Pull	Specify the data pulling behavior in both spatial and temporal domain	Center, radius, interval, duration
2	Detect	Annotate the geospatial-temporal fields with more properties and insert these properties as new columns	Annotation types, target field
3	Rank	Rank rows by one of the fields	Order, target field
4	Aggregate	Aggregate data across multiple rows	Method, target field, groupby
5	Predict	Predict data points using Time Series analysis	Parameters for an ARIMA model [343]
6	Select	Select rows from a table based on matching fields	Target field, qualifier
7	Retrieve	Retrieve the target columns	Target field
8	Noisify	Inject noise to target values	Target field, noisify method

around executing a manifest: (1) ensuring that data is only used for the declared data aggregation, (2) ensuring that the declared data aggregation is privacy-sensitive, and (3) enabling efficiency.

6.4.1 Enforcing declared data aggregation

The common way to aggregate smart home data is to pull data from distributed devices (e.g., video doorbells), save data on a centralized server, and let developers write programs to analyze the data at their discretion. However, once the data leaves users' control, it is challenging to have any assurance of privacy. Developers may save the data, combine it with other datasets, and use it in other contexts. In contrast, MapAggregate enforces data aggregation through a combination of three ideas: user-controlled cloud storage, structured aggregation pipelines, and isolated serverless functions.

User-controlled cloud storage: A significant portion of smart city applications (e.g., parking occupancy statistics in the past year) needs to aggregate information across time, requiring the system to keep the users' data in persistent storage. Developers today often store these data either on their cloud server or edge devices. Cloud storage is more scalable and cheaper but also causes significant privacy concerns. In contrast, edge storage can be costly but can better address privacy concerns of the users. As an alternative to the above two options, MapAggregate saves users' data in their own personal cloud storage, enjoying the advantages of both options. MapAggregate assumes that future users own an independent cloud account (e.g., AWS account), so they can access personal cloud computing and storage resources.

Structured aggregation pipelines: Given a manifest, the MapAggregate runtime (Fig. 6.6, 6.7) assembles `Map` and `Aggregate` pipelines (i.e., executables) using open-source and verified operator implementations (e.g., through an MD5 checksum). In doing so, MapAggregate can enforce the declared data aggregation behaviors in the manifest. Since all the operators are stateless, the assembled end-to-end pipeline is also stateless.

Isolated serverless functions: MapAggregate runtime encapsulates `Map` and `Aggregate` pipelines into independent serverless functions. Serverless computing is a cloud computing execution model in which cloud providers allocate machine resources on-demand on behalf of their customers [332, 384]. Developers only need to upload the programs to a serverless platform (e.g., AWS Lambda, Azure Functions, Google Cloud Function) and pick the event that should trigger their execution (e.g., an image push event). The serverless infrastructure then encapsulates these programs in isolated containers and handles everything else: instance selection, scaling, deployment, fault tolerance, monitoring, logging, security patches, etc. In MapAggregate, these serverless functions pull data from edge devices and execute data aggregation pipelines end-to-end within isolated sandboxes, where even the function owner cannot access the data being processed.

These three ideas, combined, make the data aggregation behavior of each serverless function transparent to outsiders and assures users that both the app store and developers can only access the aggregated

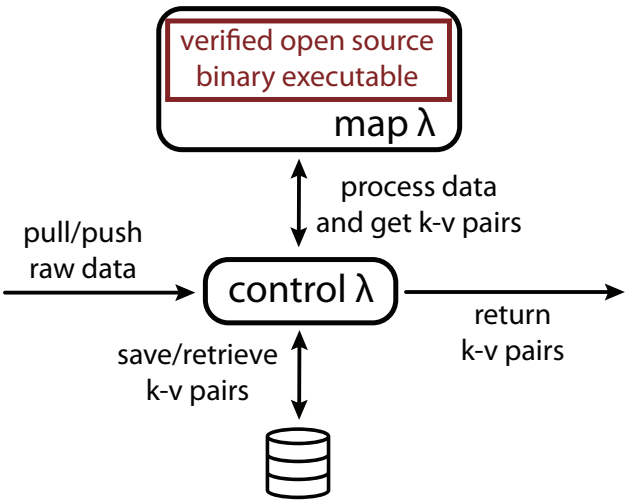


Figure 6.6: Raw data collected by smart home devices goes through two serverless functions (λ) before leaving users' private cloud. First, the control λ pre-processes the image to remove sensitive personally identifiable information (e.g., face, license plates). Second, the `map` λ transforms the high-dimensional data into a set of key-value pairs and return it to the control λ. The control decides whether to save the key-value pairs to a private database per the specification in the manifest.

data.

6.4.2 Ensuring privacy of declared data aggregation

Enforcing declared data aggregation behaviors does not necessarily guarantee that the aggregation itself is privacy-sensitive. For example, outsiders may run a reconstruction attack [130] by querying multiple aggregate statistics and using these results to reconstruct private data. Or a user may not fully understand the implications of the manifest and installs an application that may leak important information unintentionally. A completely fool-proof mechanism to mitigate both types of attacks is quite challenging. For instance, one

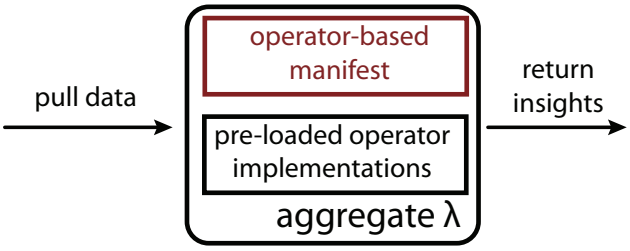


Figure 6.7: Given a manifest, the runtime assembles an aggregation pipeline using pre-loaded verified operator implementations. The pipeline combines the key-value pairs to derive aggregated insights in an isolated sandbox (i.e., a serverless function), so developers cannot access the data being aggregated.

can never quite fully predict all forms of private data that a user may unintentionally leak. Nevertheless, MapAggregate provides meaningful default safeguards that limit the kinds of private user information that adversaries can extract or infer from aggregate data. The MapAggregate runtime introduces three more privacy-protection layers to mitigate such potential privacy risks.

Sensitive PII pre-filtering: Prior to sending private data to the `map` function, MapAggregate routes the raw data from edge devices through a control function (Fig. 6.6) to remove a set of highly privacy-sensitive information (e.g., faces, license plate numbers), similar to the blurring in Google Street View Imagery [129]. The control function is also a serverless function, and is a part of the runtime within the user-controlled private cloud. This pre-filtering prevents developers from building stalking apps.

Data control policies: MapAggregate further allows users to control how developers can aggregate their data in two dimensions: the freshness of the data and the minimum number of sources in each aggregation. For example, users can specify that developers can only query data captured 24 hours ago, or MapAggregate must aggregate users' data with at least $N-1$ other users before making it accessible to developers. Since data are stored under users' control, and the structured information flows are public, MapAggregate can stop contributing data to the end-to-end pipelines if the manifest does not match users' preferences.

Data source sampling: One classic attack is that developers may compute individual data using the delta between two aggregated results. For example, a developer may query the total number of parking spots from M homes and $M+1$ homes, and use the difference to infer the number of parking spot in front of a specific home. To mitigate this attack, MapAggregate runtime only allows developers to query homes through the region parameters (i.e., center and radius), and the runtime can only return an aggregated results of N homes if there are at least 2N qualified users in the query. We discuss the privacy guarantees of these designs in §6.6.

Another potential attack is that developers may query MapAggregate repetitively to construct a set of solvable equations/inequalities. MapAggregate prevent this attack by capping the number of unique aggregations each piece of data can participate to $N-1$. In other words, if a user specifies that her data must be aggregated with at least 5 other homes, her data can only participate in 5 unique data aggregations. In doing so, outsiders can at most construct 5 equations/inequalities, so they can not use them to solve 6 unknown variables [24].

6.4.3 Enabling efficiency through users' private cloud

An essential challenge of making MapAggregate runtime practical is its efficiency. As a side effect of giving data control back to users, developers only have limited access to user-controlled storage and sandboxed computing environments. Therefore, they cannot optimize the system efficiency as much as they are able to do in conventional architectures. This shift requires MapAggregate to incorporate efficiency

considerations into the architecture.

Intermediate results caching: Performing data aggregation end-to-end in an isolated environment can be resource and latency-intensive. Consider the parking occupancy histogram application described earlier (see Fig. 6.1) as an example. The runtime here must process `365 days x 24 hours x the_number_of_involved_devices` images each time. Further, if the city council issues queries a few times each week, the runtime needs to always re-compute the results from scratch. Instead, MapAggregate executes `Map` programs and stores the intermediate results in the users' private cloud (Fig. 6.2). When a user chooses to install a manifest, MapAggregate loads the verified `Map` implementation specified in the manifest (e.g., the parking occupancy counting program), spawns a `Map` function in a private cloud that the user controls, and saves the output of the `Map` function to a private database that is still within the user's own private cloud.

This design offers a few critical advantages: (1) First, it reduces the data that needs to be stored, from hundreds of kilobytes of images to a few bytes of key-value pairs. (2) Second, it reduces the latency in responding to a query and avoids repetitive computations. For example, the parking occupancy histogram app can query results from the private database rather than always processing all the images. (3) Further, data caches also improve system availability. Should the smart home network be offline for any given reason (say, a power outage at the home), input data caches at the cloud can still be used to deliver many classes of applications (apart from the extremely time-sensitive ones that require real-time data).

Autonomous on-demand storage & computation: MapAggregate only saves and analyzes data that is necessary for installed manifests in the users' private cloud. Since each manifest already articulates what data the developers need from users (see §**??**), MapAggregate does not need to store all smart home sensor data blindly. For example, to provide data to the parking occupancy app, the MapAggregate runtime only needs to pull and process data at a one-hour interval, and save the data for up to one hour. Although maintaining a personal cloud may sound intimidating for layman users, the MapAggregate runtime operates autonomously and could potentially be used to launch and maintain the user's personal clouds in a manner that requires minimal intervention by users.

6.5 Implementation

We use an open-source domain-specific language [29] for flow-based programming to define the manifest, which specifies how the operators are chained together and the properties of each operator. We use a web-based flow editor UI (i.e., Node-Red [170]) as the programming environment to author a manifest. We implement and deploy the MapAggregate runtime on AWS Lambda [10] using two AWS accounts: one to emulate users' private cloud and the other to emulate developers' public cloud.

Private cloud: We implement both the `Map` and `Control` functions using python and execute them using the default containers offered by AWS. Note that the `Map` functions may not be constantly active if it serves

a less populated area. The cloud service provider would re-initialize these stateless functions frequently. A challenge is that many machine learning implementations may only take 10 ms to classify an image but require 10 seconds to initialize the environment. To address this issue, we compile machine learning models in `Map` functions as binary executables using Apache TVM [55].

Public cloud: We implement all `Aggregate` operators in Javascript and used a lightweight flow-based programming runtime (NoFlo [274]) to assemble these operators into an executable. Meanwhile, each public cloud maintains a database of participating users' metadata (e.g., private cloud endpoint URLs and each home's geolocation) using Amazon DynamoDB. The runtime then creates an API gateway accepting consumers' query parameters and forward them to the `Aggregate` function.

Data flow encryption: The data flow between the private cloud and public cloud is on an encrypted channel to prevent third parties and developers from intercepting the communication. When a user installs a manifest, MapAggregate runtime generates asymmetric keys between the user and the developer, saves the public key in the users' private cloud, and saves the private key in the developers' public cloud. For each data communication, the `Control` function on the private cloud encrypts the results use a public key and the `Aggregate` function decrypt the results use the private key. Note that MapAggregate saves the private key through AWS Key Management Service with the permission that only the `Aggregate` function can use the key, but developers cannot access it.

Edge emulators: Since a uniform hardware API standard for IoT devices is not well developed yet, we create a set of python programs as the device emulators. These emulators either respond to a pull request or send simulated data at a designated time interval.

6.6 Privacy model

In this section, we review how MapAggregate mitigates the privacy threats enumerated in §6.2 and discuss the privacy guarantees and new interactions around privacy that MapAggregate can offer.

6.6.1 Mitigating privacy threats

Table 6.2 summarizes MapAggregate's key techniques in mitigating different types of privacy threats. First, MapAggregate prevents developers from direct access to users' individual data (A.1 in §6.2), including raw data and denatured key-value pairs. Users only disclose denatured key-value pairs to the `Aggregate` function on an encrypted channel, and the aggregation phase happens in an isolated sandbox where developers cannot access the data being processed. While developers can potentially break the isolation by modifying the configurations of the `Aggregate` serverless functions (e.g., leveraging logging functionalities to save data), the `Control` λ verifies the configuration of the public `Aggregate` function before disclosing

Table 6.2: Privacy enhancing techniques that mitigates different privacy threats.

Privacy threats (see details in §6.2)	Techniques
A.1 Developers may save users' data for future usages.	Preprocessing in a private cloud, encrypted data flow, stateless data aggregation
A.2 App consumers may abuse the network for stalking.	Sensitive PII pre-filtering, privacy-sensitive data aggregation
A.3 Developers may hide their actual data aggregation intent.	Easy-to-analyze structured manifests

data.

Second, App consumers are highly limited in their ability to track a particular individual (A.2 in §6.2) through the MapAggregate sensor network. The `Control` function removes sensitive personally-identifiable information (PII), e.g., faces and license plates in images, in the raw data so the later phases cannot track an individual. Note, however, that we did not exhaustively implement all the PII filters in MapAggregate. We discuss the potential PII pre-filters in §7.10. Further, the privacy-sensitive data aggregation makes it hard for app consumers to infer individual data, even in corner cases (e.g., none of vehicles are parked on the street).

Third, data aggregation behaviors in MapAggregate are governed by a structured manifest that describes the precise series of operations performed both at the personal cloud and the public cloud. Essentially, the manifest is a publicly available text document authored by an application developer and tied to a single application, e.g., parking occupancy. Instead of letting developers describe their intended data aggregation behaviors in the arbitrary text (A.3 in §6.2), which can hardly be verified, MapAggregate auto-generates simple text-based summaries such as "this app aggregates your data with five other homes in your street to present approximate street parking counts" by analyzing the manifest. Similar efforts to develop privacy nutrition labels [110, 112**?**] can also apply to MapAggregate.

6.6.2 Privacy guarantees for individual homes

The precise privacy guarantees provided by MapAggregate are highly dependent on the manifest itself – the *aggregate* and *noisify* operators involved, and the control policy – the data freshness and the number of homes aggregated. Our `Aggregate` functions are carefully chosen so that the risk to privacy, should any one of the inputs be compromised by an end-user entity can be readily characterized, given a manifest and a control policy. In the interest of brevity, we do not present an exhaustive description of such guarantees for all possible manifests and policies. Instead, we provide a brief characterization of the kinds of guarantees possible for direct sum, among our most general `Aggregate` functions.

Through MapAggregate, developers can obtain two types of information: the collection of involved

homes in each query and the aggregated results. The privacy protection goal here is to prevent developers from inferring the individual data (e.g., whether the parking spot in front of a specific user's home is occupied).

Malicious developers want to compute individual data using the delta between two aggregated results. Assume that $2N+2$ homes (S) have participated in a real-time parking occupancy application, and each home obtains a parking occupancy number X_i. They then study the spatial distribution patterns and craft two set of queries: one returns the sum of N homes ($U \subset S\ with\ \#U = N$) and the other returns the sum of $N+1$ homes ($V \subset S\ with\ \#V = N+1$). The probability that U is a subset of V is: $\frac{C(2N+2,N+1)\times(N+1)}{C(2N+2,N+1)\times C(2N+2,N)}$, where $C(2N,N)$ denotes the number of ways of choosing N items from $2N$ items. If a user sets N=10, the probability that a developer can obtain such two queries is 0.0017%. Note that obtaining such two queries does not guarantee that the malicious developers can infer the value of the "delta" home since a *noisify* operator further injects noise into the aggregated results. Therefore, a malicious developer must obtain such a query multiple times to remove that noise.

6.6.3 Privacy interaction model

The MapAggregate manifest is a public, machine-readable document that specifies how the smart home owner's data will be processed at every stage and the precise transformations it undergoes at the personal cloud, public cloud, and until it reaches the end-user. The MapAggregate runtime further assures that developers can only aggregate data as specified in the manifest. MapAggregate enables a new type of privacy interaction model, where developers' data aggregation behaviors are transparent to outsiders, users can enforce and control how their data would be aggregated, and third-party audits can easily audit developers' data aggregation behaviors at scale.

User consent & control: The manifest therefore acts as a data practice specification document that home owners must explicitly agree to, prior to their data being used. Once all home owners are in agreement, MapAggregate faithfully executes the data processing pipeline precisely as specified by the manifest. For example, a user can specify that developers can only access her data older than 1 hour, which must be aggregated into fuzzy categorical values (e.g., >5 parking spots), with at least five other homes in the same neighborhood, before making it accessible for outsiders.

Third-party audits: MapAggregate manifests can also support regulators, third-party auditors and consumer organizations. Reviewing a text-based public manifest allows for a more informed understanding of app behavior, without the need to review proprietary code. Indeed, MapAggregate can provide the path forward for a structured and transparent regulatory mechanism for major technology players to handle aggregate smart home data.

Developer decision making: When authoring an manifest, developers decide privacy-utility tradeoffs at the

Table 6.3: Five end-to-end applications we have built.

#	Application description	Sensor	Data type	Computation	Storage
1	Real-time parking occupancy	Video doorbell	Image	Heavy	N
2	Yearly parking occupancy stats	Video doorbell	Image	Heavy	Y
3	Gunshot localization	Video doorbell	Audio	Medium	Y
4	Real-time noise index heatmap	Video doorbell	Audio	Lightweight	N
5	Trendy smart TV search queries	Smart TV	Table	Lightweight	N

risk of being flagged by third parties and rejected by users. Namely, if a developer does not spend enough effort to protect users' privacy, users may not install their manifest.

6.7 System evaluation

This section presents experimental evaluations of MapAggregate. We created five end-to-end MapAggregate apps and used these apps to demonstrate the feasibility of running participatory sensing applications in MapAggregate. Our results show that popular MapAggregate apps can respond to queries between 0.5 seconds and 3 seconds, at a cost below $0.005.

6.7.1 End-to-end MapAggregate apps

We implemented five end-to-end MapAggregate apps, covering different data types (imagine/audio/tabular), computational costs, and storage requirements (real-time or caching in personal cloud storage).

- **Real-time parking occupancy** reports the total number of available parking spots in front of sampled homes in a designated region. We implemented the `Map` function by training a ResNet-50 model [154, 227] and deployed it in Lambda functions with 10 GB memory. Since `Aggregate` functions are lightweight, we deployed them in Lambda functions with 2 GB memory.
- **Yearly parking stats** summarizes the hourly street parking usage in the past year. This app uses the same `Map` function as "Real-time parking occupancy," but it does not run in real-time. Instead, it computes parking occupancy incrementally and saves the results to a private database. The `Aggregate` function later pulls data from the database and aggregates them into a histogram.
- **Gunshot localization** uses the audio captured by distributed video doorbells to locate the location of a gunshot event. The `Map` function detects the precise timestamp of the gunshot event, and the `Aggregate` function combines timestamps from distributed homes to run a multilateration [386]. We implemented the `Map` function using an open-source project [257] and deployed it in Lambda

functions with 4 GB memory.

- **Real-time noise index heatmap** uses the audio captured by distributed video doorbells to generate a real-time heatmap of noise index. The `Map` function computes the the sound loudness, and the `Aggregate` function combines them to output a heatmap. We implemented the `Map` function in Python and deployed it in Lambda functions with 2 GB memory.
- **Trendy smart TV search queries** collects users' search queries on their smart TVs and outputs the most popular search queries. The `Map` function computes the term frequency of individual users' historical queries, and the `Aggregate` function merges them and ranks the term frequency. We implemented the `Map` function in Python and deployed it in Lambda functions with 2 GB memory.

6.7.2 Evaluation goals

MapAggregate has two unique designs compared to past systems: (1) storing users' data in distributed personal storage and (2) aggregating users' data in isolated serverless functions. Both changes can potentially impact latency and cost. First, serverless functions take extra time to initialize the resources. Second, breaking functionalities into independent isolated functions, namely `Control`, `Map`, `Aggregate`, introduces more communication latency than communicating through shared memory on one single machine. Third, MapAggregate users need to pay for the storage and computation of their private cloud. We used these five example apps to characterize the latency and cost of MapAggregate apps.

6.7.3 Method & Metrics

We simulated 200 participating homes distributed randomly in a 589-acre neighborhood, each running an independent private cloud. We then used Google StreetLearn dataset [254] (i.e., Google Street View images) as the image input, a noise audio data set [257] as the audio input, and TREC 2009 Million Query dataset [49] as the tabular data input.

Upon a data aggregation query, the `Aggregate` function randomly selected 10 homes from the 200 participating homes and queried their `Control` function in parallel. We ran the experiment in two setups. First, we keep all Lambda functions "warm" before starting the experiments. This emulates the scenario that a MapAggregate app runs in a busy region, where app consumers frequently query the service. Second, we do not invoke Lambda functions for 30 minutes before starting the experiments, so all Lambda functions are "cold". This emulates the scenario of long-tail apps where only few users query it infrequently.

We repeated the process ten times for each application and measured the latency of each step. Specifically, we broke the end-to-end latency into four components: (1) the execution time of a `Map` function, (2) the total time of the MapAggregate runtime calling individual private clouds and returning results to the `Map` function, (3) the execution time of an `Aggregate` function, and (4) the communication time between app

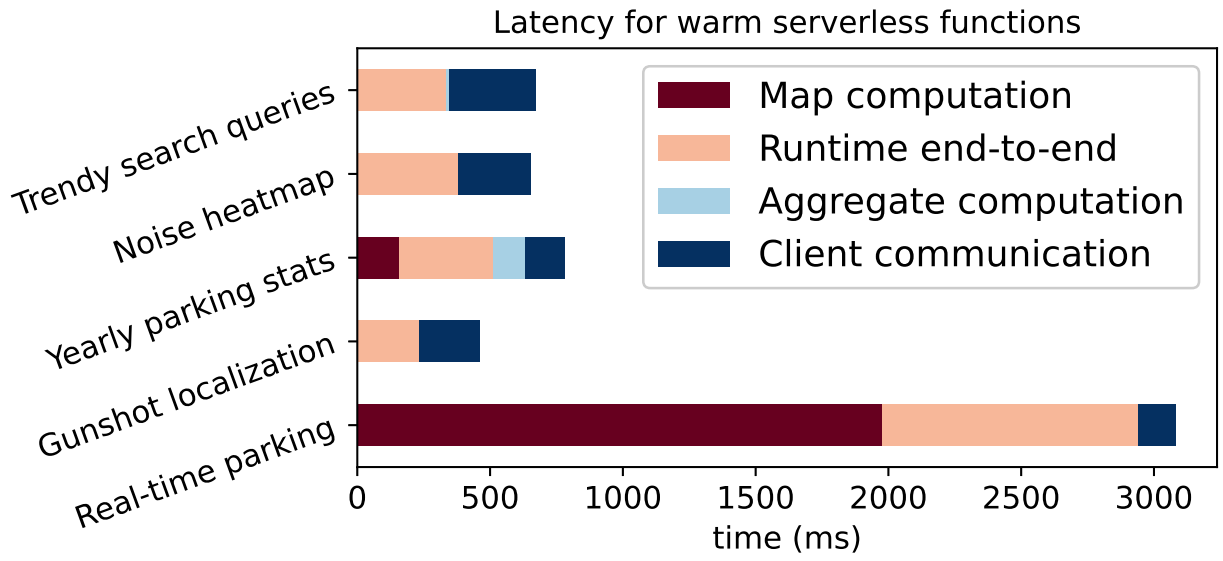


Figure 6.8: Average latency breakdown for popular apps.

consumers and the `Aggregate` function. We later used the fine-grained execution time and actual storage to compute the cost for running MapAggregate apps.

6.7.4 Latency

Fig. 6.8 presents the latency breakdown for popular apps. All apps except the "real-time parking occupancy" app can respond to queries under 700ms. The "real-time parking occupancy" consumes more time ($\approx$ 3s) since it always processes the latest data. In contrast, the "yearly parking stats" queries the cached results in the private database rather than running the expensive `Map` function. Further, running the ResNet-50 model is the most expensive computation task among all `Map` functions, consuming around 2 seconds. Note that traditional crowd sensing apps often suffer from much larger latencies (e.g., $\approx$35s in [405], $\approx$80s in [144]) since they need to pull data from user-owned edge devices.

Fig. 6.9 presents the latency breakdown for long-tail apps, such as an app that serves a rural area. Waking these cold serverless functions takes significant time, especially the heavy computing `Map` functions. For example, both *Gunshot localization* and *Real-time parking* require to run machine learning models. Even though we have compiled these models as binary executable, they take around 25 seconds to activate.

6.7.5 Cost

Table 6.4 presents the individualized costs for running the Map function, the storage, and the Aggregate function. Note that users pay for the `Map` phase and the storage, and developers pay for the `Aggregate`

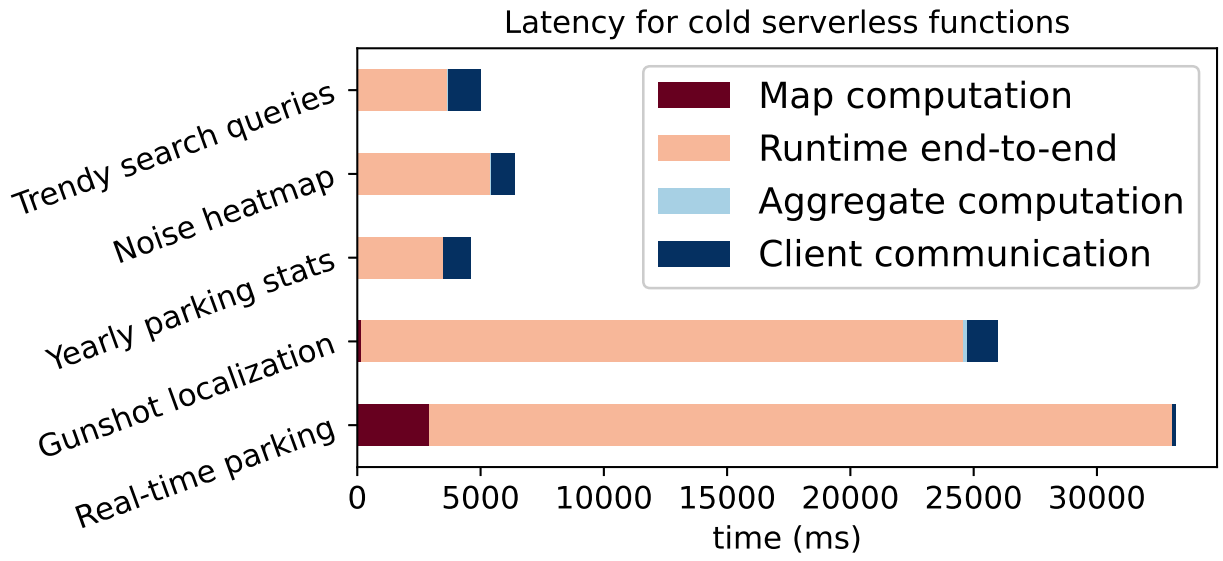


Figure 6.9: Average latency breakdown for long-tail apps

phase. Assuming that the real-time parking occupancy app pulls data from 10 homes, the total computational cost to execute a query through MapAggregate is only $0.0043. Or imagine a yearly parking stats app that pulls data from 10 homes, computes parking occupancy information each hour, and saves the results to a private database. The cost of running such an app for a year is capped at $366, assuming that no other apps reuse the intermediate results. In contrast, it often takes a few thousand to hire land use consultants to prepare a manual parking analysis for one individual location [124].

Table 6.4: Computation cost per invocation and storage cost per user in $100,000^{th}$ of 1 U.S. dollar.

App name	Map		Storage	Aggregate	
	Warm	Cold		Warm	Cold
Real-time parking	41.78	446.63	0	9.82	91.94
Parking stats	0.31	0.31	2.94e5	0.8	11.3
Gunshot localization	5.96	60.17	0	8.06	81.96
Noise heatmap	0.81	1.92	0	1.28	17.81
Trendy search queries	0.64	0.86	0	1.18	12.31

6.8 Related Work

Related work falls under three broad categories:

Privacy-sensitive databases: MapAggregate is inspired by the extensive prior work on building privacy-sensitive database systems [7, 251, 321]. For example, Airavat [321] integrates differential privacy into the MapReduce programming framework [82], offering privacy guarantees by inserting random noise into its outputs. These solutions often assume that the centralized database is trusted, but people who query the system are untrusted. In contrast, MapAggregate seeks to put data control back to users, so it does not trust developers who consolidate users' data in a centralized server. MapAggregate achieves this through two new designs: saving users' data in distributed personal clouds and aggregating users' data in isolated end-to-end serverless functions. Further, instead of only focusing on tabular data, MapAggregate also extends privacy protection to high-dimensional data (e.g., image, audio).

Enforcing data control: Several solutions have been proposed to enhance users' control of their data [81, 153, 193, 284, 366]. One example is to process all user data at user-controlled resources (e.g., edge devices, users' cloud resources [193, 284], etc.), thereby avoiding sharing data with anyone. Another approach is to use remote attestation to inform users about the server-side software stack (e.g., Ironclad [153], RiverBed [366]). However, these approaches can only inform users whether the data is being leaked, but do not delve into finer-grained actions on data (e.g., how user data is being aggregated).

The approach of chaining operators into more complex processing pipelines, and representing the pipeline in a text-based manifest, is inspired by Peekaboo [**?**], which uses an in-home hub to pre-process single homes' outgoing data in an enforceable manner. In contrast, MapAggregate's key innovation is to enforce data aggregation across multiple homes using isolated and structured stateless pipelines in the cloud.

Differential privacy: An alternative approach to data aggregation privacy is differential privacy, which has two main models: local differential privacy and global differential privacy. Local differential privacy solutions (e.g., Google RAPPOR [114], Apple Private Count Mean Sketch [15]) add a high degree of noise to ensure the data is privacy-sensitive even in the worst-case scenarios. The benefit of the local differential privacy solution is that users do not need to trust the data curator. On the other hand, global data privacy solutions (e.g., Airavat [321]) assume that the data curator is trusted. Since the curator knows the data distribution, the curator adds less noise, leading to more accurate results with the same level of privacy protection. MapAggregate's trust model is somewhat in between - it does not trust the data curator but assumes that the cloud service provider is trusted (similar to [193, 284, 366]).

In contrast to differential privacy, MapAggregate does not always add noise to individual data entries. Instead, MapAggregate protects individual privacy by combining three ideas: (1) isolating the data aggregation process, (2) disclosing only the aggregated result, and (3) limiting the number of queries for each data piece. In doing so, outsiders cannot infer individual user data by solving a set of equations. This approach

has a few advantages in constructing privacy-sensitive participatory sensing applications compared to differential privacy. First, many sensor network queries may only involve a small number of sensors, while different privacy often requires a large N to estimate accurately. Second, many app developers may not have enough resources to design sophisticated differential privacy, while the MapAggregate mechanism is more straightforward.

6.9 Discussion

Economics of MapAggregate: We believe that MapAggregate can be a win-win for both users and app developers alike. Sensing applications often entail significant infrastructure and maintenance costs. For example, acoustic gunshot detector systems are quite expensive to install and maintain, costing around $65,000–95,000 per square mile, per year [33, 240]. In contrast, MapAggregate relies on pre-existing personal smart home sensors. This significantly cuts overall system cost and maintenance overhead. Of course, not all smart home sensors may be at opportune locations for every application. However, this is offset to some extent by the rapidly growing scale of home smart home sensing infrastructure deployed by individual users globally. Further, hiding the data access behind a paywall also prevents outsiders from abusing the data access. We believe that MapAggregate provides a mechanism for smart home users to monetize the sensing infrastructure they own, although we leave the precise economics of this approach to future work.

Whitelist data access: Traditional approaches often rely on removing privacy-sensitive content from individual data, such as occluding persons' face [3, 369], blocking marked regions [303]. For example, Google Street View captures millions of miles of street view imagery and provides interactive panoramas to the broader public. To mitigate users' privacy concerns, it blurs persons' faces and license plates by default, and allows users to request to blur their houses [179]. Yet, these "blacklist" privacy protections are insufficient since developers can still infer numerous undesired insights from preprocessed data. In contrast, MapAggregate advocates for whitelist data access, requiring developers to explicitly articulate their desired data access and only allow these data access.

Data integrity: A potential limitation of any participatory sensing application is that one or more of the smart home participants may send inaccurate data to the cloud [104]. Verifying data integrity in this context is extremely challenging. One potential path forward may be to enable a set of auditing apps that explore the consistency of sensed data across neighboring homes.

Collusion & physical attacks: The privacy guarantee of MapAggregate relies on the fact that outsiders cannot construct enough equations/inequalities to solve a set of unknown variables. However, developers and app consumers can potentially leverage out-of-band knowledge to break this privacy guarantee. For example, developers may collude with some users and know the partial data these users send to the `Aggregate`

function. Another example is that an app consumer may park a few cars on the street to inject a controlled variable into the system. We acknowledge these attacks as a potential limitation of MapAggregate, albeit these types of attacks are expensive to conduct.

PII pre-filtering: One limitation is that we did not exhaustively enumerate all the potential personally identifiable information (PII) that MapAggregate must filter out. Sensed data may contain personally-identifiable attributes that are not always easy to predict up-front. For example, consider an app that stalks a celebrity by knowing they own the only Lamborghini in a region. Another example is that a thief may use the app to locate the police vehicles in the city. Future research may build a list of pre-filtering attributes to enhance privacy and prevent data abuse.

Note that pre-filtering attributes on high-dimension data are often challenging. For example, it is hard to pre-filter only luxury vehicles in an image. One potential solution is to leverage the MapAggregate architecture and mask protected attributes in the intermediate text-based key-value pairs.

6.10 Conclusion

This paper presents MapAggregate, a privacy-by-design architecture that only allows developers to query insights, on data aggregated across multiple homes, in a manner that is controlled and configured by homeowners. MapAggregate pools together data across multiple homes rather than a single one to mitigate privacy concerns. It achieves this through the use of serverless functions and private clouds, offering multiple layers of protection for users' data. We present a detailed implementation and evaluation that shows MapAggregate's system performance, privacy guarantees, and ability to deliver expressive applications.

Part 3

NAVIGATING THE DESIGN SPACE

Chapter 7

Lean Privacy Review: What Fine-grained Data Should Developers Collect?

Today, industry practitioners (e.g., data scientists, developers, product managers) rely on formal privacy reviews (a combination of user interviews, privacy risk assessments, etc.) in identifying potential customer acceptance issues with their organization's data practices. However, this process is slow and expensive, and practitioners often have to make ad-hoc privacy-related decisions with little actual feedback from users. We introduce Lean Privacy Review (LPR), a fast, cheap, and easy-to-access method to help practitioners collect direct feedback from users through the proxy of crowd workers in the early stages of design. LPR takes a proposed data practice, quickly breaks it down into smaller parts, generates a set of questionnaire surveys, solicits users' opinions, and summarizes those opinions in a compact form for practitioners to use. By doing so, LPR can help uncover the range and magnitude of different privacy concerns actual people have at a small fraction of the cost and wait-time for a formal review. We evaluated LPR using 12 real-world data practices with 240 crowd users and 24 data practitioners. Our results show that (1) the discovery of privacy concerns saturates as the number of evaluators exceeds 14 participants, which takes around 5.5 hours to complete (i.e., latency) and costs 3.7 hours of total crowd work ($\approx$ \$80 in our experiments); and (2) LPR finds 89% of privacy concerns identified by data practitioners as well as 139% additional privacy concerns that practitioners are not aware of, at a 6% estimated false alarm rate.

7.1 Introduction

Imagine a data scientist at Uber finds that users are more likely to accept surge pricing if their phone battery is low [62]. Incorporating this insight into the system may improve overall profits; however, it may also lead to negative headlines in the news media. Another example is the flawed launch of Google

Buzz [47] in 2010, in which Google used users' contact information gathered from Gmail to generate their connections in a social network service. Today, industry practitioners face a variety of similar dilemmas [8, 262, 331, 349, 391] in understanding how to best use potentially sensitive data in a manner that users will view as appropriate.

Navigating through these dilemmas requires significant expertise and effort [341]. Companies need to enumerate the range of problems pertaining to privacy, estimate the severity of each problem, and balance the potential fixes with competing interests (e.g., corporate revenue). Today, large companies like Google and Facebook have invested a great deal of resources in privacy. Privacy-related staff reside in both dedicated stand-alone teams and groups within other teams [37, 128, 266], providing decision-makers with tailored feedback through privacy reviews. A formal privacy review often involves multiple departments (e.g., legal, UX, business, public relations, domain experts) and multiple rounds of discussion [12, 38, 72] (e.g., privacy impact assessment, user interviews, factorial vignette surveys). Such a review often requires a few weeks' turnaround time [38] and costs $10,000 to $60,000 in human labor for companies [249].

This current approach to privacy reviews suffers from two major issues. First, since the privacy reviews are slow and costly, most small teams do not even have the resources to conduct such reviews. Even large organizations often only conduct comprehensive privacy reviews in a "sandwich" approach [37, 286, 356] (i.e., up-front specification followed by validation at the end). For many minor design decisions, practitioners can only offer their best guess (e.g., the "front page" test [291, 376]) as to users' expectations with little actual feedback from users [277, 394]. Second, experts conducting full privacy reviews often do not have the low-level technical knowledge of each data practice. The process requires practitioners to brief the data practice to experts and wait for experts' feedback to validate the final implementation, resulting in a considerable overhead to the whole organization [37].

This paper presents Lean Privacy Review (LPR), a new discount privacy assessment method that can collect direct feedback from users at a low cost and provide tailored feedback for a data practice design, without requiring an actual technology implementation. At the heart of LPR is a novel approach that guides crowd workers in **actively** examining a specific data practice for privacy concerns, which reduces the cost and required resources for privacy concern inspection. Figure 7.1 presents an overview of the workflow of LPR. LPR asks each crowd worker to examine a description of data practice and express their concerns in **free text**, rather than only through passive responses to Likert questions (i.e., factorial vignette surveys) [32, 334]. Although individual crowd workers may have limited privacy expertise and only find few privacy problems, our experiments (§7.9.3) show that collectively aggregating their privacy concerns can achieve good and consistent coverage.

The design and development of LPR were deeply influenced by lean startup[1] [313], agile software

[1] We borrow the term "lean" from the concept of "lean startup" since LPR resembles two important properties of a lean startup. First, LPR collects direct feedback from users to help uncover their actual privacy concerns. Second, the privacy concern collection

development [65], discount usability testing [269, 270], and speed dating [79]: small teams do short development cycles with progressively refined prototypes, getting customer feedback at each step. We envision that LPR would enable two types of usages. First, practitioners can use LPR to conduct discount privacy reviews when a formal privacy review is not available. Second, the low-cost nature of LPR makes it possible for practitioners to iterate their data practices based on fast privacy feedback, as well as explore a broader set of alternative data practices in the early stages of design. Here, LPR is not designed to replace full privacy reviews, but rather serve as a low-cost and complementary method to full privacy reviews, similar to how heuristic evaluation is complementary to full user studies.

Yet, asking crowd workers to actively examine a data practice is challenging. First, we lack an effective way to communicate data practices to non-tech-savvy users. For example, past work [139] has found that most non-specialists do not understand the details of the Facebook Cambridge Analytica data scandal [142, 365] even after being exposed to massive press coverage. Second, we need an approach to scaffold a privacy design review for non-tech-savvy users. Conducting a formal privacy review often takes hours for an expert. However, most users may not have the expertise to articulate a privacy problem and may be unable to engage for a long time to do so. Finally, interpreting the free-text responses is not easy for practitioners, who may lack the time and the expertise to analyze the qualitative responses. The rest of this paper describes our solutions (§7.4 - §7.6) to these key challenges in making LPR practical and a web-based system (§7.7) to support LPR.

7.1.1 A usage scenario of LPR

A data scientist (i.e., the data practitioner) in a search company wants to run an A/B test to determine which color is the best for web hyperlinks [21, 162]. On the one hand, the data scientist feels that this is a rather innocuous experiment, especially since users will not face any direct financial loss. On the other hand, the data scientist worries that this experiment does not explicitly obtain users' consent, and some users may feel manipulated. Figure 7.1 illustrates her interaction with LPR to evaluate the privacy concerns of her proposed data practice design.

Privacy storyboarding (§7.4). LPR provides a worksheet (Appendix §10.1) and a web interface (Fig. 7.8) to help data practitioners think through a planned data practice and generate a privacy storyboard. A privacy storyboard contains a set of short free-form textual descriptions (see examples in Fig. 7.2) organized in multiple data flows. These descriptions should contain all the necessary privacy-salient information (e.g., how the data is being collected, who it is being shared with, what information is being derived, how the data is being used, etc.) to evaluate the privacy problems while being understandable by a general audience after a quick read. The worksheet provides detailed steps to help a data practitioner break a data

process only involves crowd workers and practitioners, making it more agile than a professional privacy review.

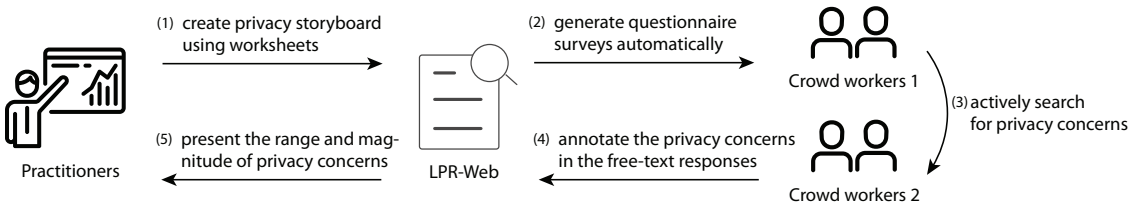


Figure 7.1: The workflow of a discounted privacy review. (1) Practitioners first transform an initial data practice design into a privacy storyboard (Fig. 7.2 left) using LPR's data action worksheet (Appendix §10.1), and (2) LPR web system (LPR-Web) would generate a set of questionnaire surveys (Fig. 7.2 middle) using a survey template. Practitioners then (3) send surveys to crowd workers and ask them to actively search for privacy concerns. (4) LPR then forwards the free-text responses to another group of crowd workers and asks them to label the privacy concerns using a taxonomy of privacy concerns (Fig. 7.2 right & Appendix §10.4). (5) Finally, LPR aggregates users' opinions and summarized the range and magnitude of privacy concerns in a compact form. In our evaluations, we found that practitioners can create the storyboard in 20-30 minutes, and the crowd review takes around 5.5 hours to complete (i.e., latency) and costs 3.7 hours of total crowd work ($\approx$ \$80).

practice down into smaller parts and generate the free-form textual descriptions. The web interface supports the data practitioner to organize these descriptions in data flows.

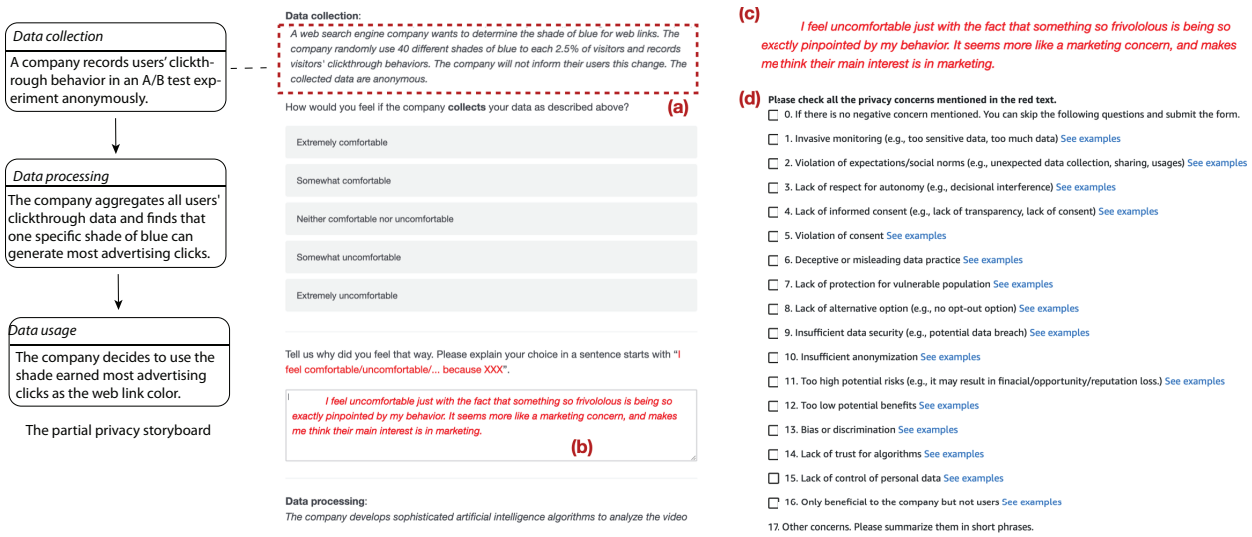
Collecting direct feedback from users (§7.5). LPR-Web then generates a set of questionnaire surveys automatically based on the privacy storyboard, and distributes them to crowd workers. Each survey asks a crowd worker to express her concerns regarding each textual description and elaborate on the reasons in free text (Fig. 7.2 middle). Each crowd task takes 5-15 minutes to complete. Similar to heuristic evaluation [270], a single individual might not find all the privacy concerns, but aggregating multiple crowd workers can improve the effectiveness since different people often discover different concerns.

Presenting feedback to practitioners (§7.6). Quantitatively interpreting users' free-text responses is challenging since users' concerns on privacy matters are often noisy and diverse [392]. LPR sends collected free-text responses (e.g., the red text in Fig. 7.2c) to another group of crowd workers for annotation of privacy concerns (Fig. 7.2 right). By doing so, LPR can aggregate free-text responses into various quantitative privacy spectra (see examples in Fig. 7.9, 7.10) to help practitioners understand the range and magnitude of users' privacy concerns.

7.1.2 Evaluation

We demonstrate through our evaluations that LPR is inexpensive, fast, consistent, and can help practitioners find potential concerns they are not aware of.

- **The applicability of privacy storyboarding** (§7.8). We selected 12 diverse real-world privacy stories



The partial privacy storyboard

Figure 7.2: An example use of LPR based on "Google's 41 Shades of Blue A/B test" [21, 162]. Suppose a practitioner wants to evaluate the privacy concerns of running an A/B test to determine which color is the best link color. She starts by creating a privacy storyboard (left). LPR then involves crowd workers in two different tasks: expressing privacy opinions (middle) and annotating free-text responses (right). LPR first asks crowd workers to examine a data action description (a) in free text (b), then forwards the collected free-text responses (c) to another group of workers for privacy concerns annotation (d).

and asked 3 participants to create privacy storyboards. Our results suggest that practitioners can generate such storyboards in a fast manner: each storyboard creation takes between 20-30 minutes.

- **The cost and latency of crowd inspection** (§7.9). LPR automatically generates surveys based on the user-generated storyboards and solicits crowd workers' opinions. We queried 20 crowd workers for each story. Our results suggest that the privacy concerns found by crowd workers converge. On average, aggregating the inspection results from any 14 crowd workers can uncover 97% of privacy concerns identified by all participants. The process costs 3.7 hours of total crowd work ($\approx$ \$80 in our experiments[2]), and the requester needs to wait for 5.5 hours[3] to get the results.
- **The quality of inspection results** (§7.9). Three trained privacy researchers compared the crowd worker results with the results from 24 software and data practitioners (a mix of trained UX researchers, privacy engineers and software engineers). They found that LPR discovered 89% of privacy concerns identified by data practitioners as well as 139% additional privacy concerns that practitioners did not identify, at a 6% false alarm rate.

7.1.3 Contributions

Our specific research contributions are as follows:

- LPR is the first method that inspects privacy concerns of data practices using the crowd. LPR takes a proposed data practice, breaks it down into smaller parts, generates a set of questionnaire surveys, solicits users' opinions, and summarizes the opinions in a compact form for practitioners to use.
- We propose a new storytelling technique, privacy storyboarding, to help practitioners communicate a data practice and relevant privacy considerations without having to implement a system. We identify a set of simple and expressive vocabulary for data actions (i.e., data collection, sharing, processing, and usage) and organize these actions in a tree-like topology (see §7.4.3 and examples in Fig. 7.4, 7.8, 10.1-10.12). The tree topology makes the storyboard modular, so practitioners can incrementally modify and append more data applications/actions.
- We introduced 12 real-world data uses cases into data ethics discussion and evaluated LPR using these practices with 240 crowd users and 24 data practitioners. Our results show that LPR is inexpensive, fast, consistent, and can provide high-quality privacy review results.

[2]The cost includes the payment to both the crowdsourcing platforms and crowd workers.

[3]The latency is because crowd workers do not start the task immediately after tasks are published, and some workers may abandon the tasks.

7.2 Related Work

LPR builds on ideas from four different areas: 1) privacy review by experts, 2) user research methods in understanding users' privacy concerns, 3) privacy surveys, and 4) crowdsourcing.

7.2.1 Privacy Review by Experts

Modern privacy regulations (e.g., [51]) argue that organizations should think about privacy in a proactive manner (i.e., Privacy by Design) rather than a reactive one (traditional privacy regulations). As such, one best practice before deploying a data practice is to have product teams consult internal privacy experts to vet problems beforehand [38], to make sure that the data practice is legal and poses an acceptable risk for the company's business [278]. These privacy experts are often a mix of experts from different relevant areas with specialized training in privacy, such as lawyers, UX researchers, privacy engineers, software engineers, and product designers [72], and might reside in dedicated stand-alone teams (e.g., a privacy council) or groups within the product team [266].

There exist a few frameworks for reviewing the privacy of a data practice, such as PRAM [277], Privacy Impact Assessment (PIA) [394], and Data Protection Impact Assessments (DPIA) [279]. These frameworks are derived from information security research and treat privacy problems in a manner similar to security risks. For example, PRAM first asks experts to map out the data processing pipeline within the target system, and then catalog contextual factors and data actions[4]. Experts then enumerate all potential problems associated with each data action and then assign scores to the likelihood and severity of each problem. In this way, builders can quantify and prioritize privacy risks for the organization (e.g., revenue loss from customer abandonment), and determine appropriate resource allocations to address the risks. The quality of such a privacy review is heavily dependent on the experts' expertise level.

Another major approach is guidelines-based, where privacy experts use a set of guidelines to design and analyze a specific data practice [169]. For example, the Fair Information Practice Principles (FIPPS) enumerates a set of guidelines — such as Transparency, Purpose Specification, Data Minimization — that an ideal privacy-sensitive system should satisfy. Data protection authorities can use the FIPPS to regulate practitioners regarding how, when, and for what purpose data can be collected, used, and disclosed [169]. Nissenbaum's "decision heuristic" [140] includes a series of guidelines articulated in nine steps, which can be used to evaluate the "system or practice in question."

However, both the privacy engineering and guidelines-based approaches look at privacy primarily from the organization's perspective (i.e., the risks for their business) rather than the end-users' view (i.e., end-users' perceptions and concerns). The likelihood and severity scores are also best guesses from these experts, which might be different from actual users' perceptions of the privacy problems. Through LPR, users'

[4]Data actions are information system operations that process personal information [277].

expectations of the transmission of their personal information —once carefully structured and aggregated —have the potential to serve as an important resource for *bottom-up* approaches to privacy decision-making.

Furthermore, these privacy frameworks require expertise in privacy, and privacy experts are scarce and expensive [72]. This constraint has a two-fold effect in practice. First, smaller teams often do not have the resources to hire such specialists. Second, even for companies with dedicated specialists, having a dedicated team reflect on each data practice's potential privacy concerns is infeasible. For example, companies like Facebook run over a thousand data science experiments each day [117, 197]; however, even a lightweight privacy discussion involving only a few data scientists, developers, and product managers can cost the company several thousand US dollars [252, 364]. LPR is designed to be complementary to these methods, when privacy experts are not available. Further, the fast and low-cost nature allows practitioners to explore more alternative data practice designs, and collect more direct feedback from users throughout the development life cycle.

7.2.2 User research methods in understanding users' privacy concerns

Beyond the privacy review from a company' s perspective, HCI & privacy researchers have developed various methods to understand users' privacy concerns, such as surveys [18, 109, 371, 392], focus groups [87, 204, 211], interviews [200, 259], experiments [108], contextual inquiry [28] and analyzing online corpus [259]. However, existing methods suffer from two constraints.

First, studying users' privacy concerns through these methods are often costly in terms of both time and money. For example, to investigate regrets associated with users' posts on Facebook, Wang et al. [371] had 569 users participate in interviews, user diaries, and online surveys. The entire process cost over $1000 in recruiting participants, and took the authors more than seven months to finish the studies. Naturally, most companies do not have the resources to justify such an investment and can not wait for that long period. Instead, LPR aims to offer a low-cost and fast method that can help data practitioners collect users' direct feedback.

Second, most UX studies (e.g., [371]) for data practices are conducted in a retrospective manner (i.e., when the target system is already deployed), since users often need to try the deployed system to understand their experiences. While techniques like Wizard of Oz [220] and similar alternative technologies [200] can help elicit users' generic privacy concerns, these privacy concerns often can only provide little direct feedback to tailored data practices. For example, Krombholz et al. [200] used Google Glass as an example of wearable technology to provoke participants' generic privacy concerns about wearables. However, users' privacy concerns regarding a data practice vary as the nuances of the data practice change [330, 392], and it is hard to emulate these nuances through an approximate set-up. In contrast, LPR develops a storytelling framework to help practitioners communicate their data practice without implementation. Further, LPR takes a procedural perspective, allowing users to dive into the procedure of the data practice to understand

the nuances and inform practitioners which part of the process might be problematic.

7.2.3 Privacy surveys

Privacy surveys are a common approach to measure users' privacy attitudes and concerns [32, 141, 141, 247, 360] across times [14, 203] and demographics [68, 74, 372]. One notable example is the over 30 privacy indexes created by Dr. Alan Westin [203]. These privacy indexes cover both the general level of privacy concerns of the public as well as the attitudes about specific privacy-related topics, for example, confidence in organizations that handle personal information, acceptance of a national identification system, and use of medical records for research.

A major limitation of these surveys is that participants' broad, generic privacy attitudes do not match well with context-specific, privacy-related behaviors, either actual or intended [69, 235]. Barkhuus [28] questioned the viability of obtaining universal answers in terms of people's "general" privacy practices, and argued for the use of more specific vocabularies to analyze contextually grounded privacy issues. A number of studies [18, 180, 387] have used the Contextual Integrity [272] framework to construct context-related questions that can help identify established privacy norms. For example, Apthorpe et al. [18] examined a range of settings, devices, and information types in specific contexts through questions like: "A sleep monitor records audio of its owner. How acceptable is it for the monitor to send this information to [different recipients]?" Contextual privacy surveys can provide valuable insights to establish privacy norms/guidelines. Still, they are often insufficient to provide direct feedback to a specific data practice, since these parameterized descriptions cannot capture the richness of complex data actions across multiple stakeholders.

The surveys generated by LPR fall into another group of privacy surveys, namely privacy incident surveys [73, 199, 392], which collect users' responses toward a specific scenario, aiming to capture the details of privacy-sensitive contexts. The most relevant work is ethical-response surveys [330], in which Schechter et al. [330] ran 3,539 factorial vignette surveys (FVS) to study five scenarios and found that a minor change to the Facebook emotional contagion experiment design can reduce users' disapproval and concern significantly.

However, exploring alternative benign variants is challenging for most practitioners. For example, to study users' privacy concerns through FVS, experimenters need to carefully design the controlled variables across variants, collect a large number of responses, and analyze the results to illustrate statistical significance. Therefore, FVS is too expensive, in terms of time, cost, and expertise requirement, when data practitioners need a tool to support their decision making in the day-to-day work. Our key insight into the process is that non-tech-savvy participants can contribute more than just answering multiple-choice questions. Instead, LPR asks participants to **actively** examine a specific data practice for privacy concerns and express their concerns in free-text. This design reduces the cost of privacy surveys for two reasons. First,

by asking participants to actively search for privacy concerns, experimenters do not need to control variable across surveys. Second, the free-text responses are more effective in collecting participants' privacy-related opinions than the Likert scores. A few free-text responses can cover multiple privacy concerns, while it may take hundreds of numerical scores to illustrate the statistical significance.

7.2.4 Crowdsourcing

Crowdsourcing is a widely used method for many tasks, such as gathering data to train algorithms [88], running user studies [194], proofreading and text editing [31], real-time captioning [208], writing news articles [195], creating taxonomies [60], penetration tests [100], exploring security configurations [175] and usability testing [230]. To the best of our knowledge, LPR is the first work to measure privacy concerns through crowdsourcing.

Crowdsourcing brings several unique benefits for the privacy review tasks, such as easy-to-access participants pool, low cost, fast response, and programmable demographics [230]. However, crowd workers may lack the motivation and expertise to engage in complex tasks and provide high-quality feedback [194].

We applied the insights from the prior crowdsourcing research to address these challenges. LPR offers a specific workflow, breaking a single privacy review task into two small types of tasks: expressing free-text privacy opinions and annotating free-text responses using a taxonomy of privacy concerns. We were inspired in part by the Find-Fix-Verify pattern in Soylent [31], which helps control the quality of crowdsourcing results. Kittur et al. [194] recommend making crowdsourced tasks more laborious to make it hard to cheat, so LPR asks participants to respond with more than 100 characters. MicroTalk [102] finds that argumentation, which requires workers to justify their responses, can improve results accuracy by 20%. Similarly, in addition to rate their comfortable level, LPR asks participants to elaborate on the reasons behind their rating.

7.3 Design goals, challenges and development method

In this section, we discuss the design goal, challenges and brief the development process of LPR.

7.3.1 LPR Design Goals

To recap, existing solutions to review privacy design issues suffer from a number of limitations, such as the lack of feedback from potential users [277, 394], high cost in terms of both time and resources [12, 38, 277, 330, 394], retrospective nature that are only useful once a system is already deployed [394], a slow review process (i.e., feedback latency) [330], and the ability to only offer generic feedback [18, 277]. In response to these limitations, we define three properties an ideal lean privacy review solution should embody.

- **Human-centered privacy measurement.** The technique should collect users' privacy opinions directly and use these opinions to help practitioners identify privacy design flaws.

- **Low cost, light-weight, and accessible.** Existing review techniques (e.g., PRAM, FRS) are expensive and time-consuming. McQuinn et al. [249] estimate that a privacy audit can cost $10,000 to $60,000. The audit process may take multiple weeks since it often involves multiple stakeholders (domain experts, UX researchers, privacy engineers, and executives) [38]. An ideal method should be fast, accessible, and low cost, so practitioners can collect users' direct feedback quickly and iterate the data practice design accordingly.
- **Practical feedback for practitioners.** The technique should provide concrete and quantitative feedback to data practitioners, not only which part of the data practice might be wrong, but also how severe the problem is (i.e., the percentage of affected people and the severity of the impact). In this way, practitioners can test different variations to balance privacy-relevant design choices with opposing interests.

7.3.2 LPR challenges

At the heart of LPR is a novel approach that guides non-specialists in **actively** examining a specific data practice for privacy concerns, which reduces the cost and required resources for privacy concern inspection. We develop and test LPR with crowd workers since they are easy to access for most practitioners. Besides, most crowdsourcing platforms allow requesters to specify the eligibility criteria so that practitioners can approximate the demographics of participants to match the target users group[5]. Along with these exciting benefits also come three significant challenges.

- **Communicating the privacy design to crowd workers**. Understanding the details of a data practice requires significant expertise and technical background.
- **Scaffolding the privacy design review**. Asking crowd workers to review a privacy design is challenging for two reasons: task duration and expertise. When a privacy engineer reviews the data practice, she needs to enumerate the potential privacy problems and then to assess the impact and likelihood. This process can take hours to complete; however, most crowd workers can hardly engage for such a long time. Besides, crowd workers may only have a vague understanding of privacy and may be unable to articulate the problem. For example, feedback such as "I just do not like this data practice" offers rather limited insights for practitioners.
- **Presenting the output to practitioners**. Interpreting results from crowd workers is not easy. First, practitioners may not have the time to read through all the raw feedback from crowd workers. Second, users' responses to privacy designs can be diverse and conflict with one another. As such, quantitatively assessing the results can be challenging.

[5]LPR should perform the best when the inspection participants represent the target users group. However, crowd workers may not resemble the target user population. We discuss this limitation in §7.11.2.

7.3.3 A Summary of the Development Process

Over a span of two years, we designed, implemented and evaluated the solutions for the above challenges iteratively. We started with a formative study (§7.4.1) to explore the design space of privacy storytelling, and used the obtained insights (§7.4.2) to devise an initial privacy storyboarding technique. We then coded real-world data practices using the proposed storytelling technique and iterated on the storyboard representation as we expanded the supported use cases (100+). In the end, we identified a set of simple and expressive vocabulary for privacy storyboarding (§7.4.3) and developed a worksheet (Appendix §10.1) to facilitate data action analysis (§7.4.4).

We then iterated the design of privacy inspection tasks for non-specialist crowd workers in 7 rounds of experimentation, aiming to communicate a privacy storyboard through a reusable template and collect participants' responses in an aggregatable manner. In each round, we experimented with different task designs and sent at least 50 surveys (5 storyboards x 10 participants) to crowd workers. The authors then manually went through the responses, analyzed the inspection quality, potential confusion, as well as the quantitative metrics (e.g., task completion rate, duration), and used these feedback to inform the task design in the next round. To avoid the learning effect, all the crowd workers can only participate in the study once. The required completion times for different tasks vary; we paid participants an average hourly rate of $15, by offering an extra bonus after survey completion.

Our initial intuition was to apply the design of usability heuristic evaluation to privacy problems (§7.5.1). We derived heuristics from past privacy research literature, and asked crowd workers to inspect the privacy problems of each data action using offered heuristics. However, the heuristic-based approach suffered from a few importation limitations. We focused on the iteration of heuristics in the first four rounds and then shifted to a two-stage design (collecting free-text responses (§7.5) first and then asking crowd workers to annotate (§7.6)). Through the process, we refined the storytelling techniques to communicate the details better, formulated the privacy problems as privacy concerns, articulated the design of the crowd privacy inspection tasks.

7.4 Communicating the Privacy Design: Privacy Storyboarding

Communicating a data practice between non-specialists and data practitioners is challenging. Existing solutions tend to choose between either being easy-to-understand for non-specialists (e.g., UX storyboards) or easy-to-generate for data practitioners (e.g., context data flow diagram [233, 337]). This section describes the design and development of the privacy storyboarding technique we created for LPR. Here, our goal is to provide a set of simple and expressive vocabularies to help practitioners effectively communicate their data practices to non-specialists without having to fully implement a working system.

7.4.1 Formative Study

To explore the design space of privacy storytelling, we first conducted semi-structured interviews with six participants (three male and three female, mean age = 25.3, max=30, min=21) to understand the communication challenges as well as informing the initial design of LPR. We recruited these participants through the flyers posted on a university campus. Two of them were professionals who live around the campus, and the rest were students. None of the participants had received specialized privacy training previously.

We chose six privacy incidents[6] (see Table 8.7) and organized the privacy scenarios similar to Woodruff et al. [392]. Namely, we identified the context, procedure, and the privacy-relevant outcomes for each privacy scenarios explicitly. We then tested each scenario with three storytelling techniques from the literature.

1. **A long textual paragraph** [330, 336] (Fig. 7.3 left). Scenario-based text descriptions/fiction/ narratives have been commonly used in many privacy-related studies (e.g., [330, 336, 392]). We chose this approach as the baseline.
2. **A hand-crafted privacy storyboard** (Fig. 7.3 right). Using a graphical representation to describe a privacy practice is inspired by scenario-based storyboarding in UX design [319, 359]. Designers often create storyboards to help users perceive, interpret, and make sense of proposed functionalities. Graphical representations are more engaging and less biased while the use of words may bias a user's reaction to particular technologies [358].
3. **Multiple short story snippets** [329]. The design of multiple short snippets is inspired by the Survey on the Ethics of Scientific Experimentation [329], in which Schechter et al. listed the procedures of a data practice in bullet points. Such a design has two advantages. First, users' privacy opinions are often beyond the utilitarian perspective. Even if the outcome of the data practice is highly beneficial, other procedural factors, such as deception, even with goodwill, may irritate users. Second, we want to help practitioners understand which step in their data practice is problematic and why, i.e., locate the problem in a specific step of the data practice.

For (1) and (2), we first presented the whole scenario, and then asked the participants: "If someone you cared about were a candidate participant for this data practice, would you want that person to be included as a participant?" [7] For (3), we presented each snippet one by one and asked the same question separately. Participants were also asked to explain their rationale for each question. To trigger participants' reflection and test their comprehension, we also presented a few variations for each scenario. For example, in the Target Pregnancy Prediction [158], we asked participants, "what if the retail company wants to predict if users are sports fans and send beer coupons to them during the World Cup?"

The experimental design is similar to Jin et al. [182]. Each participant went through all six privacy

[6] We discuss the scenario selection criteria in §7.8.1.

[7] This question is adapted from Schechter and Bravo-Lillo [330].

incidents in a randomized order, and we randomly assigned one of the three storytelling techniques to each incident. So we tested each storytelling method in 12 tasks (6 incidents x 2 participants). Each interview lasted around one hour. We compensated each participant $20 for their time. None of the participants were aware of these incidents before the study.

Table 7.1: Privacy scenarios used in the formative study.

#	Original source	Abstract
1	Target Pregnancy Prediction (Fig. 7.3) [158]	A retail company develops a customer tracking technology that can predict if a female customer is likely pregnant.
2	Facebook Emotion Contagion study [199]	A social network company conducts a massive psychological experiment, manipulating their news feeds to assess the effects on their emotions.
3	Google 41 Shades of Blue [21, 162]	A search engine company runs a large scale A/B test experiment, randomly showing 41 different shades of blue to visitors to determine the best color of links.
4	Cambridge Analytica Scandal [142, 365]	A political consulting firm harvests the personal data of millions of people' s social network profiles without their consent and used it for political advertising purpose.
5	OkCupid match score manipulation [159, 391]	A mobile dating app manipulates customer data to see how users of its dating service would react to one another.
6	Device-based price discrimination [8]	An online travel agency shows different prices to users with different devices (e.g., mobile v.s. PC, Android v.s. iOS).

7.4.2 Findings and Design choices

This pilot study resulted in the following findings.

Graphical vs. Textual representation. Graphical representations for privacy storytelling suffer from several limitations. First, generating a visual storyboard is challenging. The authors need to have a thorough understanding of the data lifecycle, decent visual design skills, and familiarity with UX Storyboarding Guidelines (e.g., number of frames, level of detail, the inclusion of text [359]). Second, manipulating the variables in visual storyboards is not easy. Visual storyboards are often developed to test distinct features. In contrast, privacy practices are often subtle and need multiple minor variations to probe the salient issues. For example, Schechter et al. [330] created ten variants for the Facebook Emotion Contagion study [199], each describing a small experiment change.

A social media company collected information about its users, including the posts its users shared online.

Using automated algorithmic model, it can measure whether users' posts were of a positive or negative mood.

Participants will not be identified and will remain anonymous.

In order to test whether the mood of a user's friends' posts will influence this user's mood, the company either randomly excluded some fraction of friends' negative posts [or positive posts] each time the news feed was loaded.

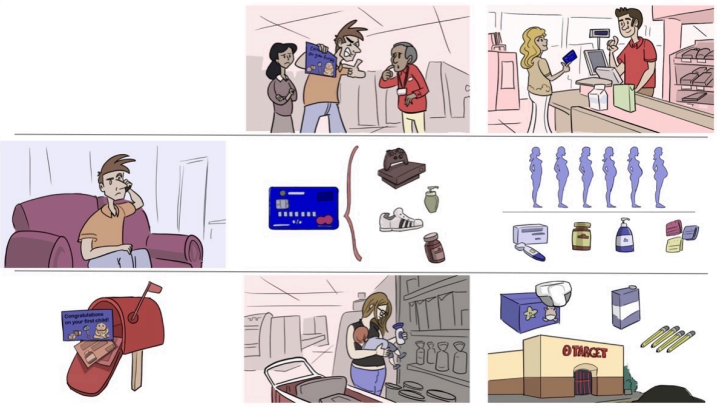


Figure 7.3: Story telling techniques used in the formative study. Left: the plain text description of "#2 Facebook Emotion Contagion Study" [199]. Right: the storyboard representation of "#1 Target Pregnancy Prediction" [158].

Resulting design choice: We decided to use textual representation since it is easy to generate and iterate. We discuss the potential bias caused by the use of words in §7.12.3 .

Multiple short snippets vs. a long paragraph. Short snippets can elicit more explicit privacy opinions from participants. When we asked participants to think about each step independently, they were able to articulate their rationale. However, for scenarios presented in long paragraphs, participants can express the overall comfortableness but struggle to articulate specific reasons.

Resulting design choice: We decided to break each scenario into multiple snippets since it allows participants to dive into the procedure of data practice and tell practitioners which part of the process might be problematic.

Risk-benefit analysis. Participants had difficulty in balancing the risks and benefits to make a confident decision/judgment. Throughout the interview, we asked follow-up questions to test participants' reactions to a few data practice variations. For example, besides personalizing the advertisements, the retail company can also use users' purchase behavior to optimize their supply chains to reduce the operation cost. We then asked participants whether knowing these benign data usages change their prior answers. In this process, most participants cannot consistently evaluate the potential benefits and harm, and elaborate their rationals as clear as the prior answer.

Resulting design choice: We decided to divide the risk-benefit analysis task into a set of single outcome evaluation tasks, where each participant only evaluates the impact of one outcome (i.e., one root-to-leaf branch in Fig. 7.4) at a time.

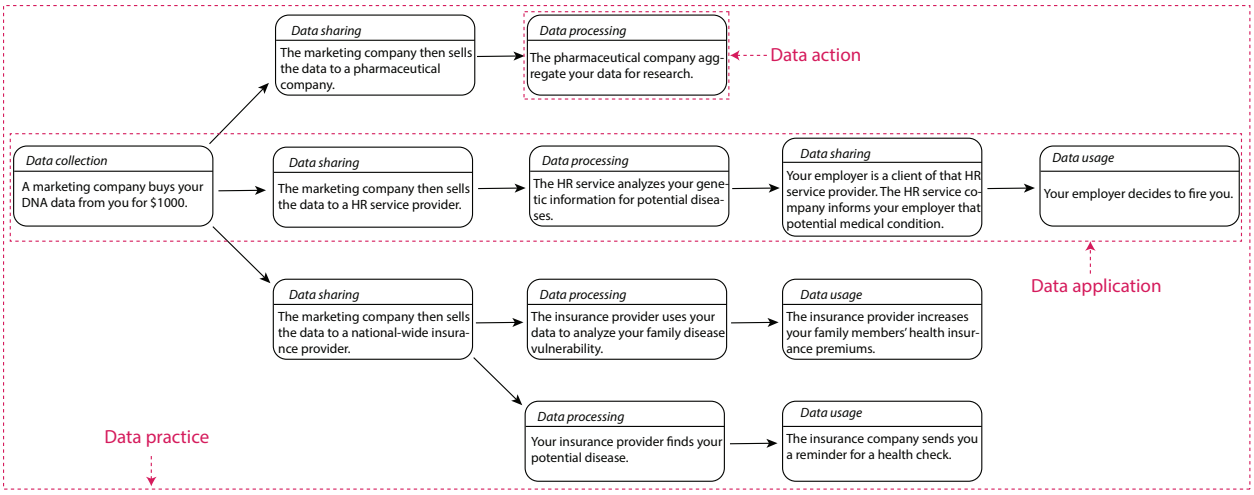


Figure 7.4: A privacy storyboard of "trading your DNA data for $1000" [392]. Practitioners can transform an initial data practice design into a privacy storyboard using LPR ' s data action worksheet. We annotate the examples of "data action", "data application" and "data practice" in red boxes. In the storyboard, each node is associated with a data action. This tree topology explains the data flow using four data action vocabularies: collection, sharing, processing, and usage. This modular design allows practitioners to modify and append more data applications/actions incrementally.

7.4.3 LPR Story Representation

These findings motivated us to consider a new storytelling technique that describes a data practice from a data-centric perspective. An ideal privacy storytelling technique should be easy to understand for non-specialists and easy to generate for data practitioners.

Term definitions. We begin by defining four key concepts (Fig. 7.4): data action, data application, data practice, and data stakeholders. We derive these terms from the past privacy discussions [164, 277, 339].

A *data action* is the smallest unit in the LPR story, which describes a specific operation where consumer businesses interact with users' data[8].

A *data application* is a chain of data actions, describing the complete data lifecycle resulting in a specific consequence. For example, Google's 41 shades of blue data science experiment can be described as a chain of three data actions (Fig. 7.2 left): a company first collects clickthrough data in an A/B test environment, then aggregates clickthrough data to find the one specific shade that generates most clicks, and finally decides to change the link color accordingly.

A *data practice* is a set of relevant data applications. For example, the clickthrough data can also be used for other data applications, such as search result evaluation [146] and advertisement billing [317].

[8]We adapted the definition of "data action" from [277].

A *data stakeholder* is an individual or group that could affect or be affected by the data practice [353]. We articulate three types of data stakeholders: data subjects, data observers, and data beneficiaries/victims. Data subjects are the people who contribute their data [164]. Data observers are the entities (e.g., people, algorithms, companies) that have access to users' data [164]. Data beneficiaries/victims are the people impacted by a data practice.

Tree[9] topology. A common way for practitioners to present the information flow within a system is with data flow diagrams [233, 337], which visualizes the data communicated between different system components and external entities. For example, a data flow diagram of an online bookstore may describe how the shopping cart module sends the transaction data to the database. Such a system-centric graph can help the developers understand the system requirements, but do not necessarily explain data flow from the users' perspective.

In response to these limitations, we change the depiction perspective to a user-centric model, which describes how data flows between stakeholders and how data impacts different stakeholders. In doing so, we propose new data flow vocabularies to help practitioners communicate complex real-world privacy matters. While each node in a traditional data flow diagram is often a system component, the basic vocabularies in LPR are individual data actions, each documenting the privacy-relevant information, such as what data is being collected, where it is being sent to, and why. Such a perspective shift allows non-specialists to track the accountability of different stakeholders explicitly.

A privacy storyboard then organizes these data actions in a tree topology. Figure 7.4 illustrates an example of an LPR story, which describes the data practice "trading your DNA data for $1000 "from [392]. Each node in the tree topology is associated with a data action, and the edges connecting nodes represent the data flow between different data actions. Each root-to-leaf chain describes the data lifecycle of a specific data application. Such a tree topology makes the storyboard modular, so practitioners can incrementally modify and append more data applications/actions to the data practice.

Data action primitives. Through iterative development, we noticed that the data actions are clusterable. Our catalog of data actions is inspired by Solove's taxonomy [339] of harmful privacy activities, where Solove enumerates 16 activities that invade privacy and organizes them into four sequential groups: (1) information collection, (2) information processing, (3) information dissemination, and (4) invasion. While Solove originally uses these groups to organize harmful activities, we find that these groups are succinct and expressive vocabularies to describe the data flow.

We first extracted 67 data applications from 14 privacy scenarios[10] in Woodruff et al. [392] as the test cases. We then coded these applications into tree typologies and iteratively refined the definition of data

[9]The tree-like topology refers to the term in computer networks. Each node in a tree has zero or more child nodes but at most one parent node.

[10]The appendix of [392] contains 20 scenarios. We excluded the non-consumer business ones.

action primitives. We eventually concluded with four types of data action primitives (Table 7.2): 1) Data collection, 2) Data sharing, 3) Data processing, and 4) Data usage. We also made two main changes to make these vocabularies expressive in describing a data practice.

- **Data actions are neutral.** The original terms from Solove's taxonomy refer to problematic privacy activities. In contrast, we design LPR to evaluate an unknown data practice (malicious or benign), so we make data action primitives neutral and extend the coverage. For example, we change "invasion" to "data usage," which refers to the potentially neutral application a business can perform through the use of users' data that will impact the users in a certain way.
- **The order of data actions is flexible.** While Solove's order[11] describes the most common sequence of privacy invasions, we made it flexible to accommodate various complex real-world privacy-related data practices (see examples in Appendix Fig. 10.1-10.12). For example, an HR service provider may purchase the data from a marketing company and then analyze the data (Fig. 7.4).

Table 7.2: A data action is the smallest unit in LPR story, which describes a specific operation that consumer businesses interact with users' data. We have four types of data action primitives: 1) Data collection, 2) Data sharing, 3) Data processing and 4) Data usage.

Primitives	Definitions	Examples
Data collection	A data observer collects/stores data from data subjects.	Collection, storage
Data processing	A data observer processes users' data to derive new data.	Re-identifying, anonymization, inference, aggregation
Data sharing	A data observer shares user's data or derived data with another data stakeholder (i.e., observer, subject and beneficiary/victim).	Sharing, dissemination, transfer, increasing data accessibility
Data usage	A data observer uses the data in a certain way that impacts a data beneficiary/victim..	Decision interference, knowledge discovery

7.4.4 Generating an LPR Story

Through an iterative coding process, we developed a method to help data practitioners think through a data practice and generate an LPR story. Instruction worksheets are presented in Appendix §10.1. Here, we describe the three major steps and explain the design rationale.

(1) Identify data applications

- *Who are the data subjects? Who are the data observers? Who are the beneficiaries and victims?*
- *How would the data practice impact the stakeholders directly or indirectly?*

[11] collection->processing->dissemination->invasion

(2) Break each data application into data actions

- *Each data action should only contain at most one stakeholder per type (i.e., data subjects, data observers (senders or receivers), beneficiaries/victims).*

(3) Describe each data action in succinct text

- *What is the context of the data action?*

Below, we examine each of these steps in detail, describing what kinds of information the question is looking for, why it is important, and offering some examples.

Identifying data applications

The first step for data practitioners is to enumerate the potential outcomes and applications that will be generated by the data practice in question. Data collected to achieve beneficial objects may adversely affect individuals' privacy as an unintended consequence [42]. For example, search engines use users' locations to make search results more relevant; however, the data collected might also be used to improve targeted advertising. Our goal here is to separate the duties [380] of different data applications by disseminating the tasks and associated privileges.

It would be challenging to enumerate every possible outcome; the focus here should be on most possible and common cases (both positive and negative). As such, we adapted two sets of questions from the privacy risk model by Hong et al. [164]. The first set asks data practitioners to think about the stakeholders impacted by the data practice: *Who are the data subjects? Who are the data observers? Who are the beneficiaries and victims?* The second set asks the author to think about the likely outcomes for each data application: *How would the data practice impact the stakeholders directly or indirectly?*

Breaking each data application into data actions.

The second step is to segment data applications into data actions and then organize them in a chain format. Our earlier pilot study (§7.4.1) suggests that the breakdown can both help participants more effectively understand the data practice and help practitioners to pinpoint specific problematic data action. The challenge here is to determine the segmentation strategy. An ideal strategy should produce relatively consistent segmentation results, which are easy to understand for non-specialists, while also be easy to use for practitioners.

The design of our segmentation strategy is inspired by the literature in UX storyboarding [359] and comic creation [245]. For example, Truong et al. [359] suggest that the optimal length of a storyboard is between three and five. Both undersegmentation and oversegmentation will decrease user understanding and engagement. We empirically tested different segmentation strategies suggested by McCloud [245], such as

subject-to-subject, scene-to-scene, action-to-action, moment-to-moment, iteratively on 94 data applications (67 from [392] and 27 from Table 8.7).

We finally concluded with the following segmentation strategy: each data action should only contain at most one stakeholder per type (i.e., data subjects, data observers, beneficiaries/victims). For example, if a data action involves two beneficiaries, this data action should be further divided into two data actions. In doing so, the crowd workers can express their opinions to stakeholders separately. Meanwhile, the story authors can more easily trace accountability by tracking the data flow. Take the Target pregnancy prediction scenario [158] as an example. We can divide the data application into three data actions: (1) a retail company collects users' purchase data from their customers; (2) the retail company processes users' purchase history to predict if a female customer is likely to be pregnant; (3) the retail company sends the tailored coupons to these mothers-to-be.

Note the final segmentation output still depends on the granularity of stakeholders. For example, practitioners may treat the whole retail company as a stakeholder so that they can test users' general perception to different prediction types (e.g., first-time parents, sports fans). Alternatively, they can also treat each department (e.g., data science department, marketing department) as an independent stakeholder so that they can trace accountability.

Once segmented, data applications may share common data actions. For example, the data collection node is often the root node of a tree, deriving multiple data usages. Merging the shared data actions results in a tree topology.

Describe each data action in succinct text.

The last step is to generate a succinct text description for each data action, explaining the privacy-salient information to a non-tech-savvy audience (e.g., crowd workers). There exist several description templates in different privacy frameworks, e.g., the 5-parameter information flow template [18, 272] from the Contextual Integrity framework, and the examples in the PIA template [90]. We applied these templates to over 100 data actions and found two challenges in using them directly in the context of LPR.

- **Existing templates primarily focus on the data collection part but barely apply to other data actions.** Taking the Target pregnancy prediction scenario [158] as an example again, the 5-parameter information flow can apply to the first data action (i.e., data collection): a retail company (recipient) collects their customers' purchase data (attribute) if they (sender) make purchase in online/offline stores (transmission principle). However, this template barely applies to the other two data actions (i.e., processing and usages). Data processing actions infer some new data insights from the raw data, data usage actions involves data beneficiary/victim. The 5-parameter template does not cover both new data insights and beneficiaries/victims.
- **Existing templates are often too compact to include important contextual information.** In the Tar-

Table 7.3: We can describe the primary data application derived from "How Target Figured Out A Teen Girl Was Pregnant Before Her Father Did" [157] as a chain of three data actions (collection - processing - usage). The following table illustrates the example output by authors using the worksheet.

Data collection	When a user made purchases at a retail store (online/offline), the retailing company collected various behavior data, such as the purchase items, payment credit card, email address, delivery address, etc.
Data processing	Based on the email, delivery address, credit card number, and so on, the company associates the purchase history with anonymized identity. The retail company later develops various models to predict user traits using the purchase history. For example, pregnant women purchase different items such as supplements like calcium, magnesium, and zinc. So the model guesses the customer may be pregnant if she buys these products in a short period.
Data usage	Once the retail company predicts the customers might be pregnant, the company will send out tailored coupons to these parents-to-be, e.g., special coupons for diapers, baby clothes catalogs. Sending coupons for baby items does not bring too much profit. The underlying goal is that the company want to re-shape their shopping habits. Research shows that first-time parents are experiencing the most hectic time in their life, and their shopping habits may experience a major change during that period. The retailing company wants to leverage this special opportunity.

get pregnancy prediction scenario, the underlying goal of sending first-time parents special coupons is that the company wants to re-shape their life-long shopping habits. Research shows that first-time parents are experiencing the most hectic time in their life, and their shopping habits may experience a major change during that period. It is hard to fit this kind of contextual information into existing compact templates.

We then iterated text descriptions for the 94 data applications, tested them with crowd workers, and concluded with a lightweight description structure: [Context] [Data action][Further description]. Each data action description starts with an introduction of the context, then presents the specific data action, and concludes with a few necessary further descriptions. Reviewing the final descriptions, we propose a set of questions (attached in Appendix §10.1) intended to help practitioners think through each data action and generate the descriptions. Table 7.3 enumerates some example output after this step.

7.5 Scaffolding the privacy design review

This section describes the design and development of a non-specialist-oriented privacy inspection, which collects users' qualitative (i.e., free-text responses) and quantitative (i.e., numerical comfortableness scores) feedback regarding their privacy concerns towards each data action (see Fig. 7.2 middle).

7.5.1 A Failed Attempt: Heuristics for Crowd Privacy Inspection

In our initial task iterations (see §7.3.3), we attempted to develop a set of heuristics to capture common privacy concerns. Similar to UX heuristic evaluation [270], we presented participants with a set of heuristics and asked them to inspect each data action using these heuristics. In doing so, we hoped to be able to associate open-ended responses with different heuristics and present an aggregatable view to data practitioners.

We experimented three sets of heuristics for privacy problems (Table 7.5) derived from a range of legal and policy documents [1, 89, 123] and tested each set with crowd workers. For each set, we tested at least five different privacy scenarios with ten crowd workers. Figure 7.5 illustrates one set of our heuristic questions. Participants were asked to go through the heuristics one by one for each data action, evaluate whether the data action violates them, and explain the reasons. In each round, we analyzed the collected responses to inform the task design (i.e., question templates and heuristics) in the next round. We observed a few trade-offs through the process.

Table 7.4: Sources we used to generate heuristics for privacy problems.

#	Source	Heuristics
1	Seven types of privacy [123]	Privacy of the Person, Privacy of Personal Behaviour and Action, Privacy of Communications, Privacy of Data and Image, Privacy of thoughts and feelings, Privacy of location and space, Privacy of association
2	Privacy rights by Roger Clarke [1]	Privacy of the Person, Privacy of Personal Behaviour, Privacy of Personal Communications, Privacy of Personal Data
3	Belmont Report [89] (Fig. 7.5)	Respect for persons, Beneficence, Justice

+ The heuristics can increase crowd workers' sensitivity to privacy problems. With heuristics, participants become more critical and verbal about their privacy concerns, tending to mention one or multiple privacy problems for each heuristic.
+ We can use heuristics to aggregate users' open-ended responses.
- Participants used the heuristics as the criteria rather than their actual feelings. For example, one participant reported that "sending personalized coupons" violates "respect for persons", but also

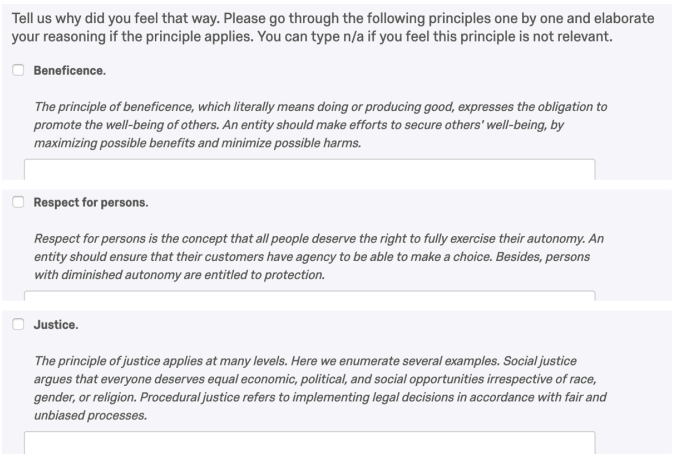


Figure 7.5: One survey screenshot of the failed attempts using heuristic guidelines. Participants were asked to go through the heuristic one by one for each data action, evaluate whether the data action violates it, and explain the rationale. The example heuristic guidelines are derived from Belmont Report [89]. While heuristics makes crowd workers more sensitive to privacy concerns, they also introduced two tradeoffs: (1) participants become overly critical; (2) participants either have a shallow understanding of the heuristics or spend significant time to learn the heuristics.

noted that he is "fine with it since it may save him some money."

- Participants tend to check all the minor privacy problems and have a hard time prioritizing them. As a result, the collective output of the heuristic-based approach quantifies both how visible the problem is, and how severe the problem is.
- Learning the heuristics during a short survey was challenging. Most crowd workers are not aware of the principles before joining the tasks. They can only gain a limited understanding during the study instruction and warm-up tasks.
- Going through the heuristics one by one was time-consuming. Inspecting four sequential data actions with seven heuristics often takes more than 1 hour. Meanwhile, the response quality from crowd workers is often reduced as the completion time increases. In the first four iterations, we aggressively reduced the number of heuristics to speed up the task. However, the last heuristic-based approach using three heuristics still takes around 45 minutes to complete a survey.

These findings motivated us to rethink the design of the privacy review task and the necessity of heuristics. Privacy review by crowd workers is different from heuristic evaluation in several respects. First, heuristic evaluation is mainly conducted by trained UX researchers since it takes time and practice to gain a deep understanding of usability heuristics. Second, privacy opinions can be more personal and abstract than usability problems and therefore can be difficult for users to pinpoint what's bothering them and map it back to heuristics.

In the later iterations, we shifted the design of privacy review tasks to a two-stage design, which breaks the task into two sub-tasks: inspection (§7.5) and annotation (§7.6). We first asked participants to provide free-text responses without heuristics, and then forwarded these responses to a different set of crowd workers to annotate privacy concerns. This inspection-annotation paradigm has several advantages over the heuristic-based approach. First, users tend to describe the privacy concerns they care about most in free text, which helps us prioritize their privacy concerns. Second, users don't need to learn and reflect on the heuristics, so it reduces the heuristic overhead and speeds up task completion. Meanwhile, we can aggregate users' open-ended responses using the labels from the annotation tasks.

In the following subsections, we introduce three important features of LPR, which further scaffold the privacy inspection task for crowd workers: (1) Eliciting users' feedback through privacy concerns (§7.5.2); (2) Searching privacy problems collectively and actively to increase the coverage (§7.5.3); and (3) Dividing the survey into mini-surveys to lower the task barrier (§7.5.4). We present the design of the annotation tasks in §7.6.

7.5.2 Eliciting Users' Feedback through Privacy Concerns

These early explorations also raised an important question: what do we aim to measure through LPR (e.g., privacy problems, privacy concerns, privacy risks), and how do we guide participants to inspect the target properties?

Indeed, privacy is a concept in disarray [339]. Researchers and practitioners have developed many terms to capture different dimensions of this complicated concept. For example, Solove [339] articulates the term "harmful privacy activities" by enumerating 16 activities that invade people's privacy. Privacy engineering techniques (e.g., PIA [394], PRAM [277]), on the other hand, use the term "privacy risk," which is the multiplication of the likelihood and the impact of privacy harms. We surveyed relevant privacy concepts in the literature (Table 7.5), tested multiple privacy concepts in experimental surveys, analyzed the responses from crowd workers.

We chose to focus on the concept of privacy concerns. The term "privacy concern" has been heavily used by privacy researchers in the HCI community [121, 203, 295, 334], which refers to users' subjective feeling (e.g., whether users are comfortable) towards a specific privacy-relevant problem. Different from privacy risk, privacy concern is not necessarily associated with personally identifiable information collection or potential privacy harm. For example, users might feel uncomfortable because of an unfair procedure, a digital market manipulation [45], or simply a lack of trust.

We decided to focus on privacy concerns for two reasons. First, users' subjective feelings towards a data practice are important, yet there exists few systematic methods to capture them directly at a low cost. Second, both our formative studies and early privacy surveys also found that users have a desire to express their privacy opinions. It is easier to ask users to reflect on their feelings rather than evaluate the data practice

using other system metrics (e.g., privacy risks).

Table 7.5: A list of commonly used privacy concepts. We articulate these privacy concepts to illustrate the conceptual differences.

#	Privacy concept	Description	References
1	Privacy harm/injury/ activity	the negative impact of a data action, which can be physical, mental, or economic injury to the consumers	[67, 277, 339]
2	Privacy risk	the likelihood of a privacy harm multiply the impact of the privacy harm	[164, 278]
3	Perceived privacy risk	users' perceived likelihood of a privacy harm multiply users' perceived impact of the privacy harm	[32, 127, 261]
4	Privacy concern	a combination of perceived privacy risk/benefit trade-off, procedural fairness, trust and feelings	[121, 203, 334]
5	Privacy norm/ expectation	established based on the collective pattern of privacy concerns	[18]
6	Privacy preference	established based on an individual's consistent privacy concerns on similar data practices	[226]
7	Privacy violation	a problematic data action that would violate privacy norms	[298]

7.5.3 Searching Privacy Problems Collectively and Actively

Existing research often formulate privacy concerns as numerical values and measure them using vignette factorial surveys [121, 330]. For example, Schechter et. al [330] uses the question *"If someone you cared about were a candidate participant for this experiment, would you want that person to be included as a participant?"* to collect respondents' responses. Respondents can answer in three options: "Yes," "I have no preference," or "No." To measure users' concerns from these responses, researchers need to survey a large number of users [330], carefully control the variables across different surveys, and quantify users' concerns through statistical analysis. This process is often expensive and time-consuming. Meanwhile, practitioners may not have the necessary skills to run such a rigorous study.

In contrast, LPR contributes a different and complementary method. Our method is inspired by heuristic evaluation [256, 270] - we make an analogy between usability problems and privacy problems. In traditional user testing (Fig. 7.7), the observer (i.e., experimenter) has the responsibility of interpreting users' actions to infer how these actions are related to the usability issues. Heuristic evaluation reduces the cost of finding usability problems by asking evaluators to actively assess the usability of a user interface using a set of heuristics.

Table 7.6: Comparison between LPR and existing techniques: privacy engineering techniques (e.g., PRAM [394], PIA [394]) and factorial vignette surveys (FVS). Methods like PRAM, PIA require privacy engineers to discover and quantify potential privacy problems. FVS requires researchers to enumerate privacy problems and quantify the issues by analyzing laypersons' responses. In LPR, laypersons both discover and quantify privacy problems.

Techniques	Who discovers privacy problems?			Who quantifies privacy problems?		
	Experts	Practitioners	Laypersons	Experts	Practitioners	Laypersons
PRAM [277]	✓	✓		✓		
FVS [330]	✓			✓		✓
LPR			✓			✓

Traditional user testing	The experimenter interprets users' action to infer how these actions are related to the usability issues.
Heuristic evaluation	The evaluators go through the interface from users' perspective, assess the usability and report their comments.

Table 7.7: Usability evaluation [268]

Privacy factorial surveys	The experimenter designs surveys consisting of varying situations and interprets the respondents judgments.
LPR	Crowd workers go through a data practice from users' perspective, assess the privacy concerns and report their comments.

Table 7.8: Privacy concern inspection

The conventional approach of using factorial vignette surveys (Fig. 7.8) operates similarly to traditional user testing, which requires the experimenter to interpret respondents' numerical answers. Instead, LPR asks users to nominate their privacy concerns actively, which is similar in spirit to heuristic evaluation. In doing so, LPR formulates privacy concerns as *the reasons why users feel uncomfortable regarding specific data action*. The input from the crowd workers would be a set of free-text responses associated with a severity score, each explaining why users feel unconformable regarding a specific data action.

In practice, LPR asks participants to answer two questions (see an example in Fig. 7.2 middle) regarding each data action independently[12]:

- *How would you feel if the company [collects|shares|processes|uses] your data as described above?* This is a 5-scale likert-score question. Users can answer in five options (from "Extremely comfortable" to "Extremely uncomfortable").

[12]adapted from [121]

- *Why did you feel that way?*
  The is a open-ended question, which requires 100-character [9, 357] minimum length. Participants are asked to write the response in a template: "I feel [comfortable|...|uncomfortable] because XXXXX".

Through our survey iterations, we also found our participants were quite expressive in describing their major privacy concerns. The reported concerns also varied across different participants, so collectively aggregating them helped improve coverage. We quantitatively evaluate the benefits of aggregating inspection results in §7.9.

7.5.4 Dividing the survey into mini-surveys

The above design allows a non-tech-savvy participant to contribute to a privacy review. However, the length of the reviewing task still remains challenging for crowd workers. Inspecting a complex data practice can be time-consuming, taking up to two hours in our early surveys, which is a long period of time for many crowd workers [320]. On the other hand, arbitrarily breaking down a data practice inspection into smaller parts might not work either since data actions are highly interdependent. Breaking a data practice inspection into data action inspections might remove useful contextual information, making it hard for participants to understand the necessary privacy details. After testing three "divide and conquer" strategies (dividing up the survey by actions, by applications, and by practices), we decided to ask users to inspect the data practice at the granularity of data applications.

Each crowd task (i.e., Human Intelligence Task) contains four parts: consent page, tutorial examples, data application evaluation, and demographic data collection. In the tutorial example page, we showed participants the example answers for a data application derived from [392]. We used a Depth-first search algorithm to parse the tree typology and generate a set of data applications. The LPR-web then loads a pre-defined survey template and replaces the data action description placeholders (see Fig. 7.13) with the actual content in each data application. Each crowd task contains 1-2 data applications, and each survey page renders the evaluation questions for one data action. It often takes around 15 minutes for a crowd worker to complete a survey. A participant is only allowed to work in one data practice once.

7.6 Aggregating the privacy inspection results

A major challenge of introducing open-ended questions into privacy surveys is that they can be difficult to aggregate. For example, Felt et al. [121] used open-ended questions in their surveys and manually coded these responses to interpret users' contexts. However, practitioners may not have the qualitative research skills or time to analyze these open-ended responses. To address this challenge, we developed a crowd-based technique to aggregate open-ended privacy opinions, which transforms the free-text responses and the comfortableness scores into a list of privacy concerns, each associating with a magnitude score.

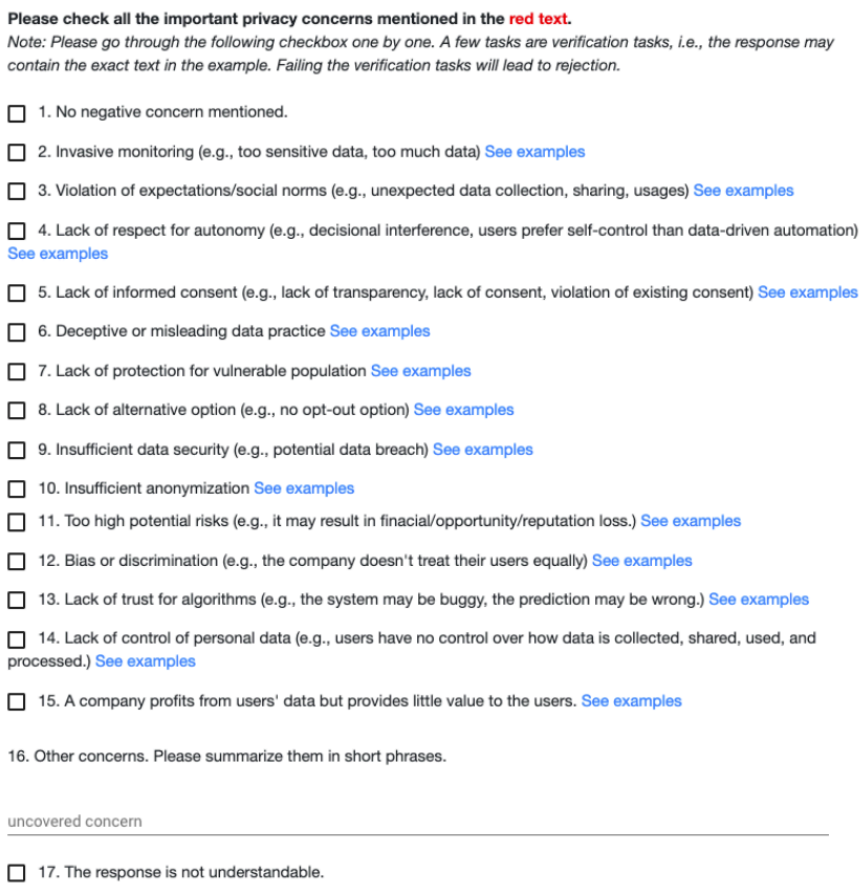


Figure 7.6: The final interface of LPR privacy annotation task. We ask each crowd worker to go through the privacy concern categories one by one and check all the concerns mentioned in users' free-text responses (collected from Fig. 7.2b).

7.6.1 Privacy Annotation

We organized the annotation tasks in a manner similar to traditional image labeling tasks. Each free-text response annotation is an independent Human Intelligence Task (HIT), and participants can complete as many tasks as they desire. We then presented each participant the data action description, the response text, and a set of privacy concern options. The annotator can select multiple privacy concerns or nominate new privacy concerns not covered in the provided list. In our experiments, we assign each response text to three different participants and consider an annotation valid if more than 2 participants annotate the same concern independently.

A list of common privacy concerns. At the core of the annotation is the list of common privacy concerns (Appendix §10.4), which is the basic vocabulary to characterize users' privacy concerns in LPR, facilitating communication between crowd workers and practitioners. Designing such a list is challenging.

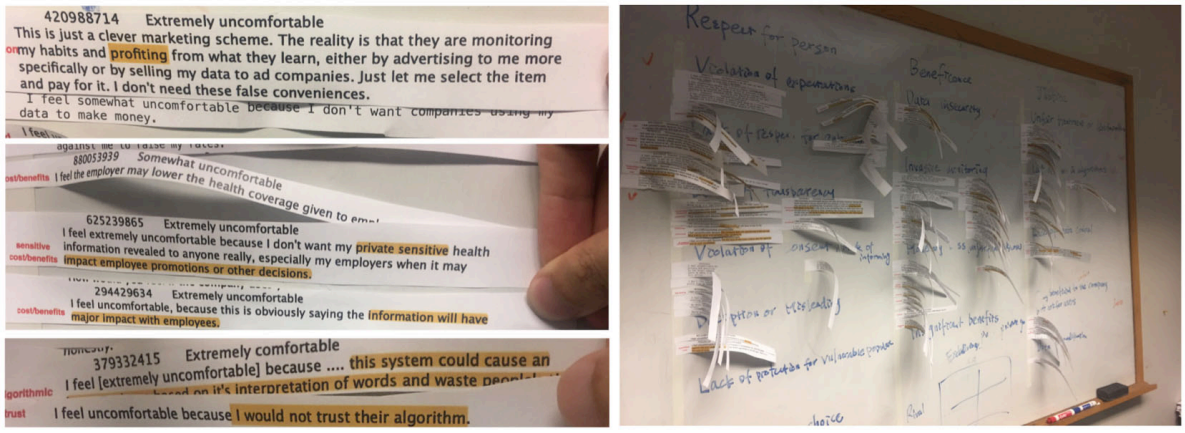


Figure 7.7: The development of a list of common privacy concerns. We first applied thematic analysis to our survey and interviews data to build the initial list. The left figure illustrates some example line-by-line coding results based on the survey responses. We then iterated the list with crowd workers, to ensure the list is easy to comprehend for non-specialists. We placed the example quotes, basic codes, and high-level themes on a whiteboard to track the development (right).

First, the list should be easy to understand for crowd workers, imposing little learning overhead. Second, the list should be comprehensive, covering most privacy concerns people may have. Third, the granularity of the list should be meaningful for practitioners, ideally connecting to some actionable items.

We used thematic analysis [39] to build the initial list. Two authors read through the free-text responses collected in survey iterations as well as the notes from previous semi-structured interviews independently and held weekly discussion. The aim of the analysis is to find out privacy concerns that the study participants had about our data practices. With this idea in mind, we performed the following steps of analysis: line-by-line coding of the survey and interviews' transcripts, with an aim to generate as many basic codes as possible, constant comparison and discussion of these basic codes, development and refine of higher-order themes to encompass and link these basic codes, and extensive case-based memo writing to track developing ideas. As the analysis proceeded, we also developed new privacy storyboards, refined the survey templates, and collected responses from different sets of crowd workers. We iteratively analyzed the data we collected throughout the process until we found consistent, recurring types of privacy concerns and no longer encountered new types of concerns, which suggested that we had reached data saturation. In total, we analyzed over 1000 responses to 67 different data applications.

We then iterated the list of privacy concerns and description text with crowd workers, to ensure the list is easy to comprehend for non-specialists. We further optimized the list for annotation efficiency by merging several conceptually relevant concerns, as we found that crowd workers often labeled them together. To help practitioners comprehend the list quickly, we further organized these privacy concern types into three

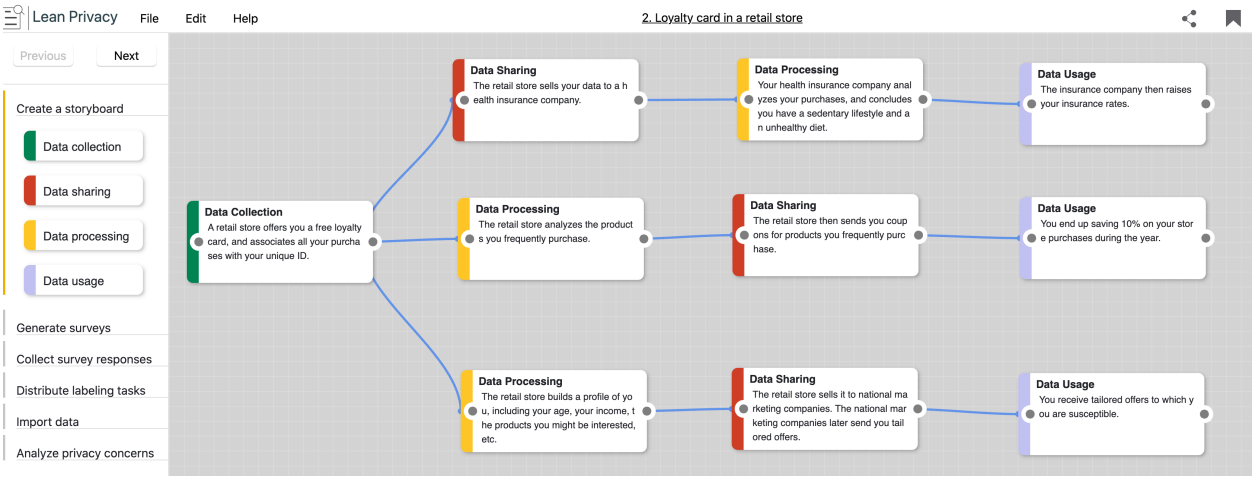


Figure 7.8: The overview of a LPR story in the web interface. The tree topology makes the story modular, so practitioners can incrementally modify and append more data actions.

high-level categories: respect for persons, beneficence, and justice (see Appendix Fig. 10.14). Figure 7.6 enumerates the text descriptions of the 14 concerns. Note the list of privacy concerns is non-exclusive, and so a free-text response may fall into multiple categories.

7.6.2 Computing the range and the magnitude of different privacy concerns

Through the privacy inspection surveys, a practitioner obtains a set of free-text responses $\{t_1, t_2, t_3 ...\}$ associated with severity scores $\{s_1, s_2, s_3, ...\}$, each explaining why users feel uncomfortable/comfortable regarding a specific data action. She then obtains a set of privacy concern annotations $A_k \sim \{a_{k1}, a_{k2}, a_{k3}, ...\}$ for each free-text response $\{t_k\}$ through the annotation tasks.

We define the range of privacy concerns of a data action as the union of annotated privacy concerns for individual free-text responses: $Range = A_1 \cup A_2 \cup A_3 \cup ....$ The range of privacy concerns of a data practice is the union of individual data actions' privacy concerns. We then quantify the magnitude of each privacy concern similarly as privacy risk estimation [277]. In essence, the magnitude is a function of the likelihood that a privacy concern expressed by the crowd workers multiplied by the level of their uncomfortableness. LPR quantifies the magnitude of each privacy concern as the sum of associated comfortableness scores from different participants divided by the total number of participants.

7.7 A web-based system to streamline LPR

We further developed a web-based system[13] to streamline the LPR process. A practitioner may use the worksheet (Appendix §10.1) to think through the data practice in the lens of LPR data actions and then create a storyboard using the web interface (Fig. 7.8). The practitioner can continuously modify the stories, append new applications, and make changes to the text descriptions. All the modifications would be saved to a remote server automatically.

Once finished, the practitioner may download a JSON file that can be imported into the Qualtrics survey platform[14]. The practitioner can then distribute the generated surveys (Fig. 7.2) through various channels (e.g., AMT, Google Marketing Platform), collect users' direct feedback, and import the survey responses to a HIT annotation template (Fig. 7.6). At the end, the practitioner can import the CSV files into the web system to view them quantitatively. The web system displays the results in an interface inspired by Heuristic Evaluation [270]. It has two views to support practitioners to navigate and consume the crowd inspection output: summary view and detailed view. A complete demonstration of the web interface is available in the accompanying video.

Summary view. Once the crowd inspection results are available, the web system shows an aggregated privacy concern spectrum (Fig. 7.9), which can help practitioners both gain a high-level overview and navigate to the specific comment by hovering mouse on a specific block. The usability problem visualization in Heuristic Evaluation [270] inspires the design of the spectrum, where each column corresponds to one participant, and each row corresponds to one type of privacy concerns. The redness in each block indicates the level of each participant's discomfort regarding the specific dimension.

Practitioners can view the privacy spectrum at the level of a data practice or a data application. For example, Figure 7.10 illustrates the spectra of three data applications in a data practice. At a glance, practitioners can find that crowd workers raise most privacy concerns in the data application 1 (Fig. 7.10 left), while the least privacy concerns in data application 2 (Fig. 7.10 middle).

Detailed view. When the practitioner taps a data action node in the review mode, LPR provides the details about the selected data action (Fig. 7.11). The detailed view aims to help practitioners dive into the details and gain a deeper understanding of users' privacy concerns. The detailed view contains three components: 1) a comfortableness distribution chart that helps authors understand the general acceptableness (Fig. 7.11 left); 2) a concern bar chart that visualizes the occurrences of privacy concerns in each category (Fig. 7.11 middle); 3) a filterable table that helps practitioners to read the raw responses associated with each privacy concern category (Fig. 7.11 right).

[13] http://leanprivacyreview.com/

[14] https://www.qualtrics.com

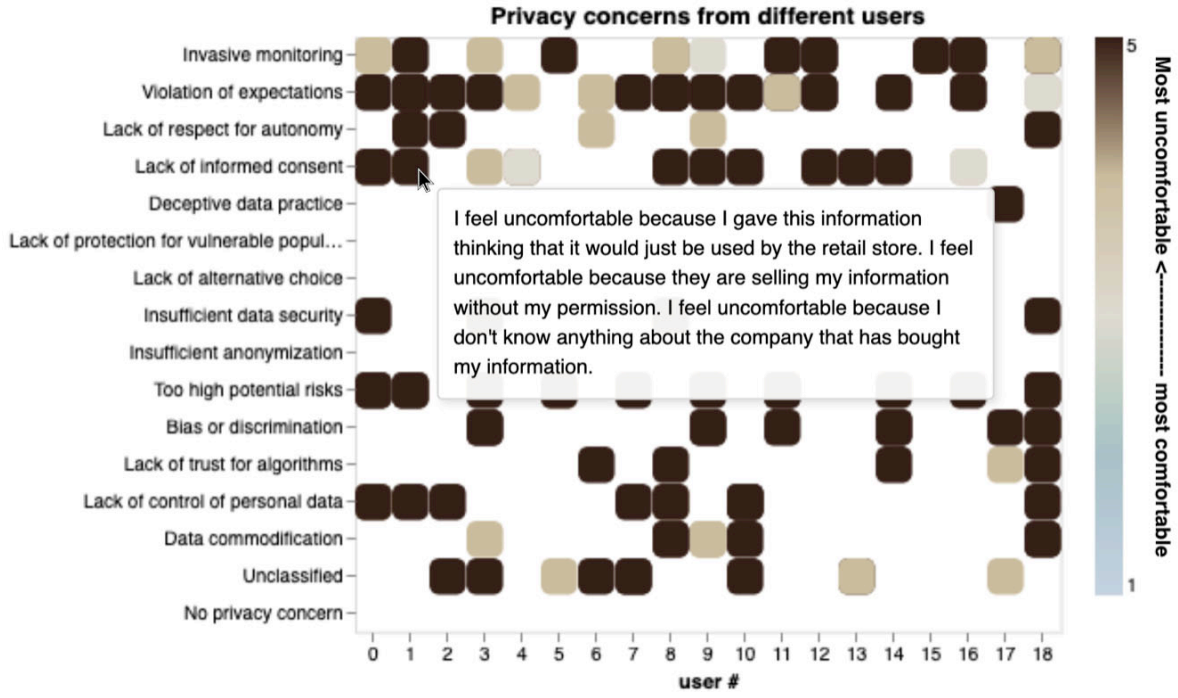


Figure 7.9: The summary view of of an example data practice: Loyalty card in a retail store [392]. The corresponding storyboard can be found in Figure 7.12. The privacy concern spectrum above renders privacy review results from 19 participants, showing who found which privacy concern. Each column corresponds to one participant, and each row corresponds to one type of privacy concerns. The redness in each block indicates the level of each participant's discomfort regarding the specific dimension. We do not include positive feedback in this privacy concern spectrum. When a practitioner hovers her mouse on a colored block, a tooltip containing the raw textual feedback will pop up. This grid plot is inspired in part by heuristic evaluation [270].

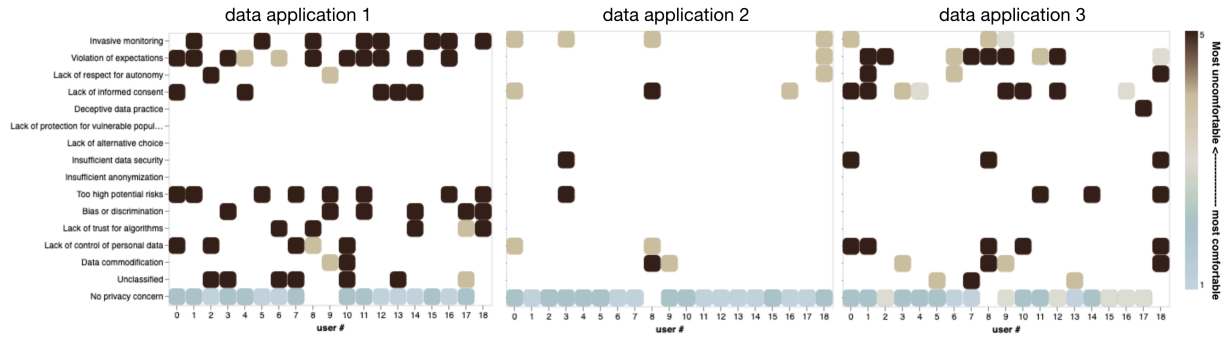


Figure 7.10: The summary views at the data application level for Loyalty card in a retail store [392]. We only annotate the free-text responses associated with non-positive scores, so all the positive responses fall into the "no privacy concern" category. While the total number of free-text responses are the same, the summary views are quite different. At a glance, the practitioners can find that the data application 1 (using the purchase data to determine insurance rates) raises most privacy concerns, while the data application 2 (using the purchase data to customize coupons) receives least negative comments.

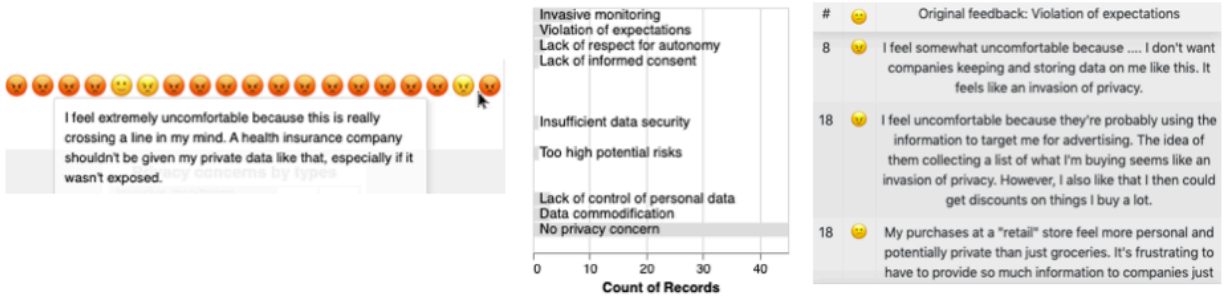


Figure 7.11: When the practitioner taps a data action node, LPR provides the details about the selected data action. The detailed view first shows a comfortableness distribution chart (left), helping the practitioner understand the general acceptableness. The detailed view then shows a concern bar chart (middle), visualizing the occurrences of privacy concerns in each category. Finally, the practitioner can explore the free-text responses by selecting the concern type of interest in the concern bar chart.

7.8 Case studies on real-world data practices

We analyzed 12 real-world data practices derived from media reports and research papers using the storytelling framework outlined in §7.4. The purpose of these case studies is twofold. We use these case studies to test the expressiveness of the storytelling framework, and later used these generated storyboards for LPR evaluations (§7.9).

7.8.1 Method

We selected 12 data practices across a diverse set of categories, such as IoT (Appendix Fig. 10.3, 10.9), eCommerce (Appendix Fig. 10.2, 10.3, 10.12), social networks (Appendix Fig. 10.6, 10.8), advertising (Appendix Fig. 10.1), computational psychology (Fig. 10.4, 10.11), data science experiments (Appendix Fig. 10.1, 10.6, 10.7) and scenarios involving vulnerable populations (Appendix Fig. 10.5, 10.6, 10.9). We intentionally excluded: (1) government surveillance since it often involves complex civil rights discussions, and (2) data breaches since these are not intentional parts of a system's design.

Individual reports or papers may only involve one data application; we combined the relevant stories if the underlying data actions are similar. For example, a social network company may use the emotion-sensing technique (Fig. 10.6) to run a psychology experiment [199], a teenage suicide detection [70], or an emotion-based advertisement targeting [213, 312]. Here, we focused on the scenarios and outcomes that are most common and natural based on a review of media reports. Although we did not enumerate all possible cases, practitioners can easily extend the story to cover more data applications in the future.

Three participants were involved in the story creation process. These participants had backgrounds in UX research, privacy research, and software engineering respectively, mimicking a real world setting in the industry [72]. Two of the participants were not aware of the storytelling framework in advance. We first briefed participants on the framework, then presented them with a set of compiled news media reports, and finally asked them to summarize these new reports and generalize a LPR story representation collaboratively. Each story creation took between 20-30 minutes.

Table 7.9: The list of 12 tested scenarios derived from media reports and research papers. We selected these data practices across a diverse set of categories, such as IoT (#3, #9), eCommerce (#2, #3, #12), social networks (#6, #8), advertising (#1), computational psychology (#4, #11), a data science experiment (#1, #6, #7) and scenarios involving vulnerable populations (#5, #6, #10).

#	Scenario name	Description
1	Search engine click-through data [21, 162]	Google runs an A/B test experiment to determine the best color for the web links in search results.
2	Loyalty card in a retail store [392]	Woodruff et al. [392] enumerated several common data usages associated with retail store loyalty cards.
3	Checkout free retail store [248]	Amazon Go stores install hundreds of cameras to enable checkout-free shopping experience and other smart features.
4	Game chat log [292]	Riot uses League of Legends chatlogs to weed out toxic employees.
5	Pregnancy intimate data [152]	The pregnancy-tracking app Ovia lets women record their most sensitive data for themselves - and their boss.
6	Social network sentiment analysis [70, 199, 213, 312]	Facebook may use the emotion-sensing technique to run a psychology experiment [199], a teenage suicide detection [70], or an emotion-based advertisement targeting [213, 312].
7	Online dating experiments [159, 391]	OkCupid runs two data science experiments to understand the romantic relationship: score manipulation & love is blind.
8	Email contacts for social network bootstrapping [156, 266]	Google appropriates the Gmail data to bootstrap Google Buzz, a social networking service.
9	A fitness tracker [301]	Sexual activity tracked by fitbit shows up in Google search results.
10	Predicting a woman's pregnancy [158]	Target uses users' purchase data to predict a woman's pregnancy, aiming to hook parents-to-be at that crucial moment.
11	An insurer employs AI [281, 374]	China's largest insurer, Ping An, starts to employ artificial intelligence to identify untrustworthy and unprofitable customers.
12	eCommerce Price discrimination [8, 62]	(1) Travelocity practices price discrimination using users' device information. (2) Uber finds that users are more likely to pay for surge pricing if the battery is low on their phones.

7.8.2 Results

Figure 7.12 illustrates an example privacy storyboard overview of data practice #2. We only offer a brief text summary for each data action in these illustrations. The complete set of all 12 illustrations is available in Appendix §10.2.

7.9 Evaluation: consistency, cost & latency and result quality

We demonstrate through our evaluation that LPR is cheap, fast, consistent, and can help practitioners find the concerns they are not aware of. Note that our evaluation focuses on the feasibility of using crowd workers to learn about potential privacy concerns pertaining to data practices rather than the usability of our

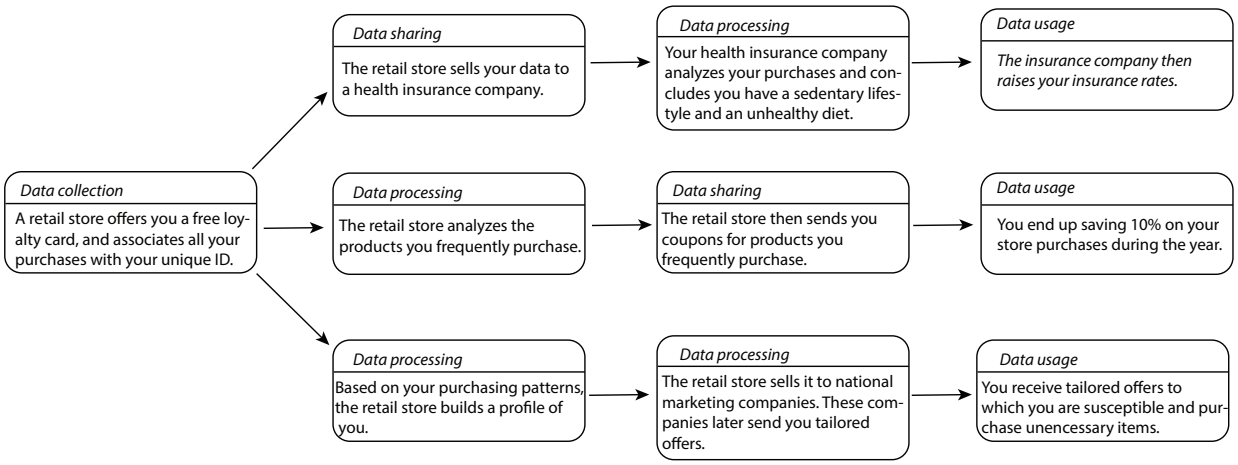


Figure 7.12: A privacy storyboard of scenario #2 "Loyalty card in a retail store "using the LPR framework. A retail store collects users' data through a loyalty card and uses the data for insurance and coupon personalization. Three participants created the storyboard collaboratively using the worksheet (Appendix §10.1).

support tools (e.g., our web site and our visualization). We leave these evaluations in future work.

Result summary: (1) Discovery of privacy concerns saturates as the number of evaluators exceeds 14 participants. (2) LPR can collect a consistent and saturated results in 5.5 hours, costing 3.7 hours of total crowd work (≈$80). (3) The crowd inspection found 89% of privacy concerns identified by a group of software and data practitioners as well as 139% more privacy concerns that practitioners are not aware of, at a 6% estimated false alarm rate.

7.9.1 Deployment and Apparatus

We deployed the generated surveys (Fig. 7.13 left) on Amazon Mechanical Turk (AMT) to test the practical applicability of LPR. We recruited 20 unique participants through TurkPrime [229] for each story and paid each participant $1 for completing a short survey (including less than 5 free text questions) or $2 for a longer survey. We also informed our participants that we will give an additional $1 reward if participants provide thoughtful reasoning in open-ended questions. To avoid priming workers during task selection [40], we advertised the survey as a "data practice survey", not as a survey specifically about privacy. On average, it takes 8 minutes to complete a short survey and around 15 minutes to complete a long survey. We granted bonus rewards to all the participants. So the average hourly pay for an inspection participant is between $12~$15.

We then distributed the neutral[15] and negative free-text responses in privacy concern annotation tasks (see Fig. 7.6) on AMT. Here, annotators are allowed to complete multiple tasks. We forwarded each free-

[15]We also forwarded the neutral responses since some neural responses contain both positive and negative opinions.

text response to three annotators and rewarded each completed task $0.20. LPR accepts the annotation if more than 2 participants annotate the response into the same concern. Each annotation task takes around 45 seconds, resulting in a hourly wage of $15.

Baseline. Our baseline is to mimic the real-world process: a small team (N=6) of practitioners need to guess users' reactions to justify the privacy design [53] when a formal privacy review is not available. We recruited 24 data practitioners (11 male, 13 female) through mailing lists and campus flyers. All participants have advanced degrees: 3 Bachelor, 14 Master, and 5 Ph.D. Participants also indicated that they have at least two years of experience in one of the following areas: User Experience Research (11), Software Engineering (7), and Privacy Engineering (6). 16 out of 24 participants have professional working experience in an IT company. We consider these participants have basic-to-intermediate knowledge about privacy, who have a comparable level of privacy expertise as the real-world teams [72].

We then tailored the survey template for practitioners (Fig. 7.13 right), in which we asked respondents to assume themselves as company employees, guess users' responses, and explain their rationale. We asked each data practitioner to review three different stories and complete the corresponding surveys, which took between 40 - 70 minutes. We compensated each participant $20 for their time.

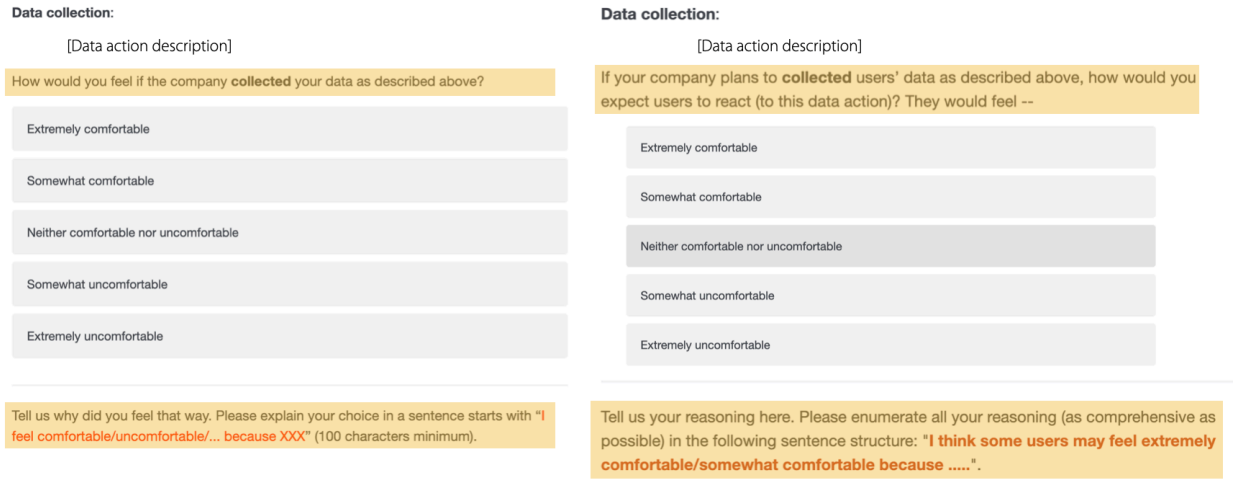


Figure 7.13: The survey templates for crowd workers (left) and the software and data practitioners (right). The left template asks participants (i.e., crowd workers) to imagine themselves as users. In contrast, the right template asks participants to imagine themselves as company employees and guess users' reactions. We highlight the differences between the templates in yellow boxes.

Ground truth. We recruited three trained HCI privacy researchers (one fourth-year Ph.D. student and two Post-doc researchers, one male and two female) to create the ground truth list of privacy concerns for each data action. None of these researchers participated in the earlier story creation process nor the authors of the paper. All three researchers have rich experience conducting user studies regarding users' privacy

concerns and have worked on usable privacy research for more than four years. These experiences allow them to anticipate privacy concerns better than most data practitioners. Although individual experts may suffer from biases and blind spots, we aggregate all three researchers' judgments to leverage the collective wisdom.

We first presented these researchers with the privacy concern taxonomy (Appendix §10.4) and briefly introduced the structure of the taxonomy. We then merged the results (i.e., lists of privacy concerns) from the crowd workers and the practitioners, and presented the combined results **blindly** to researchers. For each data action, researchers saw the text description, a list of merged privacy concerns, and the complete taxonomy. To avoid biases, researchers were not aware of the sources of individual privacy concerns.

We then asked the researchers to evaluate each privacy concern's importance on a 10-point scale and identify the "false positive" concerns (false alarms). We defined importance by adapting the definition of privacy risks into the context of privacy concerns: the likelihood multiplies the severity. We considered a privacy concern a false alarm if more than two judges marked it invalid. The inter-rater reliability for labeling false alarms was 0.82. We computed the ground truth list with magnitude by summing the ratings from individual researchers.

Caveats. The ground truth list may not cover some "unknown unknown" privacy concerns if all the participants (i.e., the crowd workers, practitioners, researchers) miss these concerns. Each data practice in the experiment has been inspected by 20 crowd workers, 6 data practitioners, and 3 researchers. It is very unlikely that any of the data practices would have any major privacy concerns which did not bother some of these participants enough to complain about it. However, it is still possible that the next subject would stumble over a new minor privacy concern. Therefore we stated the "known" number of privacy concerns for each story, and the statistics in §7.9.3 are based on the number of known problems.

7.9.2 Crowd Inspection Data summary: Free-text Responses and Annotations

In total, we collected 1182 privacy opinions from crowd workers regarding each data action, each containing a short free-text response associated with a comfortableness score (5-point scale). Figure 7.14 illustrates the distribution of the responses across different scores. Most participants took a positive or negative position on the data action, while only 6% chose to be neutral (i.e., neither comfortable or uncomfortable). The average length of the free-text responses is 213.3 (std=86.0) characters, and the longest response is 675 characters. Figure 7.15 illustrates the frequency distribution of different lengths.

We forwarded the neutral[16] and negative responses (#=739) in annotation tasks and collected three independent annotations for each response. Crowd workers annotated the aforementioned privacy concern using our provided list in 2197 tasks and reported "uncovered privacy concerns" in the other 20 tasks (less

[16]We noticed that many neutral responses contain both positive feedback and negative concerns. Participants assessed their comfortableness as neutral after balancing both perspectives.

than 1%). On average, each crowd worker checked 1.7 (std=1.0) privacy concerns in a privacy concern annotation task (Fig. 7.16). The average inter-rater reliability is 0.40 (std=0.33), indicating that crowd workers agreed with each other moderately well. Since crowd workers often stop labeling privacy concern categories once they find one or two match concerns (avg = 1.7 out of 15), this agreement is lower than the agreement between privacy researchers.

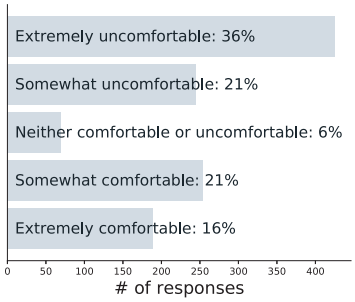


Figure 7.14: The aggregated distribution of responses in all scenarios across different scores. We present the individual distributions in Appendix §10.3.

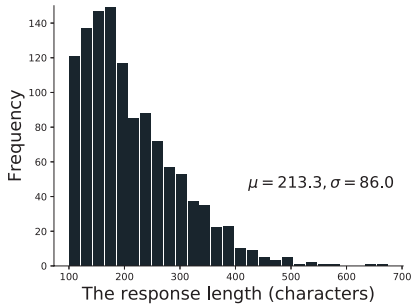


Figure 7.15: The aggregated frequency distribution of different response lengths in all 12 scenarios (Appendix Table 7.9).

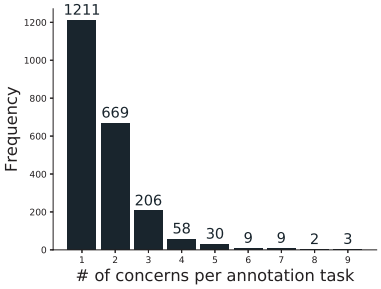


Figure 7.16: The distribution of the number of concerns found per annotation task. Most annotation participants only check one privacy concern in a privacy concern task.

For each uncovered privacy concern, participants provided a short description in the "other concern" textbox (Fig. 7.6). We manually reviewed "other concern" annotations and found that they were often covered in our provided list. For example, one participant described a new concern - "involves invasion of women's right to privacy," which we felt fell into the category of "lack of protection for the vulnerable population." We design the annotation task as a lightweight task (30-60 seconds), so it is a tradeoff that some participants may have a shallow understanding of the list. Indeed, LPR can still identify an accurate annotation by merging the annotations from multiple participants since the "other concern" annotation is infrequent (less than 1%).

7.9.3 Results

The output of the crowd inspection is a list of privacy concern annotations regarding different data actions. Each annotation is associated with a short free-text response, a comfortableness score (5-point

scale), and crowd workers' participation data (e.g., a start time, a completion time).

To characterize LPR's performance, we summarize our results based on the following three criteria:

- Consistency: Would discovery of privacy concerns saturate as the number of evaluators increases? How many crowd workers does LPR need to achieve a decent coverage?
- Cost & latency: How much does the crowd inspection cost? How long do practitioners need to wait for the feedback?
- Result quality: How good is the range of discovered privacy concerns? Can LPR prioritize the concerns well?

Consistency.

Our evaluation of the consistency is similar to the experiments Nielsen et al. [270] conducted for usability heuristic evaluation. We constructed hypothetical aggregates of varying sizes to test how many privacy concerns such aggregates would theoretically find. For each story, aggregates were formed by choosing the number of people in the aggregate randomly from the total set of crowd workers.

To recap, each privacy concern is always associated with a data action. If LPR finds that two different data actions (A, B) both have a privacy concern X, LPR recognizes A-X and B-X as two privacy concerns. However, if two free-text responses for the same data action mention a privacy concern X, LPR only considers it (i.e., A-X) as one privacy concern.

We further quantified the consistency in two settings: an explorative search and a tight-budget search. In the explorative search setting, we want to understand whether discovery of privacy concerns saturate as the number of inspection participants increases, so we considered all the reported privacy concerns valid. Instead, the tight-budget search focuses on a more realistic setting that practitioners may only have resources to address top privacy concerns. Since privacy opinions are often personal, some privacy concerns may only be reported by one individual. The tight-budget search considers a privacy concern valid if more than 2 out of 20 participants mentioned the same privacy concern for a data action.

Results: Figure 7.17 and Figure 7.18 shows the average proportion of privacy concerns found by each size of aggregate. These averages were calculated by a Monte Carlo technique, where we selected 10,000 random aggregates for each aggregate size and experiment.

Although individual inspectors only find few privacy concerns (tight-budget: 21.3% $\pm$ 5.7%; explorative: 13.3% $\pm$ 2.2%), collectively, aggregating their inspection results can reach good coverage. Aggregating 5 participants can find 66.3% of the privacy concerns in a tight budget search and 46.0% (std = 6.3%) of the privacy concerns in an explorative search. If we increase the number of participants to 14, the coverages will reach 97.1% (std = 1.2%) and 84.0% (std=4.4%), respectively. We set the 95% coverage as the saturation cutoff threshold. The discovery of privacy concerns saturates at roughly 14 participants in the tight budget search, and 18 participants in the explorative search.

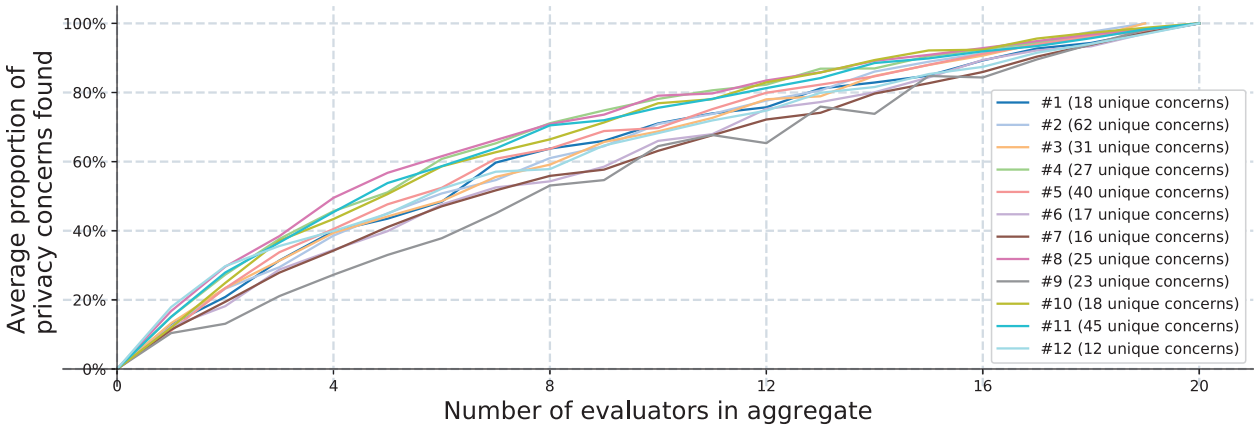


Figure 7.17: Proportion of privacy concerns found by aggregates of size 1 to 20 in the explorative search for scenarios #1 - #12. The explorative search analysis studies the discovery of privacy concerns saturation in an idealistic setting when the practitioners want to find any potential privacy concerns. Here we considered all the reported privacy concerns valid. On average, 18 participants can find 95.5% (std = 1.4%) of privacy concerns.

Cost & latency

Here we analyze the cost and latency of LPR to illustrate the benefits of crowd privacy inspection. A formal privacy review often requires a few weeks' turnaround time [38] and costs $10,000 to $60,000 in human labor for companies [249]. Our experiments show that LPR can offer a report of privacy concerns at a small fraction ($\approx$ 100X reduction) of the cost and wait-time for a formal review.

Results: Table 7.10 enumerates the cost breakdown of each data practice. On average, we spent around $114.45 for one privacy inspection, including $51.67 paid to inspectors (i.e., 4.52 hours of total crowd work), $36.95 paid to annotators (i.e., 0.76 hours of total crowd work), and $25.83 paid to the platforms. If we only involve 14 participants in the study, the average cost would be around $80 (i.e., 3.70 hours of total crowd work). In general, the cost to inspect a data practice increases as it involves more data actions. Scenario #2 is the most expensive data inspection, which also contains the most data actions (N=10). A negative data practice is more costly than a positive data practice since it receives more negative responses, resulting in more annotation tasks.

Table 7.10: The cost break down of tested scenarios. Since the actual cost also depends on the task reward, we also reported the total hours of crowd work.

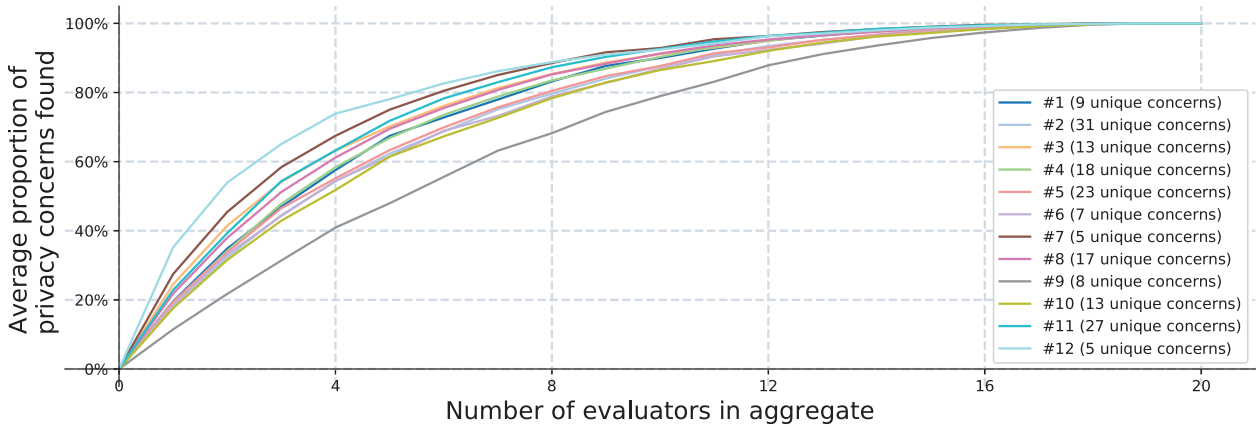


Figure 7.18: Proportion of privacy concerns found by aggregates of size 1 to 20 in the tight-budget search for scenarios #1 - #12. the tight-budget search focuses on a more realistic setting that practitioners may only have resources to address top privacy concerns. The tight-budget search considers a privacy concern valid if more than 2 out of 20 participants mentioned the same privacy concern for a data action. On average, 14 participants can find 97.1% (std = 1.2%) of privacy concerns.

	# of data actions	Inspection cost	Annotation cost	Hours of crowd work	$Cost_1$	$Cost_2$
1	7	($2.00+$1.00) x 20	$0.20 x 3 x 48	3.86 + 0.61 (hours)	$114.80	$80.36
2	10	($2.00+$1.00) x 20	$0.20 x 3 x 117	5.48 + 1.41 (hours)	$177.08	$123.96
3	6	($2.00+$1.00) x 20	$0.20 x 3 x 60	5.55 + 0.76 (hours)	$123.80	$86.66
4	6	($2.00+$1.00) x 20	$0.20 x 3 x 63	6.08 + 0.66 (hours)	$126.05	$88.24
5	5	($1.00+$1.00) x 20	$0.20 x 3 x 74	3.87 + 0.91 (hours)	$106.90	$74.83
6	6	($2.00+$1.00) x 20	$0.20 x 3 x 34	4.81 + 0.41 (hours)	$104.30	$73.01
7	4	($1.00+$1.00) x 20	$0.20 x 3 x 40	2.73 + 0.51 (hours)	$81.40	$56.98
8	4	($1.00+$1.00) x 20	$0.20 x 3 x 66	3.56 + 0.83 (hours)	$100.90	$70.63
9	4	($1.00+$1.00) x 20	$0.20 x 3 x 24	3.70 + 0.32 (hours)	$69.40	$48.58
10	3	($1.00+$1.00) x 20	$0.20 x 3 x 39	3.16 + 0.49 (hours)	$80.65	$56.46
11	6	($2.00+$1.00) x 20	$0.20 x 3 x 130	5.73 + 1.67 (hours)	$176.30	$123.41
12	8	($2.00+$1.00) x 20	$0.20 x 3 x 44	5.74 + 0.57 (hours)	$111.80	$78.26
Avg	5.75	$51.67	$36.95	4.52 + 0.76 = 5.28 h	$114.45	$80.11

Inspection cost = (Worker payment + Guaranteed bonus payment) x 20 Workers

Annotation cost = (Reward per assignment + Number of assignments per annotation) x Number of annotations

Transaction fee (not included) = TurkPrime Fee + Inspection Amazon turk fee + Annotation Amazon turk fee

Hours of crowd work = Time to answer surveys + Time to annotate responses

$Cost_1$ (The explorative search)= Inspection cost + Annotation cost + Transaction fee.

$Cost_2$ (The tight-budget search) = $Cost_1 \times 0.7$.

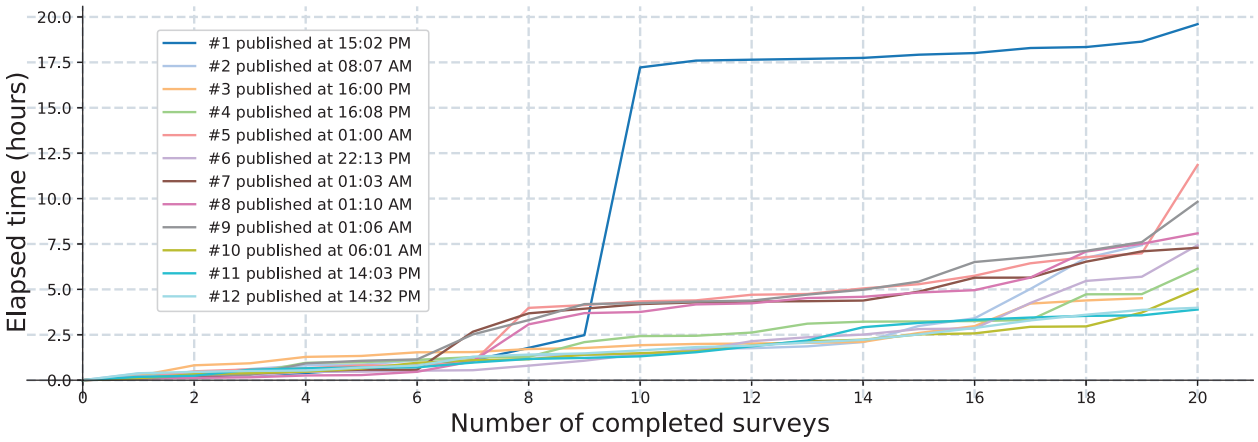


Figure 7.19: Inspection Latency for scenarios #1 - #12. Completing 20 surveys takes around 5 hours since the process is not entirely parallel. The large jump in #1 is because we tested the "Microbatch" feature in TurkPrime, which runs the HITs in sequence. The main latency comes from the locking mechanism in AMT. When a participant claims the task but does not finish, another interested participant needs to wait until the crowdsourcing platform releases the lock.

Figure 7.19 enumerates the survey completion times after releasing the complete set of tasks. The experiment for story #1 consumes more time since we tested the "Microbatch" feature in TurkPrime, which runs the HITs in sequence. The rest experiments all enabled the "Hyperbatch" feature, which ensures the fastest possible data collection by making the study available to all workers who may want to take it at the same time.

Figure 7.19 and Figure 7.20 shows the latency of inspection and annotation tasks. On average, it takes 3.30 (std = 1.16) hours to collect 14 inspection survey responses and 2 hours to annotate the responses, resulting in a total of latency of 5.5 hours.

Result quality

The crowd inspection outputs both the **range** and **magnitude** of different privacy concerns. We evaluate these two aspects by comparing the inspection results with the ground truth list.

Method (coverage). To evaluate the coverage quality, we measure the number of valid privacy concerns (Valid), the number of missed privacy concerns (Miss), and the number of false alarm (FA) for each story. We considered a privacy concern valid if the privacy experts did not mark it as a false alarm. We considered a privacy concern missed if the privacy concern is only discovered by the other inspection method but it was not labeled as a false alarm by the judges.

Results (coverage). Table 7.11 enumerates the raw numbers of Valid, Miss, and FA for both LPR

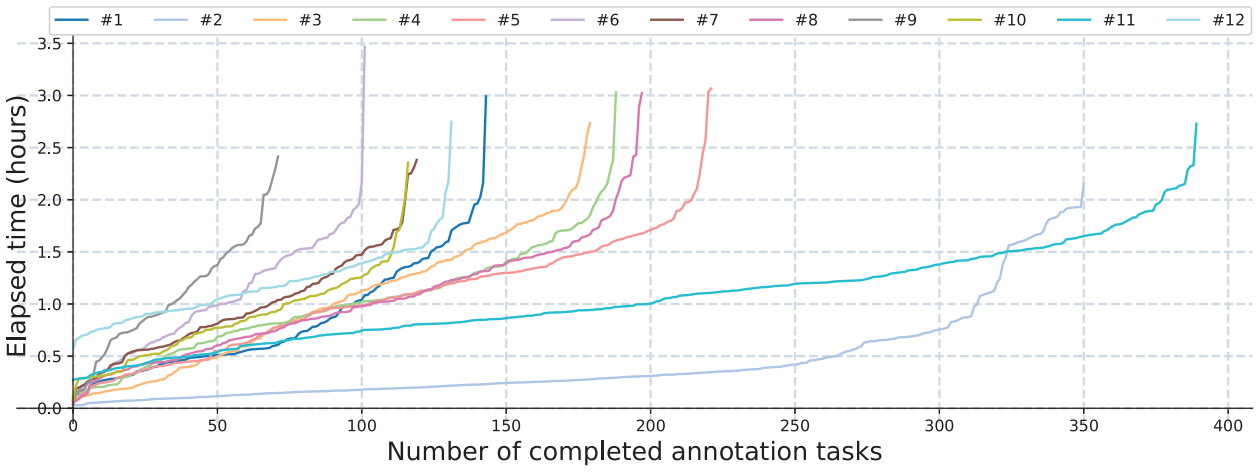


Figure 7.20: Annotation Latency. Annotation tasks are more parallel than the inspection tasks for two reasons: (1) each task takes less time to complete; (2) a participant is allowed to complete multiple tasks. However, participants are less motivated to complete the last few leftover tasks (usually one or two), which results in the extended tail distribution in elapsed time. Fortunately, LPR can still identify the annotation by merging the available annotations together without these few leftover ones. So practitioners need to wait about 2 hours to get the free-text response annotations.

and the baseline approach. In total, LPR finds 315 valid privacy concerns, misses 38 privacy concerns, and reports 19 false alarm privacy concerns; while the baseline approach finds 138 valid privacy concerns, misses 215 privacy concerns, and reports 12 false alarms. LPR finds 123 privacy concerns that are also identified by data practitioners (89.1%), as well as 192 privacy concerns that practitioners are not aware of (139.1%). The estimated false alarm rate for LPR and practitioners are 6.0% and 8.7%, respectively. LPR outperforms the baseline approach in all three metrics.

Method (magnitude). We define the magnitude of a privacy concern as the sum of associated comfortableness scores. For example, if two inspectors report a same privacy concern with "extremely uncomfortable" ratings (5), the magnitude of that privacy concern would be 10. For each data action, both approaches output a sorted list of privacy concerns based on the magnitude. We measure the Normalized Discounted Cumulative Gain (nDCG) [379] of each list to quantify the ranking quality.

$$nDCG_p = \frac{DCG_p}{DCG_m} \quad where \quad DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{log(1+i)} \tag{7.1}$$

where DCG_p is the Discounted Cumulative Gain (DCG) for a particular rank p, DCG_m is the DCG for an ideal order (i.e., the ground truth order), i is the order of a specific privacy concern in the output list, and rel_i is ground truth priority value for the privacy concern. Indeed, nDCG is the ratio of the DCG of a recommended order to the DCG of the ideal order. If the predicted list is perfectly aligned with the ground

Table 7.11: The privacy concern coverage across 12 stories. A privacy concern is a false alarm (FA) if the privacy researchers thinks it non-relevant. We consider a privacy concern valid if researchers did not mark it as a false alarm, and missed if the concern is only discovered by the other inspection method. LPR outperforms the baseline approach in all three metrics. We bold the numbers to highlight the winning method.

Participants	Story #1			Story #2			Story #3			Story #4		
	Valid	Miss	FA	Valid	Miss	FA	Valid	Miss	FA	Valid	Miss	FA
Crowd workers	**17**	**5**	**1**	**60**	**5**	2	**30**	**0**	1	**26**	**4**	1
Practitioners	13	9	3	20	45	**0**	6	24	1	12	18	**0**

Participants	Story #5			Story #6			Story #7			Story #8		
	Valid	Miss	FA	Valid	Miss	FA	Valid	Miss	FA	Valid	Miss	FA
Crowd workers	**37**	**4**	3	**15**	**0**	2	**15**	**2**	**1**	**23**	**1**	2
Practitioners	16	25	**2**	2	13	**1**	6	11	2	12	12	2

Participants	Story #9			Story #10			Story #11			Story #12		
	Valid	Miss	FA	Valid	Miss	FA	Valid	Miss	FA	Valid	Miss	FA
Crowd workers	**18**	**2**	5	**17**	**4**	1	**45**	**9**	0	**12**	**2**	0
Practitioners	7	13	**1**	7	14	**0**	28	26	0	9	5	0

truth list, the nDCG equals 1. An 0.8 nDCG indicates 80% of the best ranking.

Results (magnitude). We compute the nDCG scores for each data action and aggregate the scores of actions in the same story. Figure 7.21 enumerates the nDCG scores for all 12 stories. The privacy concern output from crowd workers achieves an average nDCG of 0.925, which outperforms the output from practitioners (0.693).

To gain a better understanding of the nDCG values, we present the magnitudes of the concerns associated with a data action in Table 7.12. As we can see, the results from crowd workers are closely aligned with the assessment by privacy researchers, with two minor mismatches: (1) crowd workers misses a low priority concern - lack of alternative choice; (2) crowd workers assigns a wrong priority to "violation of expectations," resulting in an nDCG of 0.945. On the other side, practitioners find the top privacy concerns correctly, miss four privacy concerns, and report a false alarm, producing an nDCG of 0.767.

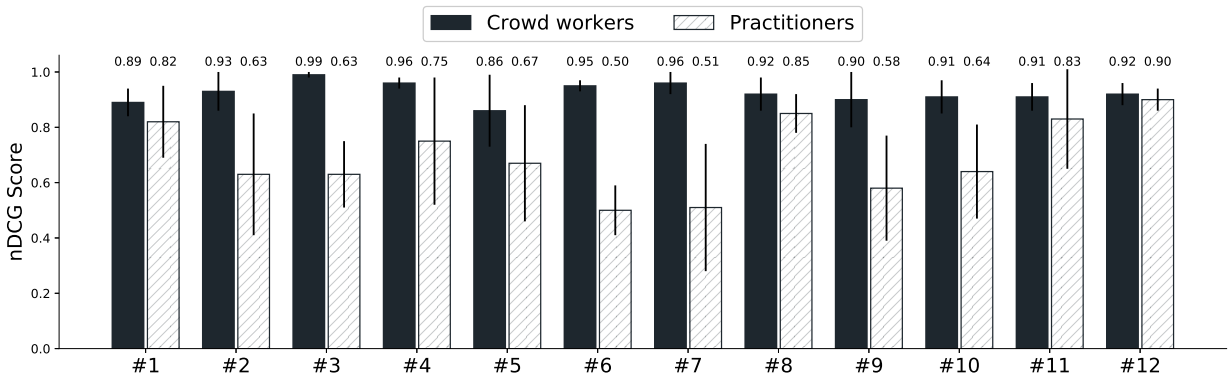


Figure 7.21: Normalized discounted cumulative gain (nDCG). If the predicted list is perfectly aligned with the ground truth list, the nDCG equals 1. An 0.8 nDCG indicates 80% of the best ranking. The privacy concern output from crowd workers outperforms the output from practitioners in all scenarios.

7.10 Discussion

7.10.1 When & how to use LPR?

Privacy feedback from actual users is often only collected after the software updates have been deployed (e.g., [231]). LPR makes it easier to collect user feedback about a data practice more frequently throughout the development lifecycle, similar to heuristic evaluation for usability feedback. We envision that the primary use cases for LPR are to (1) detect possible backlash over data collection/sharing/processing/usage when a formal privacy review is not available and (2) explore alternative benign data practice variants through storyboard iterations.

One potential application of LPR is the integration of Agile software engineering [30]. For example, a product manager may turn an ongoing "epic" or new "user story"[17] into a privacy storyboard, bring the results to an all-hands meeting discussion, and discuss privacy implications with the team. Past research [330, 392] shows that minor changes to a data practice design can significantly reduce users' disapproval and concern. However, exploring alternative benign variants through formal privacy reviews is challenging. The low-cost and lightweight nature of LPR can allow practitioners to iteratively improve their data practices based on fast user feedback, which can help a product team in avoiding implementing problematic software features/data practices.

Further, LPR can be used with great flexibility. First, practitioners can build knowledge of the data practice incrementally. A practitioner might not have all the required knowledge for a complex data practice. In that case, she can build storyboards for the familiar parts as a starting point, and incrementally add and modify data actions as needed. The design of the tree-typology and the decoupling of risk-benefit analysis

[17] "Epic" and "user story" are terms in agile software development, which refer to task narratives for sprint planning [22].

Table 7.12: The magnitude of privacy concerns for the "#7 score manipulation" action in Appendix Fig. 10.7. The results from crowd workers are closely aligned with the assessment by privacy researchers with some minor mismatches, resulting in an nDCG of 0.945. Practitioners correctly found the top privacy concerns, missed four privacy concerns, and reported a false alarm, producing an nDCG of 0.767.

Source	The magnitude of privacy concerns	DCG	nDCG
Ground truth	Violation of expectations (28), Deceptive data practice (25), Lack of informed consent (23), Lack of alternative choice (23), Lack of trust for algorithms (22), Lack of control of personal data (22), Insufficient data security (12)	101.27	1
Crowd workers	Deceptive data practice (52), Lack of trust for algorithms (28), Lack of informed consent (10), Violation of expectations (5), Lack of control of personal data (5), Insufficient data security (5)	95.67	0.945
Practitioners	Deceptive data practice (19), Lack of alternative choice (5), Data commodification (4)	77.68	0.767

support these usages. Second, the number of crowd workers involved is flexible. Our result shows that 14 participants found 97% of privacy concerns. However, practitioners may have fewer participants for early prototypes (e.g., refining wordings) and more participants for later stage prototypes, along with full privacy reviews.

7.10.2 Crowd worker engagement

Informally, crowd workers felt positive about the participation experience. At the end of all the crowd surveys, we asked participants to provide optional feedback regarding the task experience. Most participants commented favorably. One said: "*This was one of the best surveys I've done in a long time. Very thought-provoking.*" Another said: "*Thank you for the interesting survey. I enjoyed reading and reflecting on my feelings for each scenario.*" In addition, the average length of free-text responses exceeded (213.3 characters) the required length (100 characters), and a few participants enthusiastically wrote a multi-paragraph composition to elaborate on their concerns.

7.10.3 The actual cost & latency of conducting a LPR

Note that the actual cost and latency of reviewing a specific data practice depends on multiple factors, such as the task reward, the number of data actions, and the crowd worker market's engagement. Instead of predicting the exact cost for future LPR reviews, our goal here is to demonstrate that the crowd inspection

can be an order of magnitude cheaper and faster than a traditional privacy review, which often costs several thousand US dollars [252, 364] and has a turnaround time of a few weeks [177, 283].

In practice, practitioners can further reduce latency by releasing extra tasks. The main latency in our experiments comes from the locking mechanisms in AMT (see Fig. 7.20): when a participant claims the task but does not finish, another interested participant needs to wait for the crowdsourcing platform to release the lock. By releasing extra tasks, practitioners can collect enough responses without waiting for the last few locked tasks, although it will cost extra money.

7.11 Limitations: when does LPR not work?

7.11.1 Requirements for practitioners

LPR imposes two requirements for practitioners. First, practitioners need to communicate the data practice in an LPR story honestly. Since participants in LPR understand the data practice using the privacy storyboard rather than through real-world interactions, it is possible that different wordings and framings in the descriptions may impact their perception [40]. Techniques avoiding self-deception in business ethics [291] might alleviate this effect.

Second, practitioners need to have a comprehensive understanding of the data practice design and communicate it accurately to the participants. For example, when reviewing the Uber battery-based privacy surging data practice, a machine learning expert may challenge if the black-box machine learning model stealthily optimized the surge price based on users' smartphone battery data. However, crowd workers lack the expertise to foresee these types of issues. It would be the practitioners' responsibility to run extra sanity tests to understand the implications. Since LPR allows users to build and modify the storyboard incrementally, practitioners can potentially build such an understanding through multiple iterations.

7.11.2 Crowd worker representativeness

Our experiments in this chapter approximate the everyday user population by querying Amazon Mechanical Turk crowd workers. On-demand crowd workers can provide privacy-relevant feedback in a faster and cheaper way than dedicated privacy specialists (e.g., experienced privacy engineers, UX researchers). However, such an approximation may not resemble the intended user population for a given product or service. One potential solution to address this issue is to consider recruiting participants from multiple channels (e.g., marketing survey for real users, internal dog food users) or direct LPR to the targeted user group of the specific data practice in mind.

In our evaluation, all the crowd workers can only participate our study once to avoid the learning effect. However, in practice, if crowd workers get more experienced in LPR surveys over time, they might better

calibrate across data practices and spot common blind spots. Future work may look into methods to improve the productivity of crowd workers on privacy inspection.

7.12 Future work

7.12.1 Understanding the iteration process of data practice design

Our current study focuses on the feasibility and benefits of crowd inspection, which has not explored the iterative process of data practice designs. Future research should try to run multi-session studies with practitioners to observe how practitioners understand crowd workers' responses, derive new hypotheses for the sources of privacy concerns, and develop new data practice variations to validate their assumptions.

7.12.2 Privacy training for practitioners through LPR

Privacy training for practitioners has been challenging [72]. The current formal privacy review paradigm may perpetuate an undesirable trend: data practitioners tend to leave privacy considerations to dedicated specialists and do not think about them actively. In contrast, the design of LPR requires practitioners to be in the loop of the privacy decision-making process, allowing them to access the actual feedback from users and test their hypotheses at a low cost. Future research should run field studies to test whether practitioners' can better expect users' privacy concerns after using LPR, and whether this acquired privacy expertise are transferrable to unseen data practices.

7.12.3 Guidelines and best practices for authoring storyboards

To ensure quality storyboard, we asked three participants with some background in privacy to create the storyboards (see §7.8.1). We found that participants can create storyboards in 20-30 minutes. However, similar to UX storyboarding ([359]), novice users may encounter various challenges in the creation of storyboards (e.g., wording, flow organization). Future work should look into characterizing what kinds of privacy storyboards are most effective and what kinds of guidelines and tools (beyond our worksheet) can best help practitioners create those storyboards.

7.12.4 Summarizing positive opinions

The current design of LPR focuses on the negative opinions and can only summarize them into the magnitude and range of privacy concerns. For positive opinions, practitioners can only read the raw text indexed by comfortableness scores and users (e.g., Fig. 7.10 and Fig. 7.11). Future work should look into building a taxonomy for the positive opinions and building interfaces to help users contrast the positive opinions with the negative ones on the same topics.

7.12.5 Creating new privacy indexes to educate users and inform companies

It is possible to use LPR to collect privacy opinions from a large number of participants on exemplar data practices, making it possible to build a new type of privacy index that can quantify the ranges and magnitudes of privacy concerns for different demographics. We can use such an index in two ways. First, a consumer may compare their responses to the index to better calibrate their privacy sensitivity [75, 370]. For example, LPR can show a participant that 30% of other participants share the same privacy concerns of you, and 20% reported different privacy concerns. Second, the index can potentially depict a privacy norm for a specific data practice design in the form of aggregated privacy concerns (i.e., range and magnitude) quantitatively.

7.13 Conclusion

We introduce LPR, a fast, cheap, and accessible system to help practitioners evaluate users' privacy concerns of data practices in consumer-facing businesses. LPR takes a proposed data practice, quickly breaks it down into smaller parts, generates a set of questionnaire surveys, solicits users' opinions through the proxy of crowd workers, and summarize those opinions in a compact form for practitioners to use. Our experiments show that LPR finds 89% of privacy concerns identified by data practitioners as well as 139% more privacy concerns that practitioners are not aware of, at a 6% estimated false alarm rate.

LPR contributes three unique benefits to existing privacy solutions. First, LPR is more accessible than existing technologies, which can help small teams and individuals who do not have resources to hire dedicated privacy specialists. Second, LPR can help professional teams collect direct feedback from end users, ameliorating potential bias and blind spots. Third and more broadly, LPR advocates users' voices in the increasingly important privacy debate.

Chapter 8

Privacy Speed Dating: What Privacy Management Features Should Developers Build?

In this chapter, we introduce Privacy Speed Dating, a new design method for developers to prioritize independent features and identify critical design factors. We use this method to study people's smart home privacy-protective behaviors (SH-PPBs), to gain a better understanding of their privacy management do's and don'ts in this context. We first surveyed 159 participants and elicited 33 unique SH-PPB practices, revealing that users heavily rely on ad hoc approaches at the physical layer (e.g., physical blocking, manual powering off). We also characterized the types of privacy concerns users wanted to address through SH-PPBs, the reasons preventing users from doing SH-PPBs, and privacy features they wished they had to support SH-PPBs. We then storyboarded 11 privacy protection concepts to explore opportunities to better support users' needs, and asked another 227 participants to criticize and rank these design concepts. Among the 11 concepts, *Privacy Diagnostics*, which is similar to security diagnostics in anti-virus software, was far preferred over the rest. We also witnessed rich evidence of four important factors in designing SH-PPB tools, as users prefer (1) simple, (2) proactive, (3) preventative solutions that can (4) offer more control.

8.1 Introduction

In recent years, there has been growing interest in supporting users' privacy-protective behaviors (PPBs) on the web, allowing users to actively manage and protect their privacy [35]. Some examples include using tracking prevention tools (e.g., ad blockers or Do Not Track functions in a browser), periodically deleting web browser history and cookies, and entering fake information in web forms [176]. Previous studies have investigated how and why users adopt or reject PPBs in various contexts, such as identity theft [408],

[Character, Context]
John installed numerous smart home devices around multiple rooms in his house for security, sleep monitoring, video calling, watching entertainment, etc.

[Problem]
When setting up these devices, John tried to review and customize privacy settings. However, the descriptions of privacy setting options are confusing. Tired of looking through the app, John decided to continue with the default settings.

[Problem]
Lately, John read a report about hackers conduct privacy violations on the same type of voice assistants that he is using. The report criticized flaws in the voice assistant's default privacy settings.

[User Action, Technology]
John installs a privacy diagnostic app that can analyze and rate the privacy settings across all his devices. After the initial run, the app gives him a 72/100. It also pointed out where he lost points and gave step-by-step instructions on how he might improve the score.

[Resolution]
By following the app suggested instructions, John earned a 100/100. Now, he feels safer and more comfortable with all the devices.

Figure 8.1: Out of 11 different storyboards we created based on participants' wishlist features, *Privacy Diagnostics* was ranked as the most favored feature in 63% of the comparisons, when participants ranked three randomly selected storyboards. *Privacy Diagnostics* is a mobile app that can assess the smart home configurations across all the smart home devices and offer step-by-step instructions on how to improve the score. Participants praised it for "being simple," "offering privacy control," and "requiring little management efforts."

privacy lies [327], and web privacy tools [344].

However, an emerging yet less studied PPB context is how people manage privacy in smart home environments, which we refer to as smart home privacy-protective behaviors (SH-PPBs). Supporting SH-PPBs has many differences from the online context. For instance, users need to manage multiple smart devices running continuously in the background, each with different sensing capabilities that can collect private data, and each located in potentially sensitive locations within the home. These devices might also collect the data of people besides the primary user, such as roommates, guests, passersby, and neighbors. While there is limited support for empowering SH-PPBs compared to online PPBs, smart home early adopters are improvising tricks to protect their privacy. For example, on reddit, there have been online discussions on how to program smart plugs to power off cameras at certain hours [309], how to purchase devices that are still functional without cloud access [308], and how to place smart cameras inside the home to minimize privacy concerns [310].

We conducted two online surveys to understand existing SH-PPBs and explore potential opportunities to better support users' SH-PPB needs. Our first survey (N=159) used open-ended questions to investigate the practices, contexts, and desired improvements of users' SH-PPBs, as well as the barriers preventing users from taking part in SH-PPBs. This survey offers insights into users' current SH-PPB practices (§8.4.1) and their specific needs for supporting SH-PPBs (§8.4.2). Our second survey (N=227) then asked participants to critique and compare privacy protection concepts that we created based on the needs observed in the first survey, similar to the process of UX speed dating [79]. In doing so, we addressed two limitations of the first open-ended survey. First, participants often have a limited understanding of how devices work and what potential privacy threats a smart home may introduce, making it difficult for them to speculate about concepts beyond their own experiences. Second, commercial products only offer limited privacy support, and most

[Character, Context]
Annie installed multiple smart home devices in her living room. One day, she had some random conversations with her friends in the living room.

[Problem]
The next day, Annie noticed many Ads related to last night's discussion. She suspects that her smart home devices must have leaked her data.

[User Action, Technology]
Annie installs a tool that enables her to manage devices through a chatbot. She asks the chatbot, "Did my devices send my data out yesterday?" The chatbot reports all data that got sent out last night.

[Resolution]
Annie realizes that her conversations accidentally triggered the smart speaker, and part of her discussion was shared with Ads companies. She now decides to unplug the device whenever she has visitors over.

Figure 8.2: One of the less preferred SH-PPB storyboards was *Privacy Chatbot*, which was ranked as the least favored feature in 52% of the comparisons with other features. *Privacy Chatbot* monitors all the network data from smart home devices and allows users to ask privacy questions in plain English. However, participants criticized this feature since it "requires extra mental burden," "cannot prevent data leaks," "offers limited control options."

users have a relatively low baseline when evaluating a new privacy feature. This makes it hard to prioritize the privacy features to implement and identify important factors in designing SH-PPBs. Specifically, we created 11 storyboards, each depicting a privacy protection concept and how the concept might be used to address a privacy concern (see Figs. 9.1, 8.2, 10.15-10.23). We then asked each participant to critique three randomly selected storyboards in open-ended questions (i.e., what do people like/dislike about these scenarios?), and rank in order of their preference among the three concepts. We used the Plackett-Luce method [244] to merge these partial rankings into a global ranking across 11 concepts, similar to how the Elo rating system ranks chess players. We also analyzed the qualitative critiques and ranking rationale to identify the factors that most strongly influence the preference of a concept.

These methods allowed us to answer the following three research questions:

RQ1. What kinds of SH-PPBs are people already conducting, how often, and why? Our findings suggest that more than half of our participants (52%) actively do some kind of SH-PPB, despite the limited support and tedious effort required. The majority of reported SH-PPBs rely on ad hoc physical protection (57/80), such as powering off or physical blocking, while only two participants (1%) mentioned using third-party tools (e.g., Pi-Hole [161]). Among the 33 unique SH-PPBs we identified, the most common is participants manually unplugging a device based on different contexts.

RQ2. What types of support do users wish to have? Conversely, for people that have few or no SH-PPBs, what are the barriers that prevent them from doing so? Participants reported 18 types of wishlist features to enhance their SH-PPB experiences. The most popular one is a way to turn off (e.g., unplug) smart devices "really easily". Participants also wished to have more control of data collection (e.g., turning a camera's microphone off) and better privacy awareness (e.g., visual reminders for data collection). Indeed, most reported privacy features are reasonably easy to implement but do not exist in today's ecosystem. On the other hand, the most important SH-PPB barriers are "unaware of smart home privacy threats," followed

by "lacking knowledge and tools" and "requiring too much effort."

RQ3. What are potential opportunities for building tools to support future SH-PPBs? Of the 11 storyboards, a simple technique we called *Privacy Diagnostics* (Fig. 9.1), which is similar to security diagnostics in many kinds of anti-virus software, was preferred the most and far ahead of the rest. We note that only one participant in the first survey expressed desire for this feature, suggesting the usefulness of our online speed dating approach. We then used the quantitative rankings of storyboards to guide the analysis of the qualitative responses. We also witnessed rich evidence of four important factors in designing SH-PPB tools, as users prefer (1) **simple**, (2) **proactive**, (3) **preventative** solutions that can (4) **offer more control**.

Contributions: This paper makes the following contributions.

- We present the first study that systematically studies what users do and don't do in managing their privacy in smart home today. This study identifies 33 unique types of smart home privacy-protective behaviors, and finds that users heavily rely on protections at the physical layer and on ad hoc approaches.
- We present the results of online speed dating on 11 different privacy protection concepts based on needs observed from our first study. We found that a relatively simple technology, *Privacy Diagnostics*, was preferred far ahead of the rest in our study. We also identify four important factors in designing SH-PPB tools: simple, proactive, preventative solutions, and more control.

8.2 Related work

We have organized related studies into two categories: understanding privacy concerns in smart homes and privacy-protective behaviors.

8.2.1 Understanding Privacy Concerns in Smart Homes

Many studies have investigated users' privacy perceptions of smart home technologies [16, 17, 185, 219, 237, 322, 393, 401, 404, 406]. For example, Malkin et al. found that people were unsure of how smart TVs handled personal data, such as what data was collected, and how that data was used, re-purposed, and shared with third-parties [237]. Parents who bought smart toys for their children were concerned about whether their children's data would be collected and shared [250]. Children were concerned about whether their parents would be able to monitor them and hear their conversations through their toys [250]. Past research also found that users' privacy concerns vary across different data collection contexts [17], such as consent procedures, brands, data types. For example, Worthy et al. found that people's trust towards those entities that collected their data was associated with whether they would desire control of their information [393]. They argued that if users had less trust, they would seek a greater level of control [393]. Zheng et al. found that some people believed that entities collecting their information protect their data

carefully [404].

In contrast, our study focuses on how users **mitigate** their privacy concerns (RQ1 & RQ2). Past privacy concern studies have identified a few SH-PPBs through semi-structured interviews when participants cited them as anecdotal evidence to illustrate their privacy concerns. For example, Zeng et al. [401] reported two security and privacy mitigation strategies through interviewing 15 participants: isolating smart home devices in a separate network and only using indoor cameras when users are away. However, these serendipitous findings can hardly scale and generalize. Indeed, Zeng et al. suspected that users might change their behaviors to mitigate privacy risks but did not find such SH-PPBs. In contrast, we specifically examined SH-PPBs through surveys, offering a more comprehensive and systematic views of common SH-PPBs. For example, our results show that a few users modify their behaviors to mitigate privacy risks, such as dodging cameras and speakers.

8.2.2 Online and Smart Home Privacy-Protective Behaviors

There also has been much work on understanding the adoption of online PPBs, such as changing privacy settings based on the platform and audience [150], deleting online content completely [59], hiding true identity through lying to a partner [377], and adopting ambiguous language [59] and privacy lies [327]. Recent studies find that conducting online PPBs remains challenging for many users despite broad support. For example, when users mistakenly believed that web browsers provide online behavioral advertisement, they would clear the browsing history more often and deemed it an effective way to protect their privacy [395]. In contrast, we focus on the less studied context of a smart home, aiming to improve the understanding of existing SH-PPBs and explore opportunities to better support users' SH-PPB needs.

Past research has shed light on users' SH-PPB needs. For example, Yao et al. found that users want data localization, a private mode, and a network intrusion detector to have more control of their data [396]. In addition, Tabassum et al. found that users expect to give consent before smart devices share their data explicitly (e.g., conversation) [346]. However, these studies often only offer a few high-level principles, since they involve few participants, and users can hardly speculate beyond their own experiences. In contrast, the scale of our study allows us to build an understanding of users' SH-PPB needs in a bottom-up approach, grounded in users' actual SH-PPB experiences (RQ2). These insights further guided us to generate attractive privacy protection concepts.

Meanwhile, researchers and practitioners are actively developing privacy-enhancing technologies to support users' SH-PPB needs. For example, a user now can either unplug the smart speaker to protect sensitive conversations being recorded [297] or use a mute button to stop it from recording temporarily [209]. Users can also delete the conversation log in the app associated with smart speakers [209], set up access controls based on either the tasks that users are trying to accomplish [155] or the specific user who is trying to use the devices [155, 347, 400], and use third-party tools (e.g., IoT Inspector [166]) to increase their

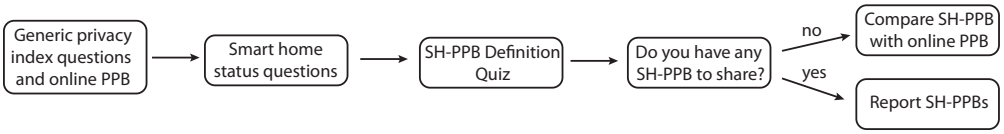


Figure 8.3: The first survey aims to collect qualitative descriptions about users' smart home privacy-protective behaviors. The base payment for the survey was $0.50, and we offered a $1 bonus for each reported SH-PPB. For those who had no SH-PPBs to share, we asked them to elaborate on any differences between SH-PPBs and online PPB experiences.

awareness of the data collected by smart home devices. However, most of these proposals are studied independently and evaluated with a relatively low baseline, making it hard to prioritize the features to implement for future products and identify the critical design factors. To fill this gap, in this study, we derive a large number of privacy protection concepts, compare them head to head, and offer a list of the pros and cons for each concept (RQ3).

8.3 Method

In this section, we describe our study protocol, recruitment process, and analysis methods.

8.3.1 Study procedure

We organized our SH-PPB exploration into two separate online surveys. The first survey posed open-ended questions to participants (N=159), asking them to share their SH-PPB experiences. This broad exploration helped us understand existing SH-PPBs while also learning users' needs to improve their existing SH-PPBs and search for new ones. Based on the findings from the first survey, we derived 11 privacy protection concepts (Table 8.7) and created structured storyboards to communicate the usage contexts of these concepts. We used a modified *speed dating* [79] method to evaluate these storyboards with a different set of participants through the second online survey (N=227).

SH-PPB Inquiry surveys: (Fig. 8.3) Participants were first asked about their perceptions of the severity of online privacy threats (e.g., "Having companies collect my online behavior is a problem for me?") and their online PPB experiences based on a set of questions adapted from a previous study [35]. We then asked about the types and the number of devices they have deployed in their homes. We used an initial list of 15 device types, supported by a popular home automation platform Home Assistant, and allowed participants to report any additional devices as freeform text.

We then presented participants with a short tutorial on SH-PPBs. We walked participants through a broad definition of SH-PPBs adapted from the online PPB context [35]: "*any strategies and actions you*

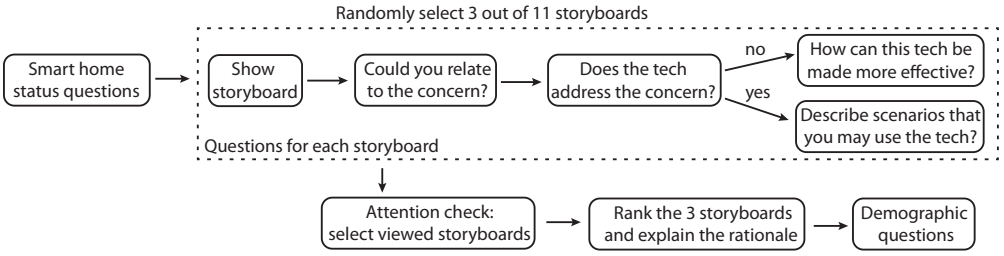


Figure 8.4: The second survey asks participants to critique three randomly selected storyboards using open-ended questions and rank their preferences. Each storyboard depicts an imagined situation with context, problem, proposed privacy protection concept, and users' reaction. To ensure data quality, we introduced an attention check question, where participants need to select the three storyboards they have viewed among 11 storyboards correctly. Participants were compensated with $4 for filling the survey, and the average time completion was 14 minutes.

take to mitigate the collection, usage, and sharing of your personal information from these smart home devices to protect your privacy." To continue filling out the survey, participants had to correctly answer a 3-item quiz to distinguish SH-PPBs from other interactions. Participants received feedback on incorrect answers, which they must then correct to proceed. In doing so, we made sure that users have a correct understanding of SH-PPBs.

After passing the quiz, participants were asked whether they have any SH-PPBs to share and were reminded that we offer a $1 bonus for each valid SH-PPB. Those who had SH-PPBs to share were then asked to describe the SH-PPB, relevant devices, associated privacy concerns, frequency of use, perceived effectiveness, and what features they wish were available to improve their SH-PPB experiences. Participants who reported online PPBs in the earlier questions but had no SH-PPBs to share were asked to elaborate on any differences between SH-PPBs and online PPB experiences.

Generating storyboards:

We first summarized users' specific needs for supporting SH-PPB by analyzing the wishlist features and the barriers that prevent users from conducting SH-PPBs. We then sought privacy protection ideas described in research papers (e.g., [110, 119, 183, 267**?**]) and online forums (e.g., [305, 307, 309]) that can potentially address the discovered needs. We discuss the storyboard selection criteria and process in §8.6.3. A typical challenge in soliciting users' feedback on a technology they have never experienced before is that users can hardly speculate on the imagined future and how technology could modify their behaviors. So, instead, we use a *speed dating* [79] like approach, presenting situated applications in a storyboard and asking participants to critique the storyboard. All storyboards (Figs. 9.1, 8.2, 10.15-10.23) contain 4-5 frames, covering the context, problem, privacy protection concept, and users' reactions, which are derived

from the first survey's responses. To ensure that storyboards have similar fidelity, the authors first created textual scripts and then hired a freelance illustrator to make all the storyboards.

Speed dating surveys (Fig. 8.4): We adapted the *speed dating* method, initially designed for semi-structured interviews, to online surveys. The key idea is to force users to critique and compare a few reasonable promising privacy protection concepts, which can help us understand the factors that most strongly influence the preference of a privacy protection concept. A major challenge in adapting *speed dating* to surveys is the participant engagement required in terms of time to compare 11 storyboards. To address this, we asked each participant to critique three randomly selected storyboards around two questions. The first question, "Could you relate to the concern?" was used to validate whether the observed users' needs are aligned with the actual needs perceived by users (i.e., concept validation). The other question, "Does this tech address the concern?", was to assess how well the proposed concept could address those needs (i.e., need validation). We asked participants to offer a 5-point Likert assessment for both questions and elaborate their reasoning in one open-ended question (>80 characters).

After the individual assessments, we posed an attention check question requiring them to select the three storyboards they viewed among all eleven storyboards. To help participants pass the attention check, we primed participants on the question at the beginning of the survey. After passing attention check questions, we asked participants to rank the three viewed storyboards and explain the rationale (>140 characters).

Survey development and pilot tests: Before officially launching the surveys, we ran three rounds of pilots (N=10) for each study and then iterated on the survey design based on the quality of the responses. For the first survey, the main challenge was to incentivize users to reflect on their PPB experiences, so we introduced the bonus compensation design. We iterated on the second survey to determine the number of storyboards each participant needs to review. We also added attention check questions to help us filter out inadequate responses.

8.3.2 Recruitment and Demographics

We surveyed smart home users through TurkPrime [229], a participant-sourcing platform for online research and surveys. To avoid priming, we advertised the survey as a "smart home technology survey"rather than one specifically about privacy. We restricted the survey to participants located in the United States, and mentioned that "participants need to have some experience in interacting with smart home technologies". We distributed the surveys in batches of ten across multiple days to solicit diverse participants. For the first survey, we stopped data collection when we did not find any new SH-PPBs in the last three separate batches, as that suggested saturation in results. For the second survey, we stopped data collection once each privacy feature had been compared and critiqued at least 60 times.

159 participants completed the first survey. Among them, 67 participants contributed 86 SH-PPB

practices, and 10 contributed more than one practice. The median task completion time was 5.4 minutes (mean = 6.7, std = 5.25) for crowd workers that did not report any SH-PPBs, and 9.5 minutes (mean = 10.8, std = 5.7) for workers that contributed at least one SH-PPB. The average hourly pay for participants was $12.50. 251 participants completed the second survey. We removed 24 who failed the attention check question in the later data analysis. To ensure our payments were fair to crowd workers, we only rejected assignments if a participant misidentified each of the three storyboards they were shown (N=11). The payment for the survey was $4, and the average time completion was 14 minutes.

Participants across two surveys reported various levels of experience with smart home technologies. Most participants reported deploying 5-10 devices (187) and 0-5 devices[1] (161), followed by 10-20 devices (31), 20-40 devices (5), 40-80 devices (2), The types of devices reported by the participants covered all 15 categories we had. Smart speakers (e.g., Amazon Echo, Google Home) were the most common device type (316), followed by Smart Lighting (194), Smart Thermostats (167), Robot Vacuums (164), Smart Cameras (127), and Smart Plugs (116). With respect to demographics, we had 154 females, 230 males, and 2 participants who preferred not to disclose their gender. Participants ranged in age from 18 to 74 (7% from 18-24, 42% from 25-34, 30% from 35-44, 13% from 45-54, 6% from 55-64, 2% from 65-74). Most reported having some college education (87%). The popularity orders of these distributions in the two surveys are identical, so we reported them in aggregate.

8.3.3 Analysis Methods and Metrics

For the first survey, we used an iterative, open coding process to analyze the open-ended questions. Two researchers first independently open coded all the responses. These openly generated codes were then collaboratively synthesized into a set of high-level codes, and the two researchers used the scheme to code the responses independently again. Upon completion, the coding team discussed potential extensions to the coding scheme. Once changes to the scheme were made, we re-coded all the responses with the new scheme. We conducted two coding iterations to reach a consensus. The overall inter-coder agreement was 0.86.

For the second survey, we used the Plackett-Luce method [244] to merge the partial rankings into a global ranking, and used thematic analysis [39] to identify the factors that most strongly influence the preference of a concept. To conduct our thematic analysis, we (1) first coded the ranking rationale to create initial pro and con codes for each concept; (2) collated these codes with the corresponding open-ended critiques; (3) grouped codes into high-level themes and cross-checked with the global ranking.

[1]Note that users may not deploy devices themselves and may sometimes interact with devices deployed by others.

8.3.4 Research Ethics

Our project was approved by the IRB at our institute. Participants need to read and sign an informed consent document before beginning the surveys. We reminded participants to focus on their own experiences and opinions and not reveal private or sensitive information throughout the surveys. Collected data is stored in a secure location accessed only by the research team. We only collected participants' contact emails for compensating them for their time. We deleted them afterwards and did not connect these emails to the rest of the study data.

8.4 Results

This section is structured along our research questions. We first present our findings concerning users' existing SH-PPBs, including how and why participants conducted SH-PPBs. Secondly, we summarized users' needs in improving their existing SH-PPBs and searching for new ones, and derived 11 privacy protection concepts. Finally, we present the speed dating results of comparing and critiquing the 11 concepts.

8.4.1 RQ1: What kinds of SH-PPBs are people already conducting, how often, and why?

To understand the current state of SH-PPBs, we first coded the reported SH-PPBs and clustered them based on devices, approaches, and privacy concerns. Many reported SH-PPBs were relevant but only differed in few key aspects, so we coded them to multiple applicable primitives. For example, one participant only powers on the camera when she is out of town, while another participant removes a cloth covering her camera when she leaves home. We coded the first as "on-demand turn-on" and "powering off," and the second as "on-demand turn-on" and "physical blocking."

High-level statistics: Out of all 159 participants, 67 reported at least one SH-PPB. Of these 67 participants, we excluded 4 who, combined, reported 6 SH-PPB practices that are not exclusive to smart homes (e.g., "using VPNs for online browsing," "taping the camera on a laptop"). For the other 92 participants, we found that 20 participants described implicit SH-PPB practices, such as "not placing smart cameras inside the house," or "only purchasing safe sensors like door sensors." Overall, more than half of our participants (83/159=52%) reported valid SH-PPBs. We note that this number is a lower bound, for two reasons. First, participants were asked to do a recall task (i.e. listing what kinds of SH-PPBs they do), which may be incomplete. Second, there may be subtle SH-PPBs that people engage in but did not think of including (e.g. choosing not to place devices in specific places in a house).

Cameras vs. Microphones vs. Other devices: Our coding process identified 14 unique SH-PPB primitives for smart cameras (Table 8.1), 13 for smart speakers (Table 8.2), and 9 for the other devices (Table 8.3). Participants were most likely to develop an SH-PPB towards cameras, followed by smart speakers and other

Table 8.1: Privacy-protective behavior primitives for cameras. The numbers in parentheses are the number of occurrences for individual categories. 68% of participants who own smart cameras reported a camera SH-PPB.

Time-based on/off	Examples
Using built-in features (3)	"I schedule my smart camera security system to automatically arm at our normal bedtime and disarm just before we get up each morning."
Using third-party solutions (1)	"I have my smart cameras set to be powered off at certain hours by programming smart plugs."
Location-based on/off	
Using third-party solutions (1)	"I setup a geo-bubble around my home so that when me or my wife's cellphone enters (about 50 feet around) our home cameras pause recording. I use an app called Ropr to accomplish this."
Manual on/off switch	
On-demand turn-on (5)	"I set up Blink mini cameras when I leave the house, but I unplug them when I am at home."
On-demand turn-off (4)	"Turn off device in some parts of the home when I'm at home."
Avoiding being captured	
Dodging cameras (3)	"I enter through the garage to avoid the doorbell camera showing what time I get home each day."
Positioning cameras (5)	"I have them positioned so that they face and record relevant areas - essentially doorways and windows - but I have my seating positions and 'workspace' areas (depending on the room) out of view from these cameras."
Configuring exclusion areas (2)	"I use exclusion areas on home security cameras to prevent unintentional recording in certain areas of my house."
Deadening sound (1)	"I purposefully keep the sound down on the TV so that the smart camera cannot pick up on it in case my wife is watching and hears what I am doing."
Temporarily turn off cameras	
Ad-hoc physical blocking (5)	"I put tape over the camera or unplug the device if I feel necessary."
Using built-in shutter (1)	"I close the shutter on Amazon Echo Show devices to prevent unintentional viewing."
Powering off (3)	"I unplug them (cameras) when I am at home."
Misc	
Deleting video footage (2)	"I delete footage from Ring doorbell (once a month), in case there is anything I do not want to be out there".
Account management (2)	"I encrypt my smart camera with passwords so no one can get into it."

devices. We found that 68% of participants who own smart cameras reported a camera SH-PPB, but the numbers drop to 29% for smart speakers and 7% for other devices. One potential explanation is that most participants understood the risks of audio and video recording and knew how to stop the devices from record-

ing (e.g., physical blocking, powering off), so they can improvise corresponding SH-PPBs accordingly.

In contrast, relatively few participants (6/159) had accurate mental models about potential inference threats regarding devices like lightbulbs and thermostats. For example, P16 understood the privacy threats of a thermostat correctly, "*we have turned off the learning mode for our Nest that tries to predict when you are home.*" P16 was concerned because "*it essentially tracks when we are home during the day and which days the thermostat is in use. Our schedule could be read if the information were ever hacked.*" But P43 misunderstood the privacy threats, leading to ineffective SH-PPBs: "*I changed the names of the settings on the smart thermostat so they aren't labeled 'home,' 'sleep,' etc.*" to prevent someone who can access this information from revealing his family's habits. Indeed, P43 spent significant effort on these less effective SH-PPBs: "*I changed the schedules on the smart lighting every three or four days, so they don't show such a consistent schedule of when there is someone home.*"

Built-in features, ad hoc physical features, and third-party solutions: In the online browsing context, most PPBs are due to either built-in privacy features provided by each website (e.g., opting-out of personalized ads) or third-party privacy features offered by the platform and privacy advocates (e.g., browser private mode, ad blockers). In contrast, there is relatively limited support for built-in privacy or third-party features for smart homes. For example, only 20 out of 80 SH-PPBs are enabled by built-in privacy features, including a built-in shutter (1) and mute buttons (8), time-based scheduling features (3), configuring exclusion areas for cameras (2), history management (3), guest access (2) and remote control (4). Only three SH-PPBs are empowered by third-party privacy features. P12 used an app called "Ropr" to set up a geo-bubble around his home so that when his or his wife's cellphone enters the bubble, the home cameras pause recording. The same user also "routed home internet traffic through a Pi-Hole ad blocking device so that web traffic can be filtered as needed to minimize the amount of data that leaves my home". P13 programmed smart plugs to power off his cameras at certain hours.

A unique aspect of smart home privacy is that users have more physical control over their devices. Indeed, the majority of reported SH-PPBs (57/80) leverage this fact to protect users' privacy. For example, users can position cameras to face and record less privacy-sensitive areas, such as doorways and windows (P8). P34 reported that they "enter through the garage to avoid the doorbell camera showing what time they get home each day." Many participants stated that they often unplug, turn off, block smart home devices when not in use, even though these devices are designed as always-on devices. We observed two interesting patterns in SH-PPBs for always-on devices: on-demand turn-on and on-demand turn-off. The first group keeps the devices off and only turns the devices on when necessary. For example, multiple participants mentioned that they only use Alexa to play music, or use cameras when they are out of town. In contrast, the other group leaves the devices on by default, but only turns them off when there is a guest or a private discussion. Both patterns are common for speakers (5 vs. 4) (Table 8.1) and cameras (13 vs. 9) (Table 8.2).

Ad hoc physical blocking is another common approach to stop devices running in the background. For

Table 8.2: Privacy Protective Behaviors for Smart Speakers. The numbers in parentheses are the number of occurrences for individual categories. 29% of participants who own smart speakers reported a speaker SH-PPB.

Manual on/off switch	Examples
On-demand turn-on (13)	"I always leave my Amazon Echo on mute unless I need to tell it something and then ask it to delete previous requests."
On-demand turn-off (9)	"Unplugging my partner's Alexa when I am casually hanging out."
Temporarily turn off speakers	
Through mute-button (8)	"I hit the mute button on the device so it's no longer listening to me."
Through the app interface (3)	"Disable Google assistant in smart devices using smart phones."
Through speech commands (1)	"Tell alexa not to listen."
Powering off (15)	"Unplug smart speaker to avoid audio recording."
Physical blocking (2)	"Throwing a towel over it when I want to discuss private things."
Avoiding being captured	
Dodging speakers (3)	"When I want to discuss private things, I let my partner go to the bathroom to talk in there."
Positioning speakers (2)	"I moved Alexa to an area of the home, where I am confident my voice cannot be picked up when I am in my home office taking phone calls."
Limiting usage scenarios (2)	"I just unplug it when not in use and plug it in when I want to use it for its timer or to play music."
Data management	
Disabling cross-device syncing (1)	"I turn off the syncing of the Alexa to the smart tv's at night so the Amazon account isn't broadcasting shared results from one device to another on what the family watches in separate rooms."
Deleting previous requests (1)	"... ask it (i.e., Alexa) to delete previous requets."
Network management (2)	"I route home internet traffic through a Pi-Hole ad blocking device so that web traffic and web requests can be filtered as needed to minimize the amount of data that leaves my home."

example, P3 reported throwing a towel over her partner' s Alexa when she wanted to discuss private things. P55 stated that he manually covered the camera in the hallway whenever he was at home and removed the cloth when he left. An extreme example of physical control leads to hardware modification. P31 complained that "it is becoming virtually impossible to find TVs without an Alexa in it," so he "physically removed the microphone/camera aspect of the device."

Implicit SH-PPBs: Many participants who did not explicitly report SH-PBBs still implicitly referenced them. In elaborating the difference between SH-PPBs and online PPB experiences, participants often de-

scribed that they do not fully adopt smart home technologies (N=9), where participants limited the number of devices and/or only adopt more privacy-friendly smarts home products. For example, P102 decided "not to use smart technology because he fears that someone else has access to it and can see/hear what he is doing." P80, who only adopts smart TV sticks and binary sensors, explained that he chooses not to utilize smart home devices to "keep his life simple and avoid hidden problems, albeit much more devices are available." Another important category of implicit PPBs is ad hoc physical privacy control (N=11), which has been discussed above. Since the descriptions of these implicit SH-PPBs are casual and often incomplete, we did not include them in Table 8.1, 8.2, 8.3.

Table 8.3: Unique SH-PPBs for other devices: thermostat, lights, lock, and Robot vacuum. The numbers in parentheses are the number of occurrences for individual categories. Only 7% of participants who own these devices reported a corresponding SH-PPB.

Noisifying personal data	Examples
Changing the schedules (2)	"I change up the schedules on the smart lighting so they don't show such a consistent schedule of when there is someone home."
Changing the configurations (1)	"I change the names of the settings on the smart thermostat so they aren't labeled 'home' and 'sleep' etc."
Misc.	
Disabling intelligent features (2)	"We have turned off the learning mode for our nest that tries to predict when you are home and turning on and using the thermostat."
Disabling cross-device syncing (1)	"I turn off the syncing of the Alexa to the smart tv's at night so the Amazon account isn't broadcasting shared results from one device to another on what the family watches in separate rooms."
Modifying the hardware (1)	"I intentionally avoid purchasing items that include putting a live microphone and/or video camera in my house.... In the few cases where I can't (no available "non-smart" options for example), I physically remove the microphone/camera aspect of the device and/or disconnect it from power when not in use."
Powering off (4)	"In the few cases where I can't (no available 'non-smart' options for example), I disconnect them (smart TV, Fire stick) from power whenever not in use".

Protection from whom and for whom: For each SH-PPB, we asked participants to explain their associated privacy concerns in free text. We then coded the attackers and the protected subjects, to surface users' goals in SH-PPBs. We observed five types of imagined attackers: general (50), service providers (11), remote hackers (10), physical intruders (7), and other residents (4). The majority of the mentioned PPBs (50/80) do not articulate who the attacker is. Instead, they often describe that they feel uncomfortable about potential risks due to undesired data collection, usages, and in-adequate security protections. While both service providers and remote hackers are commonly reported attackers, none of the participants mention any actual

negative experiences. Indeed, a few participants (N=6) ascribe these concerns to the privacy-related news and discussions with tech-savvy friends. In contrast, participants often associated "physical intruders" and "residents" with more concrete threats, such as "a thief can infer whether they are at home" or "another family member can see his TV viewing history or Alexa requests".

Besides, participants were most interested in protecting themselves (22/80) and their families (34/80). Only one participant mentioned that their guests feel uncomfortable with cameras that are turned on. None of the participants reported any SH-PPBs to protect passersby on the street or their neighbors. What's worse, some SH-PPBs may impose privacy attacks on other people. For example, P40 stated, *"I don't want to be captured on the cameras around my home... I bought them to capture other people/animals."*

SH-PPB frequency: Our results suggest that participants are actively performing SH-PPBs, despite the limited support and tedious effort required. Notably, 17 reported SH-PPBs were performed more than once a day. For example, P48 stated that "it has become a daily habit to unplug any devices with microphones when not in use." Furthermore, 16 reported SH-PPBs that were conducted on a weekly basis, such as moving a camera setup for a guest. Only 2 SH-PPBs were conducted on a monthly basis. For example, P20 stated that she "delete[s] footage from Ring doorbell once a month." Finally, 20 SH-PPBs are performed at installation time, which are mainly one-time setups. For example, P12, who set up a Pi-Hole to route home internet traffic, stated that "I set up the system to work passively, and it has been running for 8 months now continuously."

8.4.2 RQ2: What types of needs do users wish to support?

We also captured users' needs of SH-PPB supports from the first survey, by understanding (1) the features participants wished to have to address their privacy concerns better and (2) the barriers that prevent participants from conducting SH-PPBs. We then sought privacy protection concepts to address these observed needs and concluded with a list of 11 concepts used in the later speed dating experiment.

Most wanted features: Our analysis identified 18 types of wishlist features, summarized in Table 8.4. These desired privacy features expose three limitations of the widely adopted ad hoc physical approach. First, changing the configuration through the physical layer is inconvenient. For example, multiple participants expressed the desire to have an option to mute Alexa in a mobile app, so they do not have to move to unplug and plug it back in later. Second, these ad hoc approaches can only enable some basic, and often binary control. In contrast, participants want more intelligent privacy features, such as "a lid to cover the camera that gets automatically activated when I am at home," or "filtering family members in the video streams." To enable these types of features, we need more third-party solutions and built-in features. Finally, the ad hoc physical approach does not scale. For example, while P54 wants to "protect his home using smart cameras fully," he ends with only pointing the cameras to places he does not visit frequently. P54's wished-for feature

was "Selectively turn off the cameras when at home. I mean easily." Similarly, P57 wished to "Have a hub application where you can put on privacy mode for all devices with a click of a button."

Table 8.4: Most wanted features for managing smart home privacy. These features are reasonably easy to implement but do not exist. The numbers in parentheses are the number of occurrences for individual categories.

Better on/off control	Examples
Remote turn off/power off (13)	"Remote control through apps", "Wireless power switches"
Built-in scheduled on/off (9)	"Auto power off so I do not have to program a smart plug."
Context-aware auto-off (5)	"Built-in location-based on/off", "Auto-mute function on incoming calls"
Built-in physical block (3)	"Built-in shutter for cameras"
More control of data collection	
Fine-tune data collection (4)	"Turn the microphone off on the camera", "Fine-tune what exactly is captured", "Disable always-on".
Intelligent filtering (3)	"Maybe not send information when someone of the household is coming into the home, only when strangers are by the doorstep"
Opt out options for tracking (2)	"The ability to stop tracking our (TV) programming. I don't want or need future recommendations."
Better awareness	
Data practice transparency (4)	"More clarity from the service providers (Google in this case) as to what is being sent/received and how my information is being used"
Weekly summary (3)	"A review of the audio logs each week"
Device live-status (2)	"I wish that the Echo Dot would light up and stay lit up whenever the microphone is active. We sometimes forget to turn the microphone off and seeing a visual reminder would be very helpful."
Privacy rating (1)	"Some kind of certification/rating system to assess and rate smart device."
Login alert (1)	"Alerts to notify if someone tries to sign into my camera account".
Other: usability, data management, unauthorized access, scale, etc.	
Easier to delete things (4)	"Automatic deletion", "Weekly reminder to delete", "An easy way to review the data that's gathered and the ability to remove it."
Complex wake word (2)	"Maybe it could have a more complex wake word. Alexa thinks a lot of other words are Alexa. Even then I don't know as I would trust it."
Guest session (2)	"Incognito mode similar to browser history", "Creating a guest session for smart speakers."
Local-only network (1)	"Easier ways to limit device to only connect to other, local devices."
Better tutorials (1)	"I'm not sure what types of functionalities are included in the products."
Centralized management (1)	"Have a hub application where you can put on privacy mode for all devices with a click of a button."

Beyond control, another important theme behind these desired features is better awareness. For example, three participants wanted to receive a weekly review of the audio logs. In addition, two participants wanted the smart speakers to light up when it is listening for wake words, so they would not forget to turn the microphone off. Finally, participants also wish to have a third-party privacy-rating system that can assess and rate smart devices.

Barriers to SH-PPB adoption:

To elicit factors that prevent participants from conducting SH-PPBs, we asked the 87 participants who did not report having any SH-PPBs but did report online PPBs to "Compare and elaborate on any differences between managing online browsing privacy and smart home privacy" (Fig. 8.3). We excluded 10 participants who only discussed online PPBs and 20 participants who described implicit SH-PPBs, and reported on the themes that emerged from analyzing the remaining 57 responses. We organized these themes using a security sensitivity model [76], which argues that security features remain unused due to three reasons that we summarize in Table 8.5. These include: (1) **awareness**, i.e., participants who are either unaware or do not ascribe importance to privacy threats (24/57 participants); (2) **ability**, i.e., participants who are unaware of tools and methods (13/57 participants) and (3) **motivation**, participants who do not perform SH-PPB due to lack of trust in the tools, their cost, or effort needed (20/57 participants) .

Table 8.5: Breakdown of main reasons that prevent users from conducting SH-PPBs (N= 57). We organized these reasons using the security sensitivity model [76]. Overall Cohen' s kappa scores indicate very high agreement (0.85). The numbers in parentheses are the number of occurrences and Cohen's Kappa scores for individual categories.

Awareness: participants are unaware of relevant privacy threats.	
Unaware of SH Privacy threats (16 | 0.84)	e.g., trusting devices, never considering SH privacy threats
SH privacy not important. (8 | 0.75)	e.g., nothing to hide in SH, limited device intelligence.
Ability: participants do not know when, why and how to implement pro-Security and Privacy behaviors.	
Lacking ability for SH-PPBs (13 | 0.91)	e.g., unaware of potential tools and methods
Motivation: participants either do not trust in the efficacy of pro-Security and Privacy behaviors to defend against Security and Privacy threats or believe the costs of doing so are too high relative to its benefits.	
Doubting the efficacy of SH-PPBs (4 | 1.0)	e.g., desire to protect their privacy but feel unable to do so
"All or nothing" dilemma (6 | 0.76)	e.g., have to sacrifice privacy to use the service
Requiring too much effort (10 | 0.88)	e.g. more devices and data actions than online browsing

Privacy protection concepts to address SH-PPB needs:

We searched for privacy protection concepts that can either enable the wishlist features or reduce barriers to adoption of SH-PPBs. Most concepts are from privacy research literature (e.g., privacy mirrors [267], privacy nutrition labels [110]) and online forums (e.g., guaranteed protection [309], privacy presets [306]). We then mapped these concepts to applicable needs and stopped the search process after covering all of the observed needs. For example, *Anthropomorphism Icons* is motivated by one response where a non-tech-savvy participant was interested in knowing the data collection capabilities, but in her case all of the devices were purchased and installed by her husband. This process outputs an initial list of 27 concepts. After merging the concepts that overlapped, we concluded with 11 promising smart home privacy protection technologies (Table 8.7).

8.4.3 RQ3: What are potential opportunities for building tools to support future SH-PPBs?

This section presents the results of our online speed dating experiment, in which we rapidly explored these 11 application concepts and their potential interaction with users.

Quantitative analysis:

Ranking and comparison. In the speed dating survey, each participant ranked three randomly selected storyboards based on their general preferences. We then used the Plackett-Luce method [244] collated together these partial rankings to create a global preferred order of all 11 concepts (Fig. 8.5). The Plackett-Luce method is based on a classic probability model that can predict the outcome of a paired comparison, similar to how the Elo rating system ranks chess players. Fig. 8.6 illustrates the probability distribution that a storyboard wins over another. *Privacy Diagnostics* was the most favored storyboard, where 63% of participants ranked it as the most preferred and 10% ranked it as the least preferred. In contrast, *Anthropomorphism Icons* was least favored, where only 25% of participants ranked it in the top, but 52% ranked it in the bottom.

Concept validation & need validation. In the storyboard critique stage (see Fig. 8.4), we validated the concept (i.e., whether the observed need is aligned with users' actual need) and the need (i.e., whether the proposed solution can address the described need) for each storyboard. Overall, participants acknowledged most observed needs (Fig. 8.7) but expressed more diverse perceptions regarding the effectiveness of different solutions (Fig. 8.8). One example is *Privacy nutrition labels* (see Appendix Fig. 10.23), which was ranked at 9th in the global ranking. While participants related the most to the described need, namely, users wanting to know more about smart home devices' data practices before making a purchase decision, they expressed multiple concerns (see §8.4.3) regarding the solution's effectiveness.

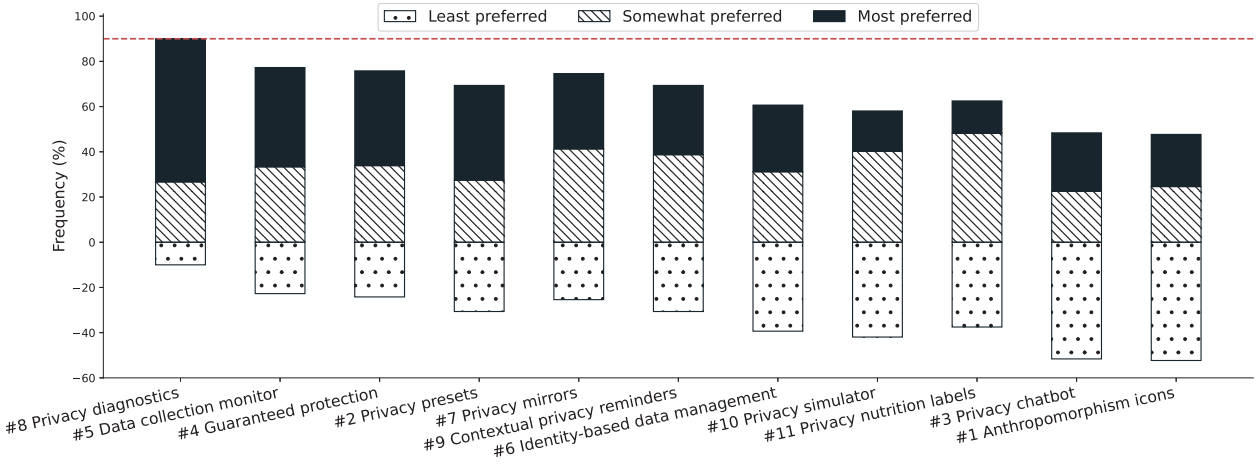


Figure 8.5: We used the Plackett-Luce method [244] to merge partial rankings into a global preferred order of 11 privacy protection concepts, similar to how the Elo rating system ranks chess players. From left to right, concepts are ranked in decreasing order of preference. A higher bar indicates more preferred responses. The dashed line facilitates the comparisons between other concepts and *Privacy Diagnostics*.

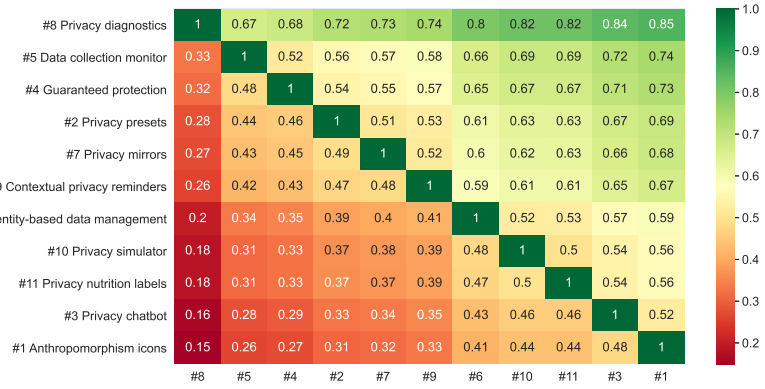


Figure 8.6: The probability distribution that a storyboard (row) wins over another storyboard (column). For example, a participant has a 85% chance to prefer *#8 Privacy Diagnostics* to *#1 Anthropomorphism Icons*.

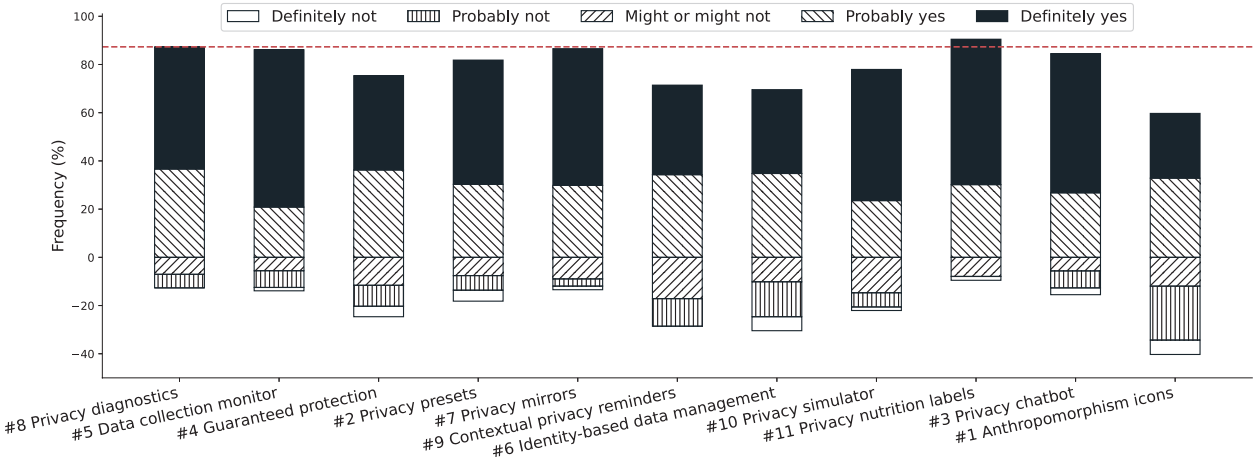


Figure 8.7: The distribution of answers to "Could you relate to the concern?" Participants reported that they could relate to the concerns described in *Privacy nutrition labels* the most, followed by *Privacy Mirrors* and *Privacy Diagnostics*. A higher bar indicates more preferred responses. The order from left to right is consistent with Fig. 8.5. The dashed line facilitates the comparisons between other concepts and *Privacy Diagnostics*.

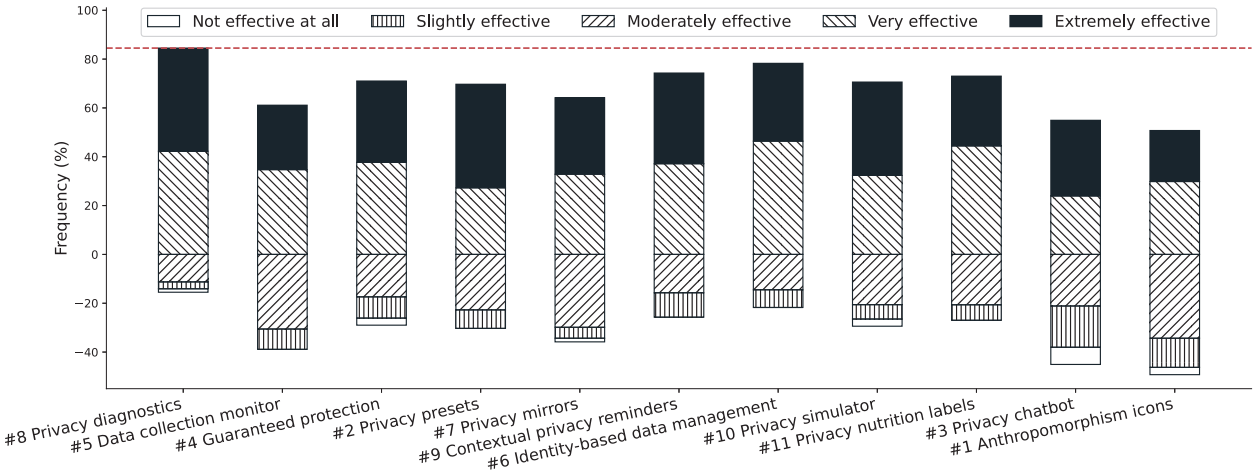


Figure 8.8: The distribution of answers to "How effective does this tech address the concern?" Participants reported that *Privacy Diagnostics* could most effectively address the corresponding needs, which may explain why it was ranked as the most preferred technology. A higher bar indicates more preferred responses. The order from left to right is consistent with Fig. 8.5. The dashed line facilitates the comparisons between other concepts and *Privacy Diagnostics*.

Table 8.6: The 11 privacy protection concepts we storyboarded and evaluated using speed dating. We analyzed users' SH-PPB needs from the first survey and then sought privacy protection concepts in the literature until we had covered all the identified needs. We present most storyboards in the Appendix due to space reasons.

#	Storyboard	Description
1	Anthropomorphism Icons	A third-party hub that displays devices' data collection capabilities in the look of a robot avatar (Fig. 10.15).
2	Privacy Presets	A third-party central hub that allows users to switch between pre-configured global privacy modes (e.g., party, business, out-of-the-town) and accordingly updates the settings across individual devices (Fig. 10.16).
3	Privacy Chatbot	A third-party chatbot that monitors outgoing network traffic requests and allows users to ask privacy-relevant questions through a chatbot (Fig. 8.2).
4	Guaranteed Protection	A smart plug that controls a device's power supply based on users' specified schedule of active hours for the device (Fig. 10.17).
5	Data Collection Live Monitor	A third-party application that monitors registered devices and enables users to view the live data collection status, including when, how, and where the data is going (Fig. 10.18).
6	Identity-based Data Management	A data management feature that associates captured audios and videos with residents' identities. A resident can not view the data associated with other residents (Fig. 10.19).
7	Privacy Mirrors	A data management feature that offers users a report about the company's knowledge of users and allows users to wipe some knowledge so that the company will forget them [267] (Fig. 10.20).
8	Privacy Diagnostics	A third-party mobile application that rates privacy settings of existing devices, provides explanations on deducted points, and gives step-by-step instructions on how to improve (Fig. 9.1).
9	Contextual Privacy Reminders	A built-in feature that reminds users to turn the devices off if the device guesses that the users might want to (Fig. 10.21).
10	Privacy Simulator	A third-party application that enables users to experiment with a smart home device's reactions and data collection practices to simulated actions in a virtual environment (Fig. 10.22).
11	Privacy Nutrition Labels	A table labeled on the packaging of a smart home device which contains concise data practice facts and a QR code linked to a website with detailed device information [110] (Fig. 10.23).

Qualitative results:

Our thematic analysis suggested that participants had different privacy concerns, varied levels of trust towards developers, and the time/money they were willing to spend to protect their privacy. This may explain why we did not find any storyboard that was consistently the most preferred across the 228 comparisons. For example, participants expressed three different types of trust regarding *Guaranteed Protection*. The majority of the participants recognized the value of *Guaranteed Protection* since they did not trust smart home device developers and wanted to have an extra layer of protection. In contrast, a significant portion of participants felt it was unnecessary since they trusted the developers, and a few participants preferred to manually shut off the devices since they did not trust the smart plug either. As a result, the concept validation ranking of *Guaranteed Protection* was relatively low. Another example is *Privacy Nutrition Labels*, where participants expressed that they want different levels of details. A few participants complained that the label contained too much information and wished for an energy-efficiency-like rating instead. In contrast, other participants felt the most helpful part was the QR code to the product website and preferred to look for more details online. These distinct personal preferences towards solutions also make need validation ratings (Fig. 8.8) generally lower than need validation ratings (Fig. 8.7).

Despite the fact that many factors were subject to personal preferences, we witnessed rich evidence of four unanimously important factors in designing SH-PPB tools, as users prefer (1) simple, (2) preventative, (3) proactive solutions that (4) offer more control.

Simple. Participants frequently mentioned that the top 4 storyboards, namely, *Privacy Diagnostics*, *Data Collection Live Monitor*, *Guaranteed Protection*, *Privacy Presets*, were simple. For example, P302 stated *"I LOVE the idea for privacy presets. For a tech savvy person, it presents a perfect way to manage the settings of everything, to integrate them in an easy to understand way."* Indeed, these concepts have relatively simple mental models comparing to many lower-ranked ones, making it easy for participants to understand the technologies and expect the output of using them. *Privacy Diagnostics* and *Data Collection Monitor* are similar to the widely adopted features in anti-virus software and firewall. *Guaranteed Protection* and *Privacy Presets* are based on the binary on-off mechanism, extending from the ad-hoc physical layer control. In contrast, participants were uncertain about the other concepts, such as how *Identity-based Data Management* handles images with multiple persons, who will enforce *Privacy Nutrition Labels*, what if developers lie to the *Privacy Simulator* platform.

More control. Another unanimous theme was that participants wished to have more control. Among the 11 technologies, six of them (#2, #4, #6, #7, #8, #9) offer some forms of control while the other five do not. Due to the isolation effect [382], whether the technology provides control options was rather salient when participants compared these two groups. Our coding process found that many participants explicitly mentioned the availability of control options in explaining their ranking rationale, and all of them viewed the option positively. For example, P255 stated that "*The (identity-based) data management actually allows*

Table 8.7: Summarized pros and cons of 11 privacy protection concepts through our thematic analysis. We ordered the concepts based on the global preferred order in Fig. 8.5.

#	Storyboard	Top pros	Top cons
8	Privacy diagnostics	Easy to use, can use it at will, can prevent data leaks by avoiding wrong configurations, can motivate users.	Need manual interaction, too late for already purchased devices
5	Data collection live monitor	Centralized interfaces & awareness, developers cannot fake network traffic	Offering no control, may forget to use it
4	Guaranteed protection	Can protect privacy even if developers are not trusted, helpful in a few sensitive conditions (e.g., bedroom sensors)	Too much efforts, prefer ad-hoc approaches if the plug is not trusted, limited binary control, not useful if developers are trusted
2	Privacy presets	Easy to use, simple and effective	Concerned about wrong configurations, need to trust the device developers
7	Privacy mirrors	Offering control to data on developers' cloud	Need to trust developers, may forget to use it
9	Contextual privacy reminders	No need for active interactions, offering control	Hard to generalize to other contexts
6	Identity-based data management	No need for active interactions, offering control, useful for people with roommates	Complex mental model regarding corner cases
10	Privacy simulator	Easy to understand the output, preventing from purchasing invasive products	Offering no control, need to trust developers, complex interfaces
11	Privacy nutrition labels	Preventing from purchasing invasive products	Hard to enforce and verify, too much information, too less information, offering no control
3	Privacy chatbot	Centralized and natural interfaces	Cannot prevent data leaks, may forget to use it, only useful for detailed questions
1	Anthropomorphism icons	Easy to understand	Offering no control, not enough information, better to use common icons

you to do something about your privacy; you can exclude certain people from accessing your info. The (data collection) monitor is nice in that you can see where your data is going, but it doesn't let you do anything about it. The (anthropomorphism) icons are just for people who aren't very tech savvy and don't really do anything." Another strong evidence is that none of the bottom four technologies (#10, #11, #3, #1) in the global ranking offers control options.

Proactive. Participants also expressed the desire for proactive technologies that do not need active user interaction. Past research shows that system proactivity, which refers to the degree of initiative a system might take based on its understanding of the context, is a vital dimension to explore for ubiquitous computing technologies [79]. Therefore, in designing the concepts, we intentionally covered a large variety of proactiveness: (1) *Identity-based Data Management* and *Contextual Privacy Reminder* (high proactive), (2) *Privacy Diagnostics* and *Privacy Presets* (medium proactive), and (3) *Data Collection Live Monitor*, *Guaranteed Protection*, *Privacy Chatbot* (low proactive). We found that all the participants appreciated the automation enabled by these proactive scenarios. More specifically, a few participants ranked the high proactive storyboards higher than the rest because they do not need active interactions. On the other hand, the few participants, who did not rank *Privacy Diagnostics* as the most preferred, complained that it requires some manual interactions. Finally, many participants criticized these low proactive solutions since they might forget to use them over time. For example, P280 criticized "*This (Privacy Chatbot) is only effective because Annie asked for the information. sometimes I might forget or not notice to do that until after the data had been sent.*"

Preventative. Participants preferred early prevention technologies to after-the-fact solutions because fixing devices that do not respect privacy is hard. We designed the storyboards to cover multiple stages in users' privacy protection. *Privacy Nutrition Labels* and *Privacy Simulator* protect users' privacy by improving their purchase decision-making. *Privacy Diagnostics*, *Data Collection Live Monitor*, *guaranteed protection*, and *anthropomorphism icons* aim to help users establish successful routines in managing their privacy. Finally, *privacy chatbot* and *privacy mirrors* are after-the-fact solutions, assisting participants in tracing the data leaks and fixing the issues.

We found that participants often mentioned the preventative property in comparing solutions across these categories, and they all viewed it positively. For example, P236 stated, "The last option (Privacy chatbot) is least effective in that it does not actively prevent the problem from happening; it only makes you aware of what has already happened." In analyzing these qualitative responses, we found it is because of the lack of solutions to fix devices with bad privacy practices. As a result, users either have to abandon the device or let the data leaks perpetuate. This negative opinion even applied to the most favored storyboard, *Privacy Diagnostics*. For example, P202 stated, "The Privacy Diagnostics seems somewhat unnecessary because most people will not buy a new replacement smart device just because it has a poor rating since most smart devices are quite expensive."

8.5 Limitations

Several limitations are important to mention. First, our samples were not representative of the general population. The population of Mechanical Turk workers is significantly less politically diverse, more educated, and younger compared to the US population [64]. Despite these limitations, past research suggests that online studies about privacy and security behavior can approximate behaviors of the wider population [311].

Second, participants may undervalue a few concepts that aim to manage smart home devices at scale. Most participants were using relatively few devices and had not hit many scale issues. Only 7 out of 386 participants have more than 20 devices, and relatively few participants expressed a need for scalability support (e.g., centralized user interfaces). While we included multiple concepts to cover the dimension of scalability (e.g., #1, #3, #5, #7), we found that few participants explicitly acknowledged the value of management scalability.

Third, social desirability may lead participants to over describe their SH-PPB experiences, especially the frequency of SH-PPBs. Therefore, we carefully avoided generalizing the results on frequency but only concluded that participants actively performed SH-PPBs. Asking participants to compare across a few reasonable promising privacy protection concepts also helped to alleviate this effect.

Fourth, the rankings of the 11 design concepts should not be used to discourage research in exploring these ideas. We used these concepts to elicit users' reactions, giving us more insights into the underlying problems, needs, and desires. But, these storyboards only cover limited usage contexts and design details, a minor change of these concepts may significantly improve users' preferences. For example, an important reason that users do not like *Privacy Nutrition Labels* is that they feel it would be hard to enforce and verify. A storyboard that states the law will enforce the labels may change users' preferences.

Finally, there may exist biases due to the genders of storyboards' protagonists [168]. We randomly assigned genders to the protagonists across storyboards, aiming to achieve a diverse set: five Male (#2, #5, #8, #10, #11), four Female (#1, #3, #7, #9) and two groups (#4, #6). The average rankings for female protagonists (average ranking μ: 2.11, standard deviation σ: 0.82) is slightly lower than male (μ: 1.92, σ: 0.80) and group (μ: 1.96, σ: 0.82). Future research may study the potential gender bias for using storyboards in UX experiments.

8.6 Discussion & Future work

8.6.1 The future of ad hoc privacy protection

We discuss three main limitations of ad hoc privacy protections in §8.4.2: they are inconvenient to change configurations, offer limited control granularity, and are hard to scale. However, there are also three

Table 8.8: The coverage of 11 storyboards across two principal dimensions: lifecycle and system proactivity.

Lifecycle	Proactivity (High)	Proactivity (Medium)	Proactivity (Low)
Install/purchase	#6 Identity-based Data Management	#10 Privacy Simulator	#11 Privacy Nutrition Labels
Routine	#7 Privacy Mirrors #9 Contextual Privacy Reminders	#5 Data Collection Live Monitor	#1 Anthropomorphism Icons #2 Privacy Presets
Deviate	#8 Privacy Diagnostics	#3 Privacy Chatbot	#4 Guaranteed Protection

important advantages of this ad hoc physical approach. First, it is cheap since participants do not need to purchase extra hardware or software. Second, it simplifies the trust model. For example, two participants expressed that they prefer manually unplugging devices to *Guaranteed Privacy*, since they do not need to purchase anything or trust the smart plugs. Finally, it has the simplest mental model. Although it can only offer limited functionality (e.g., binary on and off), our results suggested that users are very creative in appropriating simple technologies for various contexts. Therefore, we expect these physical ad hoc physical protections will coexist with specialized, built-in features and third-party features for a long time.

8.6.2 Building built-in and third party features

Our results call for more built-in and third-party privacy features for smart homes. However, implementing these features is challenging in today's ecosystem [216]. In contrast to online privacy protection, which has a few major points of leverage (e.g. web browsers, cookie management, etc), smart home users need features for the vast types of privacy-sensitive data that developers are collecting. Although each feature might be trivial to implement (e.g., turning off the microphone of a camera), developers require significant resources to implement the interfaces and control options for the heterogeneity of IoT devices. Further, smart homes lack a clear point of leverage to make third-party features work across multiple devices, similar to how Ad-blocker extensions help users manage privacy through the web browser. Future research should explore potential solutions to build these points of leverage, such as customizable DNS servers (e.g., Pi-Hole), smart home hubs (e.g., Peekaboo [181]), WiFi routers, and SNMP-like [385] protocols [181] for smart home devices.

8.6.3 The coverage of 11 storyboards

We used the IDEO brainstorming rules [171] to guide the storyboard generation & selection process. Three experienced privacy researchers first nominated individual ideas and then derived variations across two principal dimensions (similar to [79]): activity lifecycle (i.e., install/purchase, routine, and deviate) and system proactivity (i.e., high, medium, low). For example, an "install-time" *Privacy Chatbot* answers users' questions about the devices' potential behaviors before they make a purchase. By proactivity, we mean the degree of initiative that an intelligent system might take based on its contextual understanding. A "highly proactive" *Privacy Chatbot* monitors the network continuously and proactively asks users to verify these behaviors. After the process, we obtained over 50 candidates and merged them into 11 representative storyboards, ensuring a wide coverage for different ideas and dimensions (Table. 8.8). For example, we blended the "install-time" *Privacy Chatbot* into the *Privacy Nutrition Labels* (Fig. 10.23), and the "highly proactive" *Privacy Chatbot* into *Contextual Privacy Reminders* (Fig. 10.21). Besides, different concepts often introduce other dimensions, such as whether the feature offers control, serves multiple users.

8.6.4 Permuted combinations versus Factorial experiments

One alternative experiment design for the second survey is to run factorial experiments. For example, we may ask participants to rank *Privacy Chatbot* at different stages, such as purchase, install, routine, and when something goes wrong (deviate). But it has two limitations for our goal. First, the combinatorial explosion limits the number of test factors. Second, many enumerations may not necessarily be attractive.

In contrast, our ranking experiment differs in two key ways. First, we do not control the variables in each comparison task, but focus on the most promising concepts. Second, more than quantitative ranking, we also asked participants to explain their rationale in free text. In retrospect, at the core of our approach is the isolation effect [382]: when multiple homogeneous stimuli are presented, the stimulus that differs from the rest is more likely to be remembered. For example, when a participant ranks *Privacy Diagnostics*, *Data Collection Live Monitor*, and *Privacy Chatbot*, she may find that only *Privacy Diagnostics* can offer control and start to evaluate whether this is a positive factor. As we permuted the combination of storyboards in each comparison task, we implicitly guided participants to traverse the possible dimensions. This unique design allows us to compare more concepts than alternative approaches.

8.7 Conclusion

We conducted two online surveys to understand existing SH-PPBs and explore potential opportunities to better support users' needs of performing SH-PPBs. Our first study identifies 33 unique types of smart home privacy-protective behaviors, and finds that users heavily rely on protections at the physical layer and

on ad hoc approaches. Based on needs observed from our first study, we then speed dated 11 different privacy protection concepts and found that a relatively simple technology, *Privacy Diagnostics*, was preferred far above of the rest in our study. We also identify four important factors in designing SH-PPB tools: simple, preventative, proactive solutions, and more control.

Chapter 9

Conclusion

This dissertation introduces a new design pattern, called *Modular Privacy Flows* (MPF), for designing systems in allowing developers to collect data on a need-to-know basis. We conducted three empirical studies (Part 1) to characterize developers' data collection behaviors, illustrating that most developers only need partial or derived data rather than raw data. In part 2, We then introduce two MPF software architectures (Peekaboo and MapAggregate) that can reduce developers' data collection and demonstrate the many advantages. Finally, we present two design methods (Part 3) to help developers navigate data minimization's design space, including data collection decision-making and designing independent privacy features through MPF. Combined, this dissertation will scaffold the future development of data minimization.

In this last chapter, we discuss how MPF can be adopted, summarize a few lessons that influenced this work, and sketch areas for future work.

9.1 MPF adoption

Value propositions for different stakeholders. MPF offers a few important benefits to stakeholders in the privacy ecosystem (Table. 9.1).

1. **Transparent and enforceable data collection**. For each data collection, developers have to explicitly declare data transformations before the data leaves users' control in the manifest, and only through a declared chain of operators that specify how to transform the data, thus making the data collection process more transparent to users (a.1) and auditors (c.1). The improved transparency also helps developers gain privacy trust from users, which is especially beneficial for small developers (b.1).
2. **"Need-to-know" data collection**. Instead of collecting users' data aggressively for future unanticipated data usages, developers now can collect users' data on a need-to-know basis without sacrificing development flexibility (a.2). In case a new data usage demand occurs, developers can update the manifest with users' approval. Further, privacy advocates (e.g., Consumer Reports) can compare data

collection behaviors across similar applications and find the ones better respect users' privacy (d.1).

3. **Structured information flows**. Since the fixed set of operators have clearly defined semantics, third parties can analyze and rewrite the information flow in ways that are not easily possible if it were arbitrary code. So privacy advocates can leverage this structured nature to offer users a set of privacy features across apps and devices, without additional effort from app developers (d.2). For example, a static analyzer can automatically transform a manifest into natural language statements to make it easier for laypeople to understand what data will be sent out, when, and to where. These independent features can help users manage their privacy in a unified and centralized manner and guarantee a fine-grained control even if the developers do not offer one (a.4). Meanwhile, developers can save labor in building common privacy features (b.3). Privacy advocates and auditors can easily audit developers' data collection behaviors and scale the analysis through automation (c.1, d.3).
4. **Other side effects**. MPF is designed to only factor out the non-proprietary data transformations into open-source operators. Developers' proprietary software remains on their private servers, so they can still offer their services using a subscription-based business model (b.2). This design also has a positive side effect of saving developers' computation and bandwidth (b.5) and development labor in implementing repetitive data transformation tasks (b.4).

Table 9.1: Value propositions for different stakeholders

Stakeholder	Value propositions
End users	a.1 More transparent data collection a.2 Reducing unnecessary data egress a.3 Unified and centralized privacy management a.4 Built-in fine-grained controls even if the developers do not offer one
Developers	b.1 Easy to obtain users' privacy trust b.2 Keeping the proprietary nature & supporting subscription-based business models b.3 Saving labors in building common privacy features b.4 Saving labors in implementing repetitive data transformation tasks b.5 Saving computation and bandwidth with development flexibility
Auditors	c.1 Easy to audit data collection behaviors and verify developers' claims
Privacy advocates	d.1 Allowing the good to drive out the bad d.2 Possible to build privacy features without additional effort from app developers d.3 Easy to scale the analysis

Adoption barriers. The most significant barrier to MPF adoption is to convince developers to collect data on a need-to-know basis. Instead of letting developers collect raw data on their server and use it at their discretion (i.e., purpose late-binding), MPF wants to push developers to declare how they will use the data

and only collect necessary data (i.e., purpose early-binding).

At first glance, developers may not like the idea of purpose early-binding since it is hard to foresee the potential data usage at the time of software implementation. The most crucial step to overcoming this barrier is to make the purpose early-binding as flexible as possible. In other words, systems need to make it really easy for developers to update their data collection manifests.

Use the "HelloVisitor" app in Peekaboo as an example, which identifies visitors using faces in images. Imagine that the developer wants to enhance their visitor recognition algorithms by leveraging visitors' outfits, so it needs the portions of images with detected persons rather than only detected faces. In this case, the hub should check the manifest using automated programs and install it automatically without informing users. Instead of manually inspecting and approving each data request, like what we have in Android/iOS today, users can review these activities collectively using built-in privacy features and rely on trusted third parties (e.g., consumer reports) to identify wrongdoers.

What if adopted? Once adopted, this design pattern can potentially create a new type of privacy ecosystem. Regulators like FTC, and GDPR, can easily find bad privacy practices since the data collection behavior is transparent. Privacy advocates can rank app developers because analyzing the manifests at scale is easy. They can also build independent privacy features by rewriting the manifests. Small developers can save efforts in implementing common privacy features since the protocol can come with many built-in features. All developers can gain users' trust. Today, users do not trust devices like Alexa. But now, the architecture can enforce the data collection behaviors. Finally, users can manage their privacy in a centralized and unified manner. In addition, the hub can help enforce users' controls. Hopefully, through all the benefits enabled by the new system primitive, products with good privacy can drive out the ones with bad privacy.

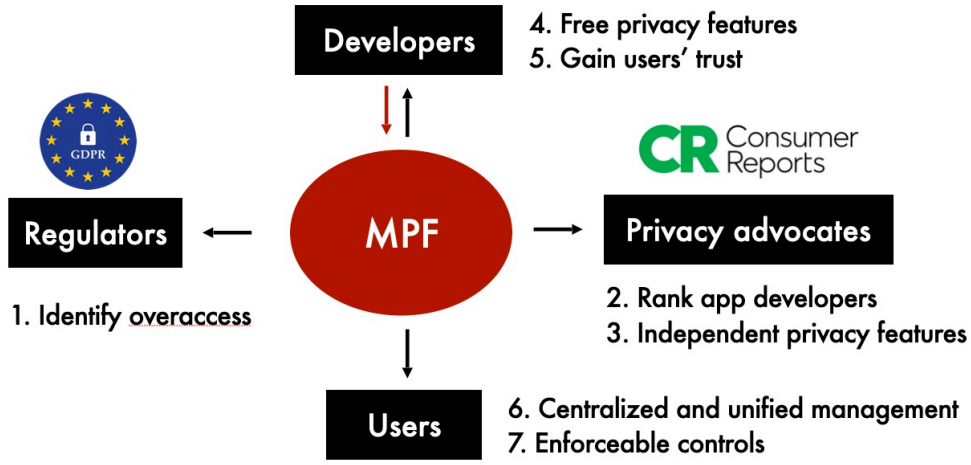


Figure 9.1: Once adopted, MPF can enable a new type of privacy ecosystem, making it easier for users, developers, regulators, and privacy advocates to protect users' privacy.

What if not adopted? We believe that MPF can create a win-win solution for all the stakeholders, enabling a virtuous cycle ecosystem where building trustworthy systems is rewarded, and developers compete to guarantee greater user protection, not less.

In contrast, if we do not push for solutions like MPF, data privacy can potentially be a lose-lose game for both users and developers. On the one hand, users are afraid that they no longer control the data once it leaves their homes, so they do not want to adopt new technologies like Alexa devices. Meanwhile, we will see more independent tools like Pi-hole [161] and Ad-blocker, reducing the quality of collected data. But on the other hand, developers may collect data even more aggressively to mitigate the noise.

9.2 Lessons learned

Generalization of MPF. We characterize the MPF design pattern as a combination of three simple ideas: (1) system builders first introduce a small set of stateless verb operators; (2) developers then chain these operators to implement data actions (i.e., data pre-processing and data aggregation); (3) finally, system runtimes isolate the actual code implementation from the manifest declaration, so developers can only access users' data in the way they declare in the manifest.

Peekaboo contributes an example operator design for implementing data pre-processing and designs an example runtime using an in-home hub. MapAggregate contributes another example operator design for implementing data aggregation and designs an example runtime using serverless cloud infrastructure. However, there exist many alternative ways to implement MPF. For example, an alternative design is to run the Peekaboo runtime in a personal private cloud. Or future work may consider deriving operators for other privacy-related data actions, such as data noisification and data retention. Most benefits mentioned in this dissertation should still hold.

Privacy guarantee. MPF, as a design pattern, does not offer any privacy guarantee. However, MPF systems can achieve some privacy guarantee through the data transformation they provide through the operators. For example, the `Aggregate` operator in MapAggregate can offer a privacy guarantee by only disclosing aggregated data. Future research may create operators to support more sophisticated privacy enhancing technologies, such as differential privacy (DP), federated learning (FL), multiparty computation (MPC).

When should we (or not) use MPF? MPF in its current implementations (i.e., Peekaboo & MapAggregate), is most helpful in designing accesses to allow third-party developers to access users' daily data (e.g., streams from video doorbells). In contrast, system builders should consider other approaches, such as Federated learning, multiparty computation, and differential privacy, when designing accesses for highly sensitive data (e.g., medical data). Here we enumerate a few important considerations here.

1. **Data types**. Much of Users' daily data often are high-dimensional, such as video, audio, image, and time series, while most highly-sensitive data, such as medical/bank data, are often tabular. Developing generalizable and efficient DP/FL/MPC for high-dimension data remains an open problem. In contrast, we can easily generalize MPF for most data types.
2. **Developer accountability**. Medical/bank data are only open to a limited number of accountable partners. For example, there are only 4,236 FDIC-insured commercial banks and 6,093 hospitals in the U.S. in 2022. In contrast, users' data can potentially be accessed by millions of less accountable developers. For example, there are 5.9 million Android developers and 2.8 million iOS developers [173]. As a result, while it is possible to audit these banks and hospitals to verify that they put enough effort into protecting users' privacy, it is not feasible to audit millions of app developers who are only associated with email addresses. MPF allows the system builders and third-party privacy advocates to audit developers at scale. Further, these millions of app developers will not have enough resources to build sophisticated DP/FL/MPC solutions, while authoring an MPF requires much fewer resources.
3. **Data sensitivity**. Medical/bank data can potentially be privacy sensitive even after data transformation. So only using data minimization can potentially be insufficient for these usages. Instead, users' daily data, such as web browsing history, are less privacy-sensitive than medical data and can enable many different applications. While data minimization may not offer perfect privacy protection, it is much easier to balance the utility of ubiquitous computing applications and users' privacy protection.

MPF v.s. REpresentational State Transfer (REST). Over the past two decades, REST has become the standard for designing web APIs [352]. REST uses the standard CRUD HTTP Verbs (GET, POST, PUT, DELETE) to access server data resources (HTTP URIs). However, one limitation of REST APIs has shown to be too inflexible to keep up with the rapidly changing requirements of the clients that access them. Recently, technologies like GraphQL were developed to cope with this limitation. GraphQL organizes data into a graph and uses a resolver to allow clients to adjust their queries without changing the server APIs.

In contrast, MPF demonstrates a new way to address the limitation of REST APIs, which expands the standard CRUD verbs into a set of stateless verb operators that serverside developers can implement. As described in the introduction, MPF can be more flexible and verifiable than the GraphQL approach since developers can customize these operators. Future research and industry practitioners may look into using MPF beyond the context of privacy, studying whether MPF can potentially improve development flexibility and efficiency.

9.3 Future work

Modular Privacy Flows also enables a few new research directions.

Manifests as Interactive Privacy Policies. Traditionally, we have machine-readable implementations like

C++ and Java. However, because they are not comprehensible and accessible to end-users and auditors, developers have to manually create some natural language policies for outsiders. This conventional paradigm creates a few challenges around privacy.

- Many developers still do not offer privacy notices/policies/features.
- Developers may lie in the policies/privacy interfaces.
- Developers' policies may be vague and hard to understand for users.
- Developers may only offer all-or-nothing policies.
- Users need to interact with 100+ services. The have no time to manage these privacy policies.

An important feature of MPF's manifest is that it can be a new type of program that bridges the human-machine divide. In contrast to these arbitrary programs, MPF has two unique features: (1) the manifest only uses a small and fixed set of operators with well-defined semantics, and (2) all these operators are stateless. These two features make it easy to analyze and rewrite these data transformation behaviors using these manifests. In Peekaboo, we build a simple proof-of-concept feature that generates privacy nutrition labels automatically using any arbitrary manifest. However, future research may seek to build more privacy features and study how these interactive manifests impact users' mental model.

Broader application domains. In this dissertation, we chose to apply MPF to smart home and smart city contexts for two reasons. First, users' privacy will suffer more in these ubiquitous computing environments since these data are more privacy intrusive than web browsing data and mobile data. Second, there is no standard in these contexts yet, making it more likely to have MPF adopted. However, we can potentially apply MPF to many other contexts. Here, we enumerate a few common examples.

1. **Browser extension**. Browser extensions are much more dangerous than most people realize [160]. Extensions often have access to everything users do online, such as passwords, web browsing histories, and more. Future browsers may consider adopting MPF and require all developers to submit a manifest to access users' data.
2. **HTTP cookie**. Targeted ads per se does not seem fundamentally evil, unless you think putting car ads in car magazines is also evil [116]. We believe what makes users uncomfortable is that the ad tech industry tracks users' online activity, inferring many insights that users are unaware of and using these insights in unknown contexts. One alternative solution is that browsers may allow advertisers to access users' attributes through MPF manifests. For example, browsers may introduce a set of operators which can compute users' interests using their local browsing history. Instead of tracking users using unique IDs, advertisers can personalize the ad by querying the attributes computed locally.
3. **Personal data store**. The zoom-google-calendar data access (Chapter 1 §1.2) is an example of personal data store. Besides, we can apply a similar MPF paradigm to email and wearable health data.
4. **Mobile apps**. Mobile apps are experiencing an explosion of permissions. Use the mobile location data as an example. There are three different location-relevant permission dimensions: category

(background or foreground), accuracy (fine location or coarse location), and user choices ("while using the app" v.s. "only this time" v.s. "deny"). Further, developers may infer users' location information through network data (e.g., IP address, WiFi Mac address). Applying MPF to mobile apps can have two significant benefits. First, system builders can implement a small set of reusable operators to compose many data permissions. Second, these manifests can enhance the privacy protection of many data resources beyond GPS sensors. For example, instead of always sending developers the raw IP, a manifest may preprocess the IP address into a more privacy-friendly format.

5. **Social network apps**. The Facebook / Cambridge Analytica data scandal is another data access design fiasco. While the initial app "thisismydigitallife" built by Aleksandr Kogan only had 270 thousand participants, the app accessed the information of each participant's friends, harvesting data from up to 87 million Facebook users [52]. Future social network apps may adopt MPF in designing data access for third parties.

Principle of Least Privilege. This dissertation focuses on using MPF to tackle privacy challenges, which are mainly about "read" access. MPF can potentially be generalized as a design pattern for the principle of least privilege, handling access problems in "write" and "actuation" as well. For example, Casolare [50] found that Android apps can establish intercommunication through SharedPreferences. One potential fix is to require developers to write/read SharedPreferences through a MPF-like manifest.

Moving forward, we may generalize the idea of manifests to many other entities, such as hardware components, software packages, and online chat, whitelisting each entity's interactions with the rest of the world in a common, structured, useful, and understandable way. One example is the diesel gate: Volkswagen had intentionally programmed their diesel engines to activate their emissions controls only during the lab test. MPF, which only uses stateless verb operators to author data actions (e.g., data pre-processing, data aggregation), would be insufficient to represent these types of interactions. Future research may explore new types of modular manifests using non-verb operators.

Chapter 10

Appendix

10.1 Appendix: Data Action Analysis Work Sheet for Lean Privacy Review

10.1.1 Overview

This worksheet aims to help a data practitioner construct a privacy storyboard based on a data practice he/she has in mind. A privacy storyboard contains a set of data actions organized in a tree topology. We provide definitions, instructions, and examples below.

10.1.2 An example data practice

We illustrate the procedure of privacy storyboarding through an example data practice, derived from "Is your pregnancy app sharing your intimate data with your boss?" published in The Washington Post in April 2019.[1]

We offer a summary of the news report below:

The period- and pregnancy-tracking app Ovia helps users track their pregnancy journeys. Employers who pay the apps' developer can offer their workers a special version of the apps that relays their health data —in a "de-identified," aggregated form —to an internal employer website accessible by human resources personnel. The companies offer it alongside other health benefits and incentivize workers to input as much about their bodies as they can, saying the data can help the companies minimize health-care spending, discover medical problems, and better plan for the months ahead. However, experts worry that the employers could identify women based on information relayed in confidence, particularly in workplaces where few women are pregnant at any given time.

[1] See https://www.washingtonpost.com/technology/2019/04/10/tracking-your-pregnancy-an-app-may-be-more-public-than-you-think/?arc404=true for the news report.

10.1.3 Instructions

Step 1(a). Enumerate the potential outcomes that will be generated by the data practice.

The news report describes two possible outcomes (data applications).

1. Employers can use the information to determine when and how many female employees will take maternity leave, which can then be used to plan for openings and health insurance.
2. Although the pregnancy related data is anonymized, the employer may still be able to identify the specific female employee through other information.

In practice, it might be challenging to enumerate every possible outcome in one pass; the focus here should be on the most possible and common cases (both positive and negative). You may revisit this step later and add new outcomes incrementally.

Step 1(b). Identify the data stakeholders for each data application. A data stakeholder is an individual or group that could affect or be affected by the data practice.

Table 10.1: There can be three types of data stakeholders: Data subjects, Data observers, and Data beneficiaries/victims. From the news report above, we identify the following stakeholders.

Primitives	Definitions	Examples
Data subject	the people who contribute their data	Female employees
Data observer	the entities that have access to people ' s data	the App developer, the employers
Data beneficiaries/victims	the people impacted by a data practice	the employees, the app developer, the employers

In practice, you may identify more outcomes after enumerating the stakeholders. You may revisit Step 1(a) to iterate the storyboard.

Step 2(a). Identify the data actions in a data practice and organize them in a tree typology.

Table 10.2: A data action is the smallest unit in a LPR story, which describes a specific operation that consumer businesses interact with users ' data. We used four types of data action primitives: 1) Data collection, 2) Data sharing, 3) Data processing and 4) Data usage.

Primitives	Definitions
Data collection	A data observer collects/stores data from data subjects.
Data processing	A data observer processes users' data to derive new data.
Data sharing	A data observer shares users' data or derived data with another data stakeholder (i.e., observer, subject and beneficiary/victim).
Data usage	A data observer uses the data in a certain way that impacts a data beneficiary/victim.

We identify the following data actions from the news story above.

1. Data collection: A pregnancy app collects data from their uses.
2. Data processing: The app processes users' behavior data and generates some statistics.
3. Data sharing: The app shares the data with many employers.
4. Data usage: Employers use the data to determine healthcare spending.
5. Data usage: Employers use the data to identify female employees who are pregnant.

Step 2(b). Organize the flow of the data actions.

We organize the two data applications as the following two sequences with shared data actions.

Application 1: (1) -> (2) -> (3) -> (4)

Application 2: (1) -> (2) -> (3) -> (5)

Step 3. Describe each data action in succinct text.

The last step is to generate a text description for each data action, explaining the details of a data action to the non-tech-savvy audience. An ideal description should contain all the necessary privacy-salient information (e.g., how the data is being collected, who it is being shared with, what information is being derived, how the data is being used, etc.) while being succinct.

You may think each data action through the following questions and use a lightweight description structure: [Context] [Data action][Further description - optional].

Questions for data collection.

- **Context**: Who is the data observer? What is the data subject? What data is being collected/stored? Why is the company collecting/storing the data?
- **Data action**: How is the data being collected?
- **Further description**: Is the data collection anonymous/pseudo-anonymous/non-anonymous? Do the users receive an explicit notification? How will the company keep the data? When will they delete the data? Do users receive compensation?

Data action (1): A pregnancy app collects data from their users.

Detailed description: [Context] A pregnancy-tracking app provides health-care aid for women to understand their bodies better. Users can log in to record their bodily function, e.g., sex drive, medications, and mood. The app has a "fertility algorithms", which analyze the users' menstrual data and suggest good times to try to conceive. [Data action] Your company establishes a partnership with the app developer. [Further description] Your company pays each employee $1 a day in gift card if she regularly uses the app.

Questions for data processing.

- **Context**: Who is the observer? What is the raw data? What is the derived data?

- **Data action**: How is the data being processed?
- **Further description**: Is the processing anonymized? Who did the processing? Algorithms or humans? How is the processing method developed?

Data action (2): The app processes users' health data and generate related statistics.

Detailed description: The pregnancy-tracking app aggregates the data of employees in the same company and removes personally identifiable information (e.g., name, email, age).

Questions for data sharing.

- **Context**: Who is the sender? Who is the recipient? Why does the sender share the data?
- **Data action**: How is the data being shared?
- **Further description**: Is the data sharing for profit? Is the process secure?

Data action (3): The app shares the data with many employers.

Detailed description: The employer pays the app developer to get their employees' aggregated data and related statistics, e.g., how many workers using the app had faced high-risk pregnancies or had given birth prematurely; how soon the new moms planned to return to work.

Questions for data usage.

- **Context**: Who is the observer? Who is the data beneficiary/victim?
- **Data action**: What are the potential risks (i.e., the probability and impact) for impacted users?
- **Further description**: Are users aware of such data usage? Does the company have users' informed consent? Does the company give users compensation?

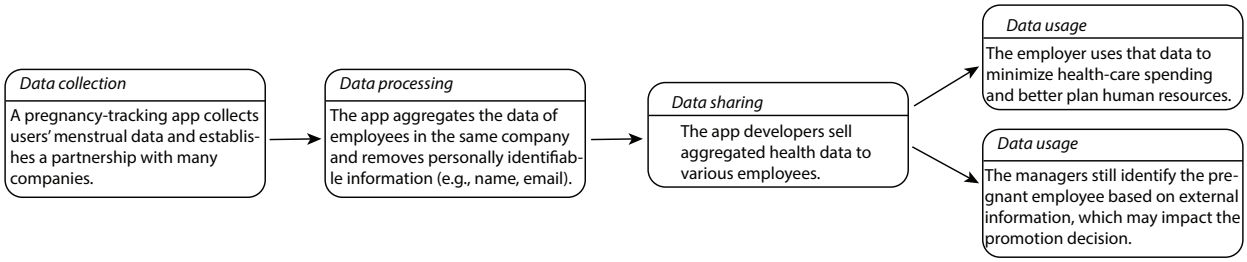
Data action (4): Employers use the data to determine healthcare spending.

Detailed description: The employer plans to use the data to minimize health-care spending, discover medical problems, and better plan human resources for the months ahead.

Data action (5): Employers use the data to identify female employees who are pregnant.

Detailed description: The employer can still identify the employee based on information relayed in confidence, particularly in workplaces where few women are pregnant at any given time. These intimate information may impact that employee's promotion.

Here, we generate the final version of privacy storyboard for this data practice.



10.2 Appendix: 12 real-world privacy stories in Lean Privacy Review and their privacy storyboards

Figure 10.1-10.12 illustrate the privacy storyboards of 12 real-world data practices. We only offer a brief text summary for each data action in these illustrations. The complete textual description is available on the accompanying website.

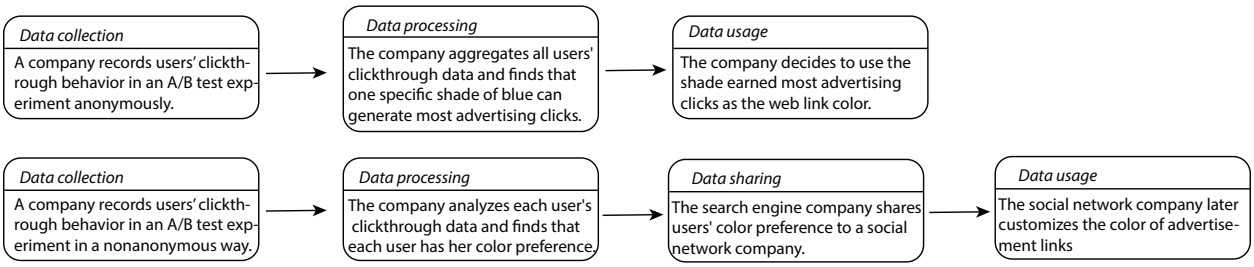


Figure 10.1: A privacy storyboard of "#1 Search engine clickthrough data". A company records users' clickthrough behavior in an A/B test experiment anonymously and uses the data for advertising and search personalization.

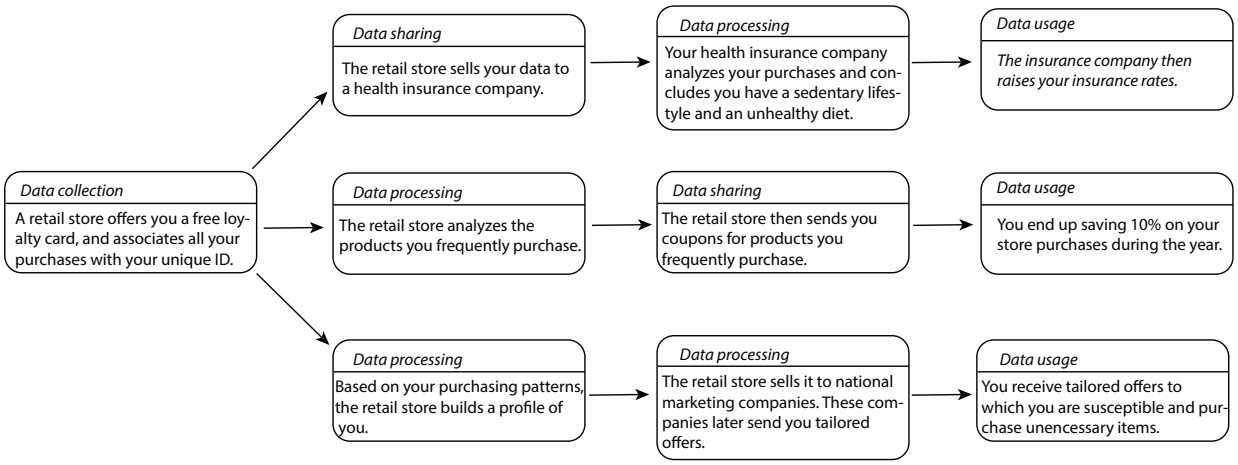


Figure 10.2: A privacy storyboard of "#2 Loyalty card in a retail store". A retail store collects users' data through a loyalty card and uses the data for insurance and coupon personalization.

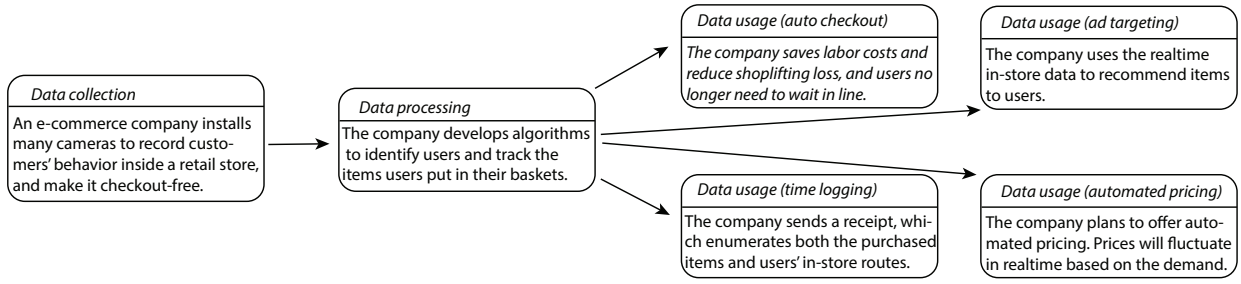


Figure 10.3: A privacy storyboard of "#3 checkout free retail store". An e-commerce company opens a checkout-free retail store by installing various sensors inside a physical store.

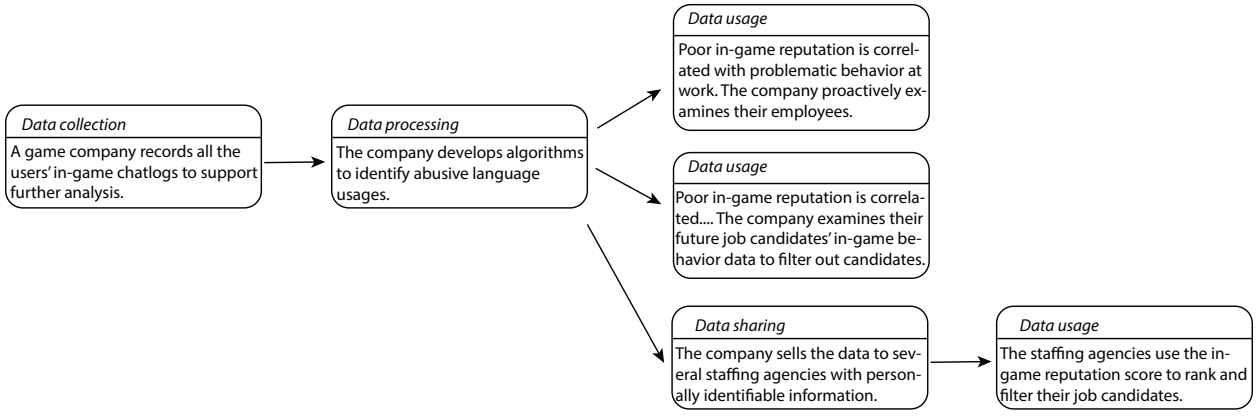


Figure 10.4: A privacy storyboard of "#4 game chat log". An online game company uses chatlogs to identify potential problems in the workspace.

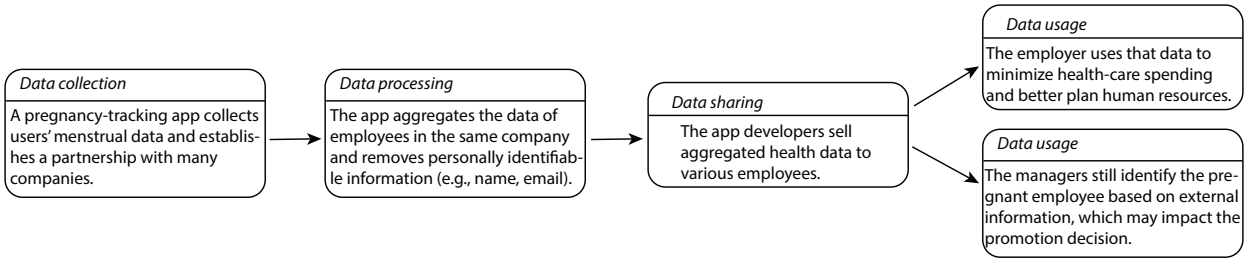


Figure 10.5: A privacy storyboard of "#5 pregnancy intimate data". A pregnancy app shares users' intimate body data with their employers.

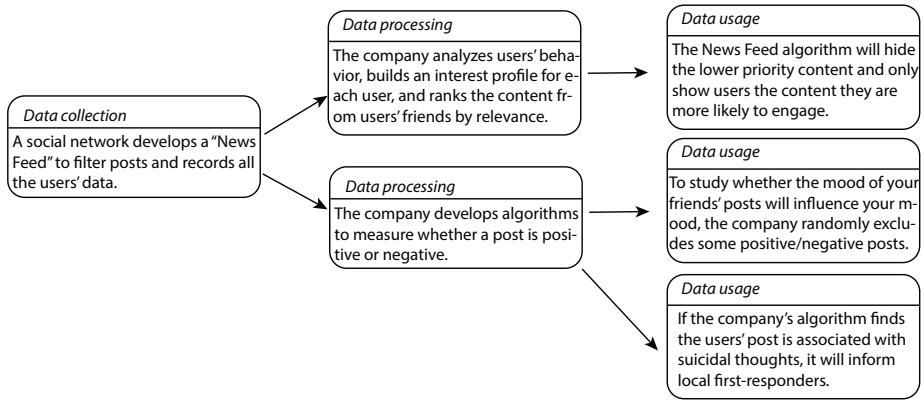


Figure 10.6: A privacy storyboard of "#6 social network". A social networking service company analyzes users' posts through sentiment analysis and uses insights in different ways.

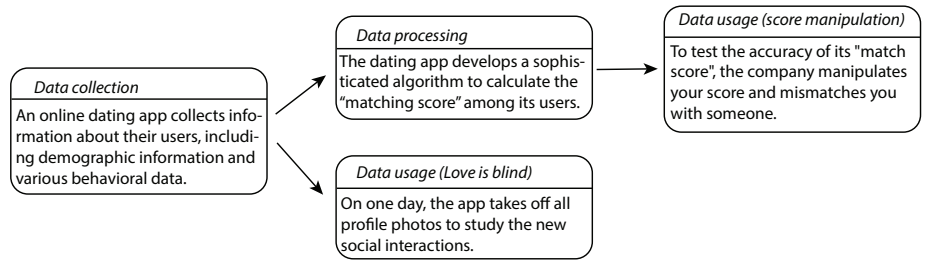


Figure 10.7: A privacy storyboard of "#7 data science experiments in a dating app". An online dating app conducts several experiments to understand the nature of romantics.

Figure 10.8: A privacy storyboard of "#8 Email contacts for social network bootstrapping". A technology company appropriates users' email data to bootstrap a new social network service.

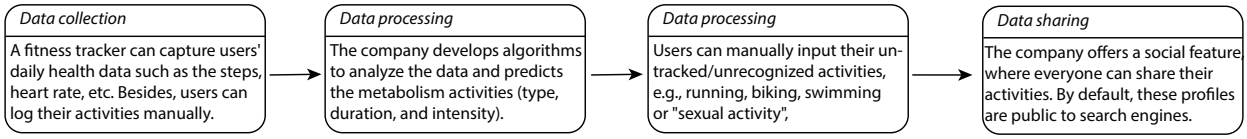


Figure 10.9: A privacy storyboard of "#9 Fitness tracking". A wearable technology company collects users' intimate behavior data and makes them public by default.

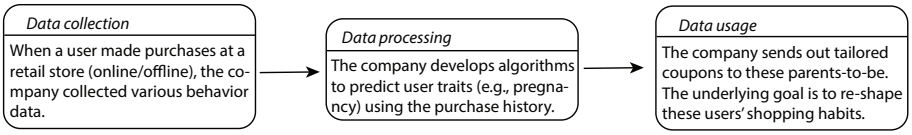


Figure 10.10: A privacy storyboard of "#10 retail store pregnancy". A retail store predicts users' pregnancy status by analyzing their purchase history.

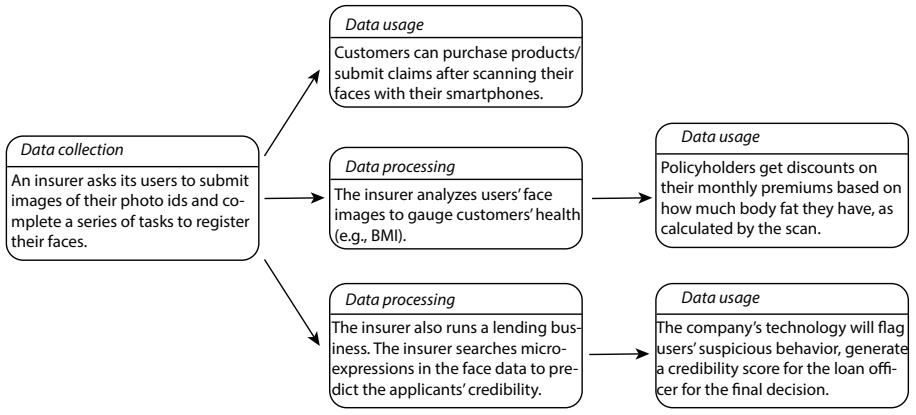


Figure 10.11: A privacy storyboard of "#11 an insurer employs AI". An insurance company uses facial-recognition technology to identify untrustworthy and unprofitable customers.

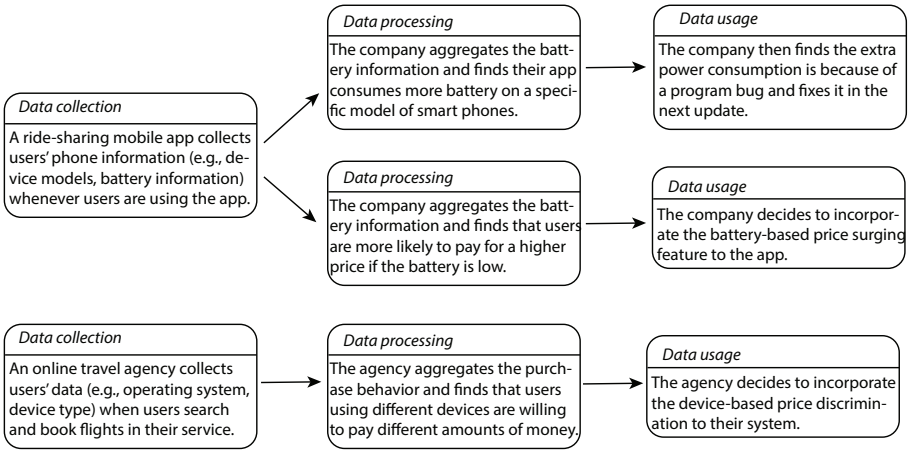


Figure 10.12: A privacy storyboard of "#12 dynamic pricing". Technology companies collect users' behavior data to adjust the service price dynamically.

10.3 Appendix: The distribution of responses across different scores for individual scenarios in Lean Privacy Review Experiments

10.4 Appendix: Complete LPR Privacy concern taxonomy

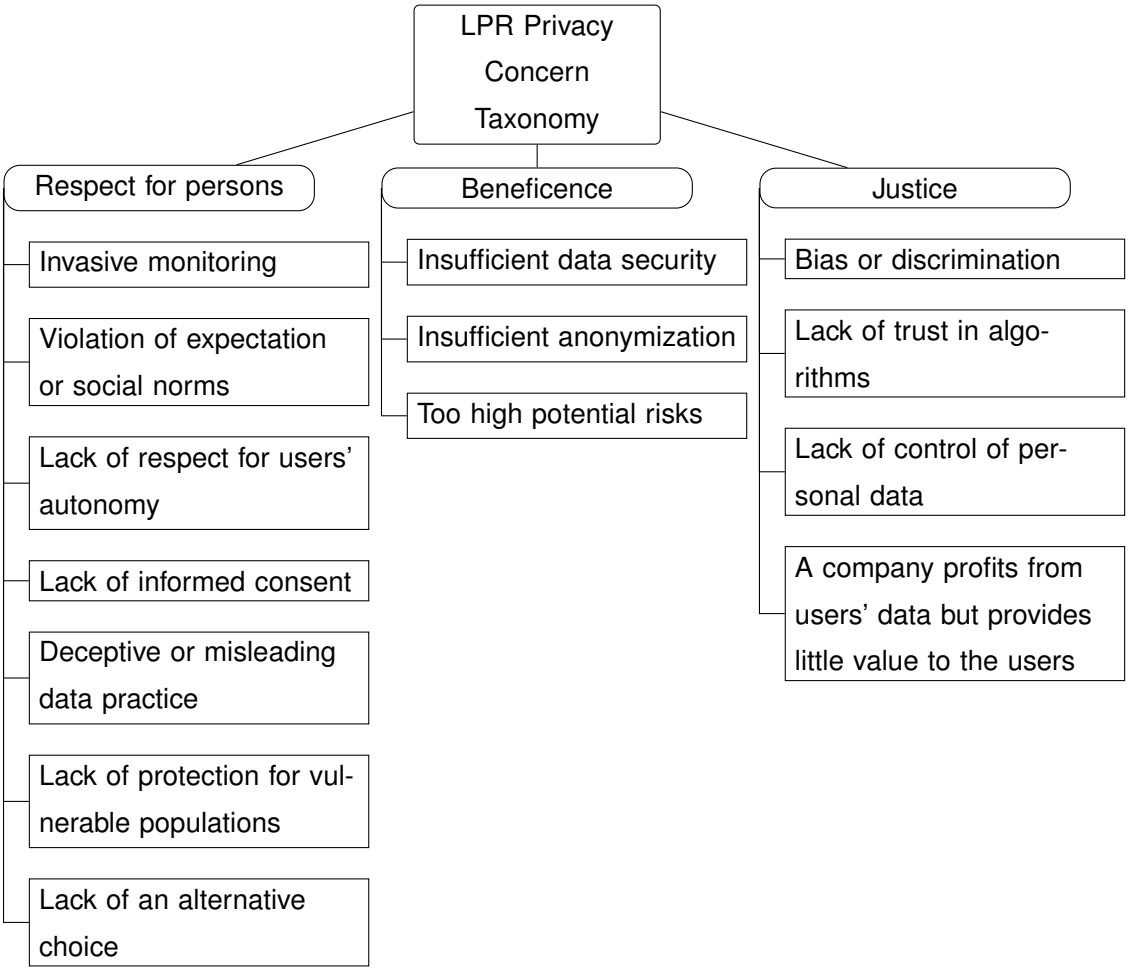


Figure 10.14: The overview of LPR privacy concern taxonomy. We used a ground theory approach to build the list of common privacy concerns, which is the basic vocabulary to characterize users' free-text privacy responses. To help practitioners comprehend the list quickly, we further organized these privacy concern types into three high-level categories: respect for persons, beneficence, and justice.

Table 10.3: The privacy concerns (PC) under *Respect for persons*. Respect for persons: the company should respect users' dignity, autonomy, and provides special protection to vulnerable populations. The example quotes are from crowd workers' survey responses.

PC category	Example subcategories	Example quotes
Invasive monitoring	Too much data	I feel uncomfortable because I am being watched so closely.
	Too sensitive data	I don't like the idea of my fertility data being used in any way.
Violation of expectations/social norms	Unexpected data sharing	No one should be sharing this with other companies unless there was illegal activity going on.
	Unexpected data collection	I don't think the company should have any information like that.
	Unexpected data appropriation	I don't think that how someone acts in a game should impact their ability to get a job.
Lack of respect for autonomy	Decisional interference	Marketers are using this information to get me to click on their ads, and it may not be the product that draws my attention but rather the color.
	Users prefer self-control than data-driven automation	Just let me select the item and pay for it. I don't need these false conveniences.
Lack of informed consent	Lack of transparency	The app is now operating in hidden ways that are not explained to the user.
	Lack of consent	I don't like knowing they are using psychological tactics to gain profit from me based on info they acquired without my consent.
	Violation of existing consent	I don't think that I agree to share my data.

Deceptive or misleading data practice		I feel uncomfortable because the company is misleading me, and that goes against the tacit understanding I had when agreeing to use the app.
Lack of protection for vulnerable populations		I think the company should not send pregnancy product ads to high school girls.
		I feel targeting ads at sad/anxious teenagers is unethical.
Lack of an alternative choice.	No opt-out option	I feel uncomfortable because I cannot choose not to participate.

Table 10.4: The privacy concerns (PC) under *Beneficence*. Beneficence: the benefits of a data practice for users should justify the potential risks/costs. The example quotes are from crowd workers' survey responses.

PC category	Example subcategories	Example quotes
Insufficient data security	Potential data breach	I feel uncomfortable because a data breach may occur.
Insufficient anonymization	Insufficient anonymization	Because my company is small, so it wouldn't be very anonymous even without identifiable information.
Too high potential risks	Risk of financial loss	I don't trust that the company won't just use the data to further cut their own costs whenever possible and further screw over their own employees with less benefits, etc.
	Risk of opportunity loss	I don't want my private sensitive health information revealed to anyone really, especially my employers, when it may impact employee promotions or other decisions.
	Risk of reputation loss	

Table 10.5: The privacy concerns (PC) under *Justice*. Justice: each user deserves equal and fair treatment in the given data practice. The example quotes are from crowd workers' survey responses.

PC category	Example subcategories	Example quotes
Bias or discrimination		The idea of fluctuating pricing seem somewhat unethical and possibly illegal. Everyone should be charged the same.
Lack of trust for algorithms	Concerns on the imperfect implementation	I would not trust their algorithms.
	Concerns on algorithmic automation	I don't like the idea of a store getting ideas about me based solely on what I buy. I think I'm more dimensional than my purchases alone may suggest.
Lack of control of personal data		I have no way to control the data after sharing the data.
A company profits from users' data but provides little value to the users (i.e., data commodification)		The company will use this data to profit off my shopping and buying habits. I do not see any benefit of this save for additional targeted marketing.
		I feel uncomfortable because I don't want my clicks tracked for the purpose of making a company more money off of me through ads.

10.5 Appendix: Storyboards for Privacy Speed Dating

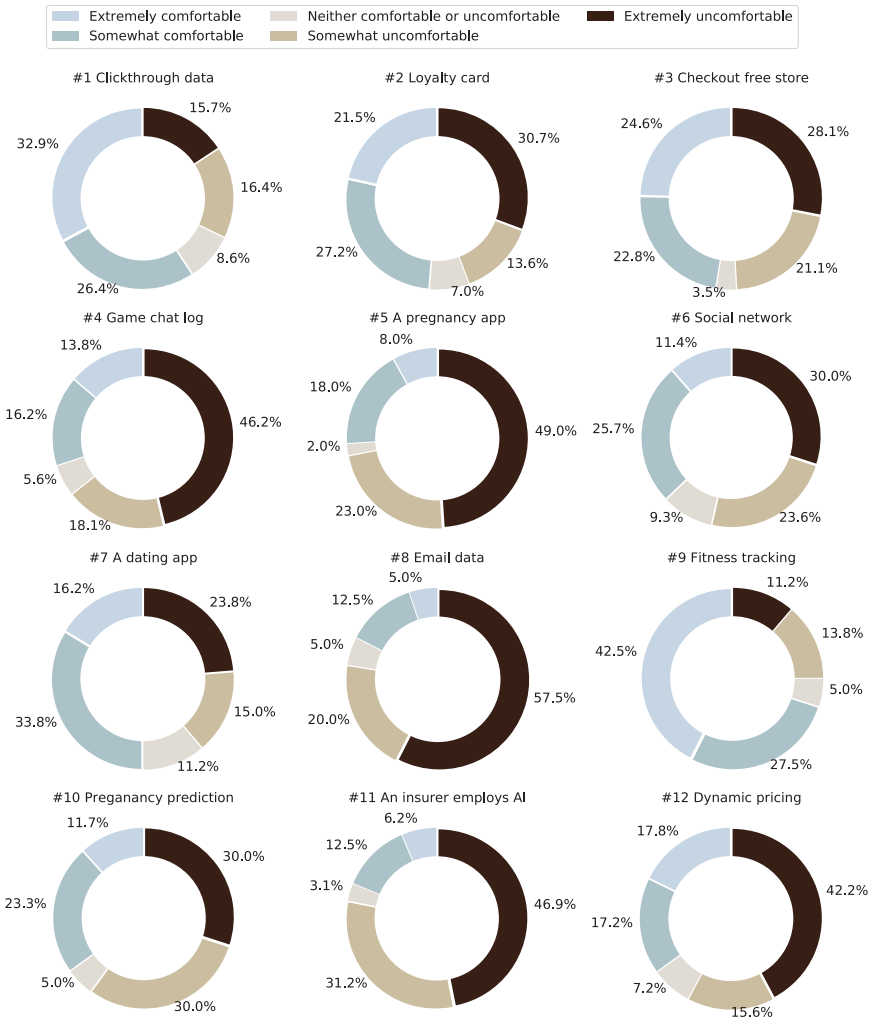


Figure 10.13: The individual distribution of responses in each scenario across different scores. At a glance, the results offer many surprising results. For example, the seemingly less severed data practice, Google Buzz data appropriation [156, 266] (i.e., "#8 Email data"), received the most negative feedback. At the same time, the highly cited emotion contagion experiment [70, 199, 213, 312] (i.e., "#6 Social network") is less criticized by the crowd workers. Our focus of this paper is to introduce the LPR framework; we will present the findings from our data in a separate document.

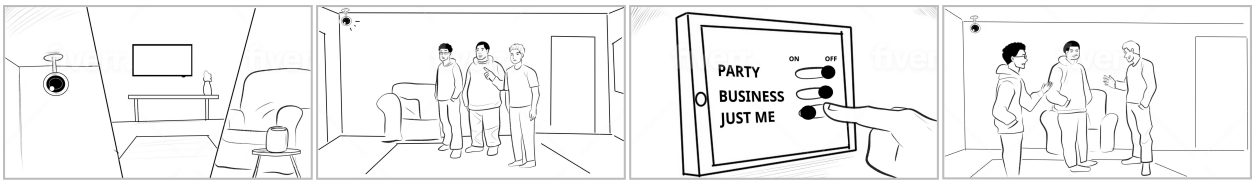
[Character, Context]
Mary has several smart home devices at her house. Some of them are bought by her husband, who seems tech-savvier; others are gifts from her friends and relatives.

[Problem]
Though she always wonders how and why these devices work so smartly, she doesn't want to read through those lengthy product descriptions.

[User Action, Technology]
Mary installs a central hub display that shows all her installed devices in a spreadsheet. Next to each device name, a robot avatar indicates the types of data each device can collect. For example, if the device has a microphone, the avatar would have two ears.

[Resolution]
Mary now has a better understanding of her devices. Whenever she wants to know how the devices work, she can refer to the central hub.

Figure 10.15: Storyboard #1 Anthropomorphism Icons



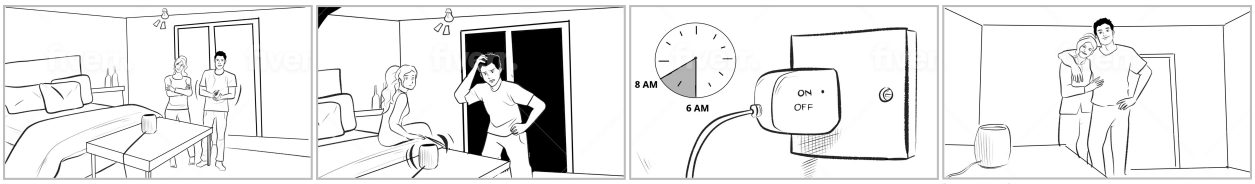
[Character, Context]
Carl lives in a house with many smart home devices. These devices keep running continuously throughout the day.

[Problem]
Carl works at home a lot, and he does not want these devices to monitor his business phone calls. Occasionally, Carl invites guests to parties at his place. His party mates also feel uncomfortable about being recorded.

[User Action, Technology]
He installs a mode switch sets up a few privacy modes to manage all the devices. For example, when hosting a party, he can switch to the "Party" Mode that stops recording the data of his guests. When working at home, he can switch to the "Business" mode that turns off all devices' microphones.

[Resolution]
Carl can now easily accommodate his and his guests' privacy with one mode switch and be assured that everyone's privacy need is met.

Figure 10.16: Storyboard #2 Privacy Presets



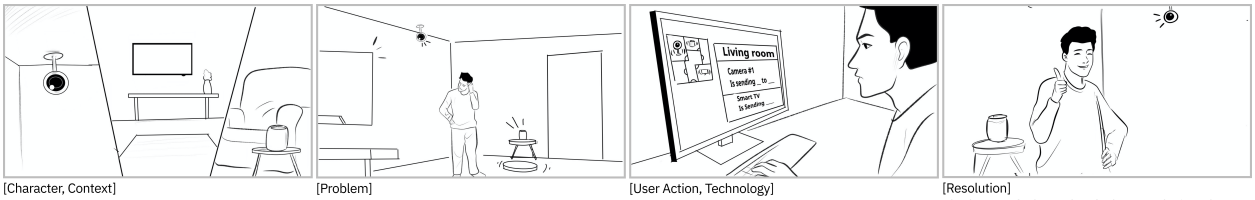
[Character, Context]
Charley and Lu have a smart speaker in their bedroom. They also have a set of motion detectors and smart switches to control their bedroom lighting.

[Problem]
The couple is concerned that the smart speaker, smart switch, and motion detectors might be actively working at night while making love.

[User Action, Technology]
The couple installs smart power plugs to control the devices' power supply. For example, they only use the smart speaker in the morning to wake them up, so they specify that the plug only powers the smart speaker between 6 AM and 8 AM.

[Resolution]
In doing so, the couple feels more comfortable placing these smart home devices in their bedroom.

Figure 10.17: Storyboard #4 Guaranteed Protection



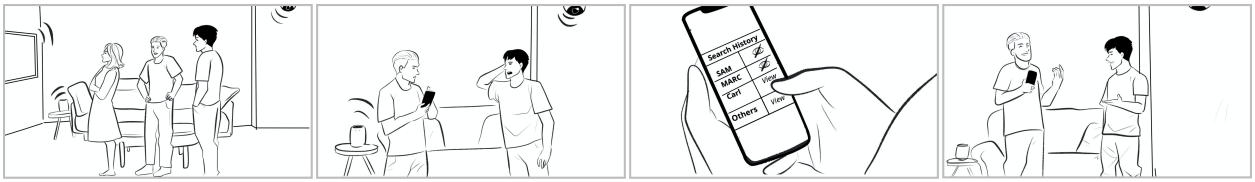
[Character, Context]
Charley installed numerous smart home devices at different places in his house.

[Problem]
He feels overwhelmed because these devices can collect personal data and send it to external servers without his awareness.

[User Action, Technology]
Charley installs a tool that enables him to view devices' data collection attributes including when, how, and where his data is going.

[Resolution]
Charley now feels much safer because he is no longer blinded from the data collection behaviors of his smart home devices.

Figure 10.18: Storyboard #5 Data Collection Live Monitor



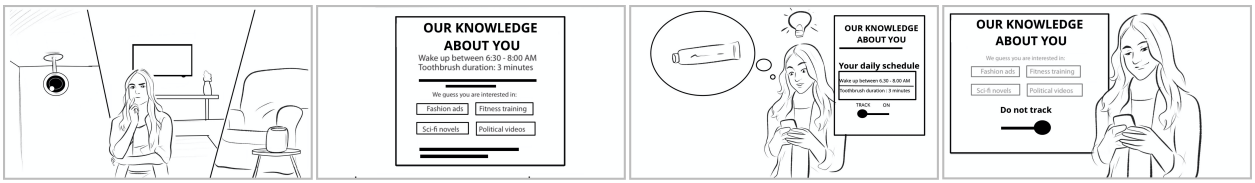
[Character, Context]
Sam rents a house with housemates, and they share smart home devices for security, temperature control, light control, etc. These devices record their voices, photos, setting preferences, etc. The residents use a mobile app to manage these devices using their accounts, respectively.

[Problem]
Sam is concerned that his collected by these shared devices will be accessible to other residents. For instance, his search history recorded by the voice assistant and images recorded by the smart camera in the front door is available to his housemates.

[User Action, Technology]
The mobile app now offers a new feature: the app will associate these personal data with residents' identities. By default, a user can only view one's personal search history and video recordings.

[Resolution]
Sam now feels relieved and secured about his privacy and is not worried about his friends viewing his data anymore.

Figure 10.19: Storyboard #6 Identity-based Data Management



[Character, Context, Problem]
Amy doesn't go through a single day without using smart home products from a single company. Though these devices improve her life quality, Amy begins to wonder what information the company knows about her.

[Technology]
Due to legal regulation, companies offer users a report about their knowledge of users and allows users to wipe some information so the company will forget them. For example, the report might tell Amy that her smart home vendor knows when she wakes up, brushes her teeth, and goes to sleep.

[User Action]
After reviewing the report, Amy understands why she gets the toothpaste advertisement when it runs out. However, since she is not interested in political videos, she requests the company to forget that information.

[Resolution]
Later, she finds fewer political video recommendations in her video feeds. She enjoys the transparency and control in the process and starts to read such a report every few weeks.

Figure 10.20: Storyboard #7 Privacy Mirrors

[Character, Context]
Lisa is cautious with her smart home device usage. She usually turns her smart speaker off during work hours, dinner time, and sleeping periods. She would turn the device back onto listening mode after work.

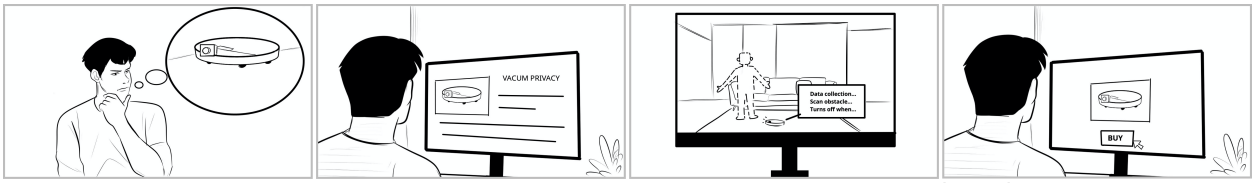
[Problem]
However, it's hard to remember to change the settings. Lisa feels frustrated when there is no response from her voice assistant or when her phone call is interrupted by the voice assistant because she forgot to turn on/off the device.

[Technology]
The smart home system recently offers a new feature, enabling users to set contextual reminders (e.g., remind one to turn off the speaker when having a family dinner, alert one to turn off the camera if found naked).

[User Action, Technology]
After Lisa turn on the feature, the device would now ask, "Lisa, would you like to turn me off for an hour?" around dinner time. And Lisa would readily confirm after receiving the notification.

[Resolution]
Lisa feels much relieved to manage her device settings.

Figure 10.21: Storyboard #9 Contextual Privacy Reminders



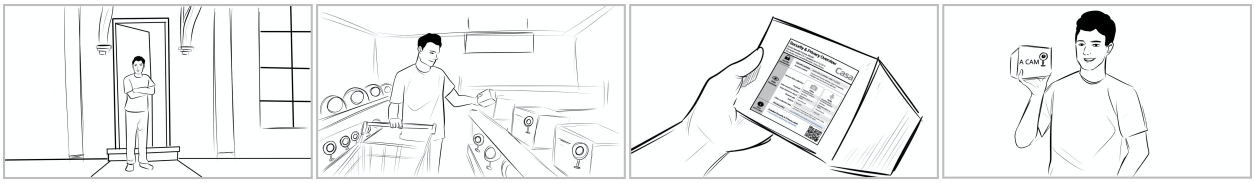
[Character, Context]
John is considering purchasing a vacuum robot, which searches for dirty spots to clean up with a camera. However, he feels skeptical and concerned that the camera might capture image/video data about people in his family.

[Problem]
Through exhaustive online searches, John found that the manufacturer's statement about data collection is very vague. Customer reviews and product testing reports barely touch on that topic either.

[User Action, Technology]
John uses a third-party tool that offers a simulation feature. He can drag an object into a simulated environment to see how the vacuum will react in the virtual setting and verify what data the robot might send out.

[Resolution]
John learns that the robot actually turn-off the camera automatically when it detects a person in the scene. He is now assured and decides to purchase the robot.

Figure 10.22: Storyboard #10 Privacy Simulator



[Character, Context]
Adam wants to purchase a smart camera to guard his front door for hours away from home.

[Problem]
While Adam is browsing through all types of security cameras at the store, he wonders how these cameras would handle the data they collect.

[User Action, Technology]
Later, he notices a privacy label on the packaging of each device. The label concisely informs him of the actual data practice and contains a QR Code that directs him to more detailed information on a website.

[Resolution]
Adam feels relieved as he can quickly understand how the devices work with his data. Furthermore, he can easily compare this information between different devices and purchase the one that suits his needs.

Figure 10.23: Storyboard #11 Privacy Nutrition Labels

10.6 Appendix: Privacy Speed Dating Survey Instrument

The appendix material is formatted differently from what participants saw in the survey. Sections correspond to the components in Fig. 8.3 and 8.4.

10.6.1 Generic privacy index questions and online PPB

Q1 Having companies collect my online behavior is a problem for me. ○ Strongly agree ○ Agree ○ Somewhat agree ○ Neither agree nor disagree ○ Somewhat disagree ○ Disagree ○ Strongly disagree
Q2 Having companies use my online behavior to show me advertisements is a problem for me. (Same options as Q1)
Q3 Having companies share my online behavior with other companies is a problem for me. (Same options as Q1)
Q4 Do you ever use some of the following behaviors to protect your personal information and privacy on the internet?

□ Use an ad blocker

□ Delete cookies

□ Decide to refrain from visiting a website because it is only accessible when you accept cookies

□ Decline to accept cookies when a website offers the choice

□ Use the private mode in your browser

□ Delete browser history

□ Use opt-out websites (such as www.youronlinechoices.com) to configure whether ads are based on personal data

□ Use the "Do Not Track" function in your browser

□ Use special software in your browser (such as Ghostery and Abine Taco) that makes it harder for companies to collect personal data

□ I don't use any of the behaviors above.

10.6.2 Smart home status questions

Q1 What best describes your living situation? ○ Rent ○ Own
Q2 Who do you live with at your home? ○ Alone ○ With family member(s) ○ With roommate(s)
Q3 Please check all types of smart home devices you have deployed in your home.

□ Smart Lighting (e.g., Philips Hue, LIFX, IKEA Tradfri, Xiaomi Yeelight)

□ Motion Sensor (e.g., Xiaomi Motion Sensor, Philips Hue Motion, Ecolink Motion Detector)

□ Smart Camera (e.g., Yi, Foscam, Xiaomi Xiaofang, Amrecst ProHD)

□ Smart Fan (e.g., Xiaomi Air Purifier, Dyson Link Air Purifier)

□ Robot Vacuum (e.g., Dyson 360 Eye, iRobot Roomba, Neato Robotics Botvac)

□ Door Sensor (e.g., Xiaomi Door Sensor, Ring Alarm Contact Sensor, Nest Detect Sensor)

□ Smart Switch/Button (e.g., Amazon Dash Button, Fibaro The Button, Xiaomi Button. Elko ESH)

□ Smart Temperature Sensor (e.g., Broadlink A1, Wink Relay, Fibaro Z, Xiaomi Temperature and Humidity Sensor)

□ Media (e.g., Amazon Echo, Google Home, Sonos PLAY, Apple TV)

□ Smart Power/Plug (e.g., Broadlink SP, TP-link Kasa Wi-fi Plug, Belkin Wemo Smart Plug, Fibaro Wall Plug)

□ IR Blaster (e.g., Broadlink RM, Xiaomi Universal IR Remote Controller)

□ Smart Smoke Detector (e.g., Nest Protect, Kidde Smoke Detector)

□ Smart Alarm (e.g., Dome Home Automation Siren and Chime, SimpliSafe Home Security System)

□ Smart Thermostat (e.g., Nest Learning Thermostat, Ecobee, GoControl Thermostat)

□ Smart Home Hub (e.g., Samsung Smart Things Hub, Wink Hub, Xiaomi Gateway)

□ I don't use any smart home devices.

Q4 Please estimate the number of smart home devices you have deployed in your home. ○ 0-5 ○ 5-10 ○ 10-20 ○ 20-40 ○ 40-80 ○ 80+

10.6.3 SH-PPB Definition Quiz

Bobby recently set up a smart camera in his living room for security purposes. You can see the layout of his living room below. The smart camera records and streams any video data it captures within the blue-shaded zone only.

Q1 Before setting up the camera, he used to sit on Seat A within the blue-shaded zone, but now he sits on Seat B more often to avoid being captured by the camera. Is this a smart home privacy-protective behavior? ○ Yes ○ No

Q2 Now, every time he gets into the blue-shaded zone, he receives an alert on his phone, which reports to him as "something has entered the camera' s view." Before setting up the camera, he used to sit on Seat A within the blue shaded zone, but now he sits on Seat B more often to avoid receiving repetitive non-useful alerts. Is this a smart home privacy-protective behavior? ○ Yes ○ No

Q3 Bobby also routed the power to the smart camera through a smart plug. The smart plug would automatically turn off the power for the camera when Bobby is at home. So the camera would not unnecessarily record him in the video streams. Is this a smart home privacy-protective behavior? ○ Yes ○ No

10.6.4 SH-PPB Inquiry Questions

Q1 Do you have any privacy-protective strategies, tricks, or experiences to share with us? We will review your answers manually. Note, for each valid unique PPB, we offer $1 bonus. ○ Yes ○ No

If the participant answers no: Q2 Here are the techniques you perform to protect your personal information and privacy on the internet: [we carry forward the choices the participants answered in online PPB questions]. Could you compare and elaborate on any differences in your privacy-protective behavior between online browsing and smart home usage.

If the participant answers yes: Q3 What do you actively do to mitigate the collection, usage, and sharing of your personal information from smart home devices to protect your online privacy? Please articulate the

context of your tricks, the types of involved devices, if any, the types of privacy concerns you may have, and how they can address your privacy concerns.

Q3.1 What is your trick/PPB?

Q3.2 What device(s) is relevant to your trick/PPB?

Q3.2 What device(s) is relevant to your trick/PPB?

Q3.3 What is your privacy concern(s)?

Q3.4 When and how often would you perform this trick/behavior?

Q3.5 What device features would you wish to have to better address your privacy concern(s)?

10.6.5 Questions for each storyboard

Q1 Could you relate to the person's concerns shown in the storyboard? ○ Definitely yes ○ Probably yes ○ Might or might not ○ Probably not ○ Definitely not

Q2 How does this technology shown in the storyboard address the described concerns? ○ Extremely effective ○ Very effective ○ Moderately effective ○ Slightly effective ○ Not effective at all

If the participant considers the technology effective in Q2:
Q3 Could you describe some scenarios that you may use the technology shown in the storyboard?

If the participant does not consider the technology effective in Q2:
Q4 How can this technology be made more effective to address your need? For example, what would you like to change or add?

10.6.6 Storyboards ranking

Q1 Please rank the technologies shown in the following scenarios in order of effectiveness of addressing your privacy need in your smart home, where 1 is most effective and 3 is least effective.

Q2 Could you briefly explain the rationale for your ranking?

Bibliography

[1] Roger clarke's 'privacy introduction and definitions'. http://www.rogerclarke.com/DV/Intro.html, 2016. (Accessed on 07/17/2018). 7.5.1, 7.4

[2] 42matters. Google play categories. https://42matters.com/docs/app-market-data/android/apps/google-play-categories, 2018. (Accessed on 02/11/2018). 2.4.1

[3] Paarijaat Aditya, Rijurekha Sen, Peter Druschel, Seong Joon Oh, Rodrigo Benenson, Mario Fritz, Bernt Schiele, Bobby Bhattacharjee, and Tong Tong Wu. I-pic: A platform for privacy-compliant image capture. In *Proceedings of the 14th annual international conference on mobile systems, applications, and services*, pages 235–248. ACM, 2016. 5.7, 6.1, 6.9

[4] Yuvraj Agarwal and Malcolm Hall. Protectmyprivacy: detecting and mitigating privacy leaks on ios devices using crowdsourcing. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 97–110, 2013. 2.1, 2.12, 5.3.1, 5.7

[5] Aseem Agarwala. Google ai blog: Automatic photography with google clips. https://ai.googleblog.com/2018/05/automatic-photography-with-google-clips.html, 05 2018. (Accessed on 06/08/2020). 3.1, 5.6.1

[6] Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the fifth ACM conference on Digital libraries*, pages 85–94. ACM, 2000. 2.3.1, 2.3.1

[7] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Hippocratic databases. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 143–154. Elsevier, 2002. 6.8

[8] Fox Van Allen. When the device you use determines the price you get. *Techlicious*, 2014. URL https://www.techlicious.com/blog/price-discrimination-by-operating-system-device/. 7.1, 7.1, 7.9

[9] Amazon. Does amazon have a minimum character requirement for reviews? - selling on amazon / general selling questions - amazon seller forums. https://sellercentral.amazon.

com/forums/t/does-amazon-have-a-minimum-character-requirement-for-reviews/130681, 05 2016. (Accessed on 02/17/2020). 7.5.3

[10] Amazon. Serverless computing - aws lambda - amazon web services. https://aws.amazon.com/lambda/, 12 2021. (Accessed on 12/19/2021). 6.1, 6.5

[11] Shahriyar Amini, Janne Lindqvist, Jason Hong, Jialiu Lin, Eran Toch, and Norman Sadeh. Caché: caching location-enhanced content to improve user privacy. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 197–210, 2011. 5.1

[12] Cynthia Van Ort Andrew Clearwater, Chad Quayle. An agile approach to pias and privacy by design. https://iapp.org/resources/article/an-agile-approach-to-pias-and-privacy-by-design/, 09 2016. (Accessed on 03/30/2020). 7.1, 7.3.1

[13] Android Developers. App manifest overview | android developers. https://developer.android.com/guide/topics/manifest/manifest-intro, 02 2021. (Accessed on 02/04/2021). 5.7

[14] Annie I Antón, Julia B Earp, and Jessica D Young. How internet users' privacy concerns have evolved since 2002. *IEEE Security & Privacy*, 8(1), 2010. 7.2.3

[15] Apple. Learning with privacy at scale - apple machine learning research. https://machinelearning.apple.com/research/learning-with-privacy-at-scale, 12 2021. (Accessed on 12/18/2021). 6.8

[16] Noah Apthorpe, Dillon Reisman, and Nick Feamster. A smart home is no castle: Privacy vulnerabilities of encrypted iot traffic. *arXiv preprint arXiv:1705.06805*, 2017. 5.6.3, 8.2.1

[17] Noah Apthorpe, Yan Shvartzshnaider, Arunesh Mathur, Dillon Reisman, and Nick Feamster. Discovering smart home internet of things privacy norms using contextual integrity. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2(2), jul 2018. doi: 10.1145/3214262. URL https://doi.org/10.1145/3214262. 8.2.1

[18] Noah Apthorpe, Yan Shvartzshnaider, Arunesh Mathur, Dillon Reisman, and Nick Feamster. Discovering smart home internet of things privacy norms using contextual integrity. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2(2):59:1–59:23, July 2018. ISSN 2474-9567. doi: 10.1145/3214262. URL http://doi.acm.org/10.1145/3214262. 7.2.2, 7.2.3, 7.3.1, 7.4.4, 7.5

[19] Noah Apthorpe, Yan Shvartzshnaider, Arunesh Mathur, Dillon Reisman, and Nick Feamster. Discovering smart home internet of things privacy norms using contextual integrity. *arXiv preprint arXiv:1805.06031*, 2018. 2.10

[20] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy H Katz, Andrew Konwinski, Gunho Lee, David A Patterson, Ariel Rabkin, Ion Stoica, et al. Above the clouds: A berkeley

view of cloud computing. Technical report, Technical Report UCB/EECS-2009-28, EECS Department, University of California ⋯, 2009. 6.2

[21] Charles Arthur. Marissa mayer's appointment: what does it mean for yahoo? | technology | the guardian. https://www.theguardian.com/technology/2012/jul/16/marissa-mayer-appointment-mean-yahoo?newsfeed=true, 2012. (Accessed on 01/30/2019). 7.1.1, 7.2, 7.1, 7.9

[22] Atlassian. Epics, stories, themes, and initiatives | atlassian. https://www.atlassian.com/agile/project-management/epics-stories-themes, 04 2020. (Accessed on 04/29/2020). 17

[23] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. Pscout: analyzing the android permission specification. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 217–228. ACM, 2012. 2.12, 2.8

[24] David Avis and Bohdan Kaluzny. Solving inequalities and proving farkas's lemma made easy. *The American Mathematical Monthly*, 111(2):152–157, 2004. 6.4.2

[25] Microsoft Azure. Cognitive services—apis for ai developers | microsoft azure. https://azure.microsoft.com/en-us/services/cognitive-services/, 11 2020. (Accessed on 11/15/2020). 5.6.2

[26] Nick Babich. Mobile app ux design: Making a great first impression. https://uxplanet.org/mobile-app-ux-design-making-a-great-first-impression-bed2805b967d, 2016. (Accessed on 10/23/2018). 2.9

[27] Guha Balakrishnan, Fredo Durand, and John Guttag. Detecting pulse from head motions in video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3430–3437, 2013. 5.3.1, 5.1, 5.5

[28] Louise Barkhuus. The mismeasurement of privacy: Using contextual integrity to reconsider privacy in hci. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 367–376, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1015-4. doi: 10.1145/2207676.2207727. URL http://doi.acm.org/10.1145/2207676.2207727. 7.2.2, 7.2.3

[29] Flow based Programming. flowbased/fbp: Fbp flow definition language parser. https://github.com/flowbased/fbp, 08 2022. (Accessed on 08/16/2022). 6.5

[30] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. Manifesto for agile software development. 2001. 7.10.1

[31] Michael S Bernstein, Greg Little, Robert C Miller, Björn Hartmann, Mark S Ackerman, David R Karger, David Crowell, and Katrina Panovich. Soylent: a word processor with a crowd inside. In

Proceedings of the 23nd annual ACM symposium on User interface software and technology, pages 313–322. ACM, 2010. 7.2.4

[32] Jaspreet Bhatia and Travis D Breaux. Empirical measurement of perceived privacy risk. 2018. 7.1, 7.2.3, 7.5

[33] E Blackburn and D Mares. The hidden costs of police technology: Evaluating acoustic gunshot detection systems. *Police Chief Magazine*, 2019. 6.9

[34] Joshua Bloch. How to design a good api and why it matters. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 506–507, 2006. 5.3.1, 6.3.3

[35] Sophie C Boerman, Sanne Kruikemeier, and Frederik J Zuiderveen Borgesius. Exploring motivations for online privacy protection behavior: Insights from panel data. *Communication Research*, page 0093650218800915, 2018. 8.1, 8.3.1

[36] Connor Bolton, Kevin Fu, Josiah Hester, and Jun Han. How to curtail oversensing in the home. *Communications of the ACM*, 63(6):20–24, 2020. 6.1

[37] Engin Bozdag. Privacy at speed: Privacy by design for agile development at uber. San Francisco, CA, January 2020. USENIX Association. 7.1

[38] Engin Bozdag. Privacy at speed: Privacy by design for agile development at uber. 2020. 7.1, 7.2.1, 7.3.1, 7.9.3

[39] Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. *Qualitative research in psychology*, 3(2):77–101, 2006. 7.6.1, 8.3.3

[40] Alex Braunstein, Laura Granka, and Jessica Staddon. Indirect content privacy surveys: measuring privacy without asking about it. In *Proceedings of the Seventh Symposium on Usable Privacy and Security*, page 15. ACM, 2011. 7.9.1, 7.11.1

[41] Lauren Bridges. Amazon' s ring is the largest civilian surveillance network the us has ever seen | lauren bridges | the guardian. `https://www.theguardian.com/commentisfree/2021/may/18/amazon-ring-largest-civilian-surveillance-network-us`, 05 2021. (Accessed on 12/15/2021). 6.1

[42] Sean Brooks, Sean Brooks, Michael Garcia, Naomi Lefkovitz, Suzanne Lightman, and Ellen Nadeau. *An introduction to privacy engineering and risk management in federal systems*. US Department of Commerce, National Institute of Standards and Technology, 2017. 7.4.4

[43] A.J. Bernheim Brush, John Krumm, and James Scott. Exploring end user preferences for location obfuscation, location-based services, and the value of location. In *Proceedings of the 12th ACM*

International Conference on Ubiquitous Computing, UbiComp '10, pages 95–104, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-843-8. doi: 10.1145/1864349.1864381. URL http://doi.acm.org/10.1145/1864349.1864381. 2.1

[44] Jeffrey A Burke, Deborah Estrin, Mark Hansen, Andrew Parker, Nithya Ramanathan, Sasank Reddy, and Mani B Srivastava. Participatory sensing. 2006. 6.1

[45] Ryan Calo. Digital market manipulation. *Geo. Wash. L. Rev.*, 82:995, 2013. 7.5.2

[46] Packet Capture. Packet capture - android apps on google play. https://play.google.com/store/apps/details?id=app.greyshirts.sslcapture&hl=en, 2017. (Accessed on 11/10/2017). 2.2.2

[47] Nicholas Carlson. Warning: Google buzz has a huge privacy flaw - business insider. https://www.businessinsider.com/warning-google-buzz-has-a-huge-privacy-flaw-2010-2, 2010. (Accessed on 12/18/2018). 7.1

[48] Patrick Carter, Collin Mulliner, Martina Lindorfer, William Robertson, and Engin Kirda. Curiousdroid: automated user interface interaction for android application analysis sandboxes. In *International Conference on Financial Cryptography and Data Security*, pages 231–249. Springer, 2016. 2.9

[49] Ben Carterette, Virgiliu Pavlu, Hui Fang, and Evangelos Kanoulas. Million query track 2009 overview. In *TREC*. Citeseer, 2009. 6.7.3

[50] Rosangela Casolare, Fabio Martinelli, Francesco Mercaldo, and Antonella Santone. Android collusion: Detecting malicious applications inter-communication through sharedpreferences. *Information*, 11(6):304, 2020. 9.3

[51] Ann Cavoukian. Privacy by design in law, policy and practice. *A white paper for regulators, decision-makers and policy-makers*, 2011. 7.2.1

[52] Alvin Chang. The facebook and cambridge analytica scandal, explained with a simple diagram - vox. https://www.vox.com/policy-and-politics/2018/3/23/17151916/facebook-cambridge-analytica-trump-diagram, 09 2022. (Accessed on 09/24/2022). 5

[53] Bill Chappell. Google: don't expect privacy when sending to gmail | technology | the guardian. https://www.theguardian.com/technology/2013/aug/14/google-gmail-users-privacy-email-lawsuit, 08 2013. (Accessed on 04/11/2020). 7.9.1

[54] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848,

2017. 5.3.1, 5.1

[55] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 578–594, 2018. 6.5

[56] Xin Chen and Sencun Zhu. Droidjust: Automated functionality-aware privacy leakage analysis for android applications. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 2015. 2.12, 2.8

[57] Haotian Chi, Qiang Zeng, Xiaojiang Du, and Lannan Luo. Pfirewall: Semantics-aware customizable data flow control for home automation systems. *arXiv preprint arXiv:1910.07987*, 2019. 5.7

[58] Jianfeng Chi, Emmanuel Owusu, Xuwang Yin, Tong Yu, William Chan, Patrick Tague, and Yuan Tian. Privacy partitioning: Protecting user data during the deep learning inference phase. *arXiv preprint arXiv:1812.02863*, 2018. 5.7

[59] Jeffrey T Child, Sandra Petronio, Esther A Agyeman-Budu, and David A Westermann. Blog scrubbing: Exploring triggers that change privacy rules. *Computers in Human Behavior*, 27(5):2017–2027, 2011. 8.2.2

[60] Lydia B Chilton, Greg Little, Darren Edge, Daniel S Weld, and James A Landay. Cascade: Crowdsourcing taxonomy creation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1999–2008, 2013. 7.2.4

[61] Saksham Chitkara, Nishad Gothoskar, Suhas Harish, Jason I Hong, and Yuvraj Agarwal. Does this app really need my location?: Context-aware privacy management for smartphones. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(3):42, 2017. 2.1, 2.12, 2.8

[62] Amit Chowdhry. Uber: Users are more likely to pay surge pricing if their phone battery is low. *Forbes*, 2016. URL `https://www.forbes.com/sites/amitchowdhry/2016/05/25/uber-low-battery`. 7.1, 7.9

[63] Cisco. What is mud? - manufacturer usage description - document - cisco devnet. `https://developer.cisco.com/docs/mud/`, 02 2021. (Accessed on 02/05/2021). 5.2, 5.7

[64] Cloud Research. Strengths and limitations of mechanical turk. `https://www.cloudresearch.com/resources/blog/strengths-and-limitations-of-mechanical-turk/`, 09 2021. (Accessed on 09/08/2021). 8.5

[65] David Cohen, Mikael Lindvall, and Patricia Costa. An introduction to agile methods. *Adv. Comput.*, 62(03):1–66, 2004. 7.1

[66] Julie E Cohen. Examined lives: Informational privacy and the subject as object. In *Law and Society Approaches to Cyberspace*, pages 473–538. Routledge, 2017. 1.1

[67] Federal Trade Commission et al. Protecting consumer privacy in an era of rapid change. *FTC report*, 2012. 7.5

[68] The Nielsen Company. Privacy please! u.s. smartphone app users concerned with privacy when it comes to location. https://www.nielsen.com/us/en/insights/news/2011/privacy-please-u-s-smartphone-app-users-concerned-with-privacy-when-it-comes-to-location.html, 2011. (Accessed on 12/20/2018). 7.2.3

[69] Sunny Consolvo, Ian E Smith, Tara Matthews, Anthony LaMarca, Jason Tabert, and Pauline Powledge. Location disclosure to social relations: why, when, & what people want to share. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 81–90. ACM, 2005. 7.2.3

[70] Josh Constine. Facebook rolls out ai to detect suicidal posts before they' re reported | techcrunch. https://techcrunch.com/2017/11/27/facebook-ai-suicide-prevention/?guccounter=1, 11 2017. (Accessed on 02/04/2020). 7.8.1, 7.9, 10.13

[71] Andrea Continella, Yanick Fratantonio, Martina Lindorfer, Alessandro Puccetti, Ali Zand, Christopher Kruegel, and Giovanni Vigna. Obfuscation-resilient privacy leak detection for mobile apps through differential analysis. In *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS)*, 2017. 2.8

[72] Lorrie Faith Cranor and Norman Sadeh. Privacy engineering emerges as a hot new career. *IEEE Potentials*, 32(6):7–9, 2013. 1, 7.1, 7.2.1, 7.8.1, 7.9.1, 7.12.2

[73] Mary J Culnan and Pamela K Armstrong. Information privacy concerns, procedural fairness, and impersonal trust: An empirical investigation. *Organization science*, 10(1):104–115, 1999. 7.2.3

[74] Adele Da Veiga. An information privacy culture instrument to measure consumer privacy expectations and confidence. *Information & Computer Security*, (just-accepted):00–00, 2018. 7.2.3

[75] Sauvik Das. Social cybersecurity: Understanding and leveraging social influence to increase security sensitivity. *it-Information Technology*, 58(5):237–245, 2016. 7.12.5

[76] Sauvik Das, Tiffany Hyun-Jin Kim, Laura A Dabbish, and Jason I Hong. The effect of social influence on security sensitivity. In *10th Symposium On Usable Privacy and Security ({SOUPS} 2014)*, pages 143–157, 2014. 8.4.2, 8.5

[77] Databroker. Welcome to databroker | the marketplace for data. https://www.databroker.global/, 08 2022. (Accessed on 08/07/2022). 6.1

[78] Amit Datta, Michael Carl Tschantz, and Anupam Datta. Automated experiments on ad privacy settings: A tale of opacity, choice, and discrimination. *Proceedings on privacy enhancing technologies*, 2015(1):92–112, 2015. 2.9

[79] Scott Davidoff, Min Kyung Lee, Anind K Dey, and John Zimmerman. Rapidly exploring application design through speed dating. In *International Conference on Ubiquitous Computing*, pages 429–446. Springer, 2007. 7.1, 8.1, 8.3.1, 8.3.1, 8.4.3, 8.6.3

[80] Jan Dawson. 'fear of one company knowing too much about us' and other privacy concerns - vox. `https://www.vox.com/2015/6/23/11563790/classifying-privacy-concerns`, 01 2015. (Accessed on 12/19/2021). 6.2

[81] Yves-Alexandre De Montjoye, Erez Shmueli, Samuel S Wang, and Alex Sandy Pentland. openpds: Protecting the privacy of metadata through safeanswers. *PloS one*, 9(7):e98790, 2014. 1.2, 6.8

[82] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. 2004. 4.2, 1, 6.8

[83] VIP Defense. Vip defense - a privacy and anonymity keeping company. `http://www.vipdefense.com/`, 2017. (Accessed on 11/08/2017). 2.8

[84] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, pages 845–854. ACM, 2017. 2.9

[85] Dylan D Furszyfer Del Rio, Benjamin K Sovacool, and Steve Griffiths. Culture, energy and climate sustainability, and smart home technologies: A mixed methods comparison of four countries. *Energy and Climate Change*, 2:100035, 2021. 4.1, 4.1

[86] Soteris Demetriou, Nan Zhang, Yeonjoon Lee, XiaoFeng Wang, Carl A Gunter, Xiaoyong Zhou, and Michael Grace. Hanguard: Sdn-driven protection of smart home wifi devices from malicious mobile apps. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 122–133, 2017. 5.7

[87] George Demiris, Brian K Hensel, Marjorie Skubic, and Marilyn Rantz. Senior residents' perceived need of and preferences for "smart home" sensor technologies. *International journal of technology assessment in health care*, 24(1):120–124, 2008. 7.2.2

[88] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009. 7.2.4

[89] Education Department of Health et al. The belmont report. ethical principles and guidelines for the

protection of human subjects of research. *The Journal of the American College of Dentists*, 81(3):4, 2014. 7.5.1, 7.4, 7.5

[90] Department of Homeland Security . Privacy impact assessment template. https://www.dhs.gov/xlibrary/assets/privacy/privacy_pia_template.pdf, 03 2020. (Accessed on 03/23/2020). 7.4.4

[91] Nikolay Derkach. Tutorial: Reverse engineering a private api. https://www.toptal.com/back-end/reverse-engineering-the-private-api-hacking-your-couch, 2017. (Accessed on 11/04/2017). 2.1.1, 2.6

[92] Anthony Desnos et al. Androguard. *URL: https://github.com/androguard/androguard*, 2011. 2.4.1

[93] Android Developer. Network security configuration ălă android developers. https://developer.android.com/training/articles/security-config. (Accessed on 09/07/2020). 5.3.1

[94] Apple Developer. Expected app behaviors. https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/ExpectedAppBehaviors/ExpectedAppBehaviors.html, 2018. (Accessed on 01/31/2018). 2.1

[95] Apple Developer. Requesting permission - app architecture - ios - human interface guidelines - apple developer. https://developer.apple.com/design/human-interface-guidelines/ios/app-architecture/requesting-permission/, 04 2020. (Accessed on 04/16/2020). 5.7

[96] Android Developers. Advertising id. http://www.androiddocs.com/google/play-services/id.html, 2017. (Accessed on 12/30/2017). 2.3.1

[97] Android Developers. Requesting permissions. https://developer.android.com/guide/topics/permissions/requesting.html, 2017. (Accessed on 11/10/2017). 2.2.2

[98] Android Developers. Uuid. https://developer.android.com/reference/java/util/UUID.html, 2017. (Accessed on 12/30/2017). 2.3.1

[99] Android Developers. Best practices for unique identifiers. https://developer.android.com/training/articles/user-data-ids.html, 2018. (Accessed on 02/14/2018). 2

[100] Aaron Yi Ding, Gianluca Limon De Jesus, and Marijn Janssen. Ethical hacking for boosting iot vulnerability management: a first look into bug bounty programs and responsible disclosure. In *Proceedings of the Eighth International Conference on Telecommunications and Remote Sensing*, pages 49–55, 2019. 7.2.4

[101] Colin Dixon, Ratul Mahajan, Sharad Agarwal, AJ Brush, Bongshin Lee, Stefan Saroiu, and Paramvir Bahl. An operating system for the home. In *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, pages 337–352, 2012. 1, 5.1, 5.5, 5.7

[102] Ryan Drapeau, Lydia B Chilton, Jonathan Bragg, and Daniel S Weld. Microtalk: Using argumentation to improve crowdsourcing accuracy. In *Fourth AAAI Conference on Human Computation and Crowdsourcing*, 2016. 7.2.4

[103] Nora A Draper and Joseph Turow. The corporate cultivation of digital resignation. *New Media & Society*, 21(8):1824–1839, 2019. 1.1

[104] Akshay Dua, Nirupama Bulusu, Wu-Chang Feng, and Wen Hu. Towards trustworthy participatory sensing. In *Proceedings of the 4th USENIX Conference on Hot Topics in Security*, pages 8–8. USENIX Association Berkeley, CA, USA, 2009. 6.9

[105] Matt Duckham and Lars Kulik. Location privacy and location-aware computing. *Dynamic & mobile GIS: investigating change in space and time*, 3:35–51, 2006. 5.3.1

[106] Anthony Dunne and Fiona Raby. *Speculative everything: design, fiction, and social dreaming*. MIT press, 2013. 3.1

[107] Manuel Egele, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. PiOS : Detecting privacy leaks in iOS applications. In *NDSS 2011, 18th Annual Network and Distributed System Security Symposium, 6-9 February 2011, San Diego, CA, USA*, San Diego, UNITED STATES, 02 2011. URL `http://www.eurecom.fr/publication/3282`. 2.12, 2.8

[108] Serge Egelman, Adrienne Porter Felt, and David Wagner. Choice architecture and smartphone privacy: There' sa price for that. In *The economics of information security and privacy*, pages 211–236. Springer, 2013. 7.2.2

[109] Malin Eiband, Mohamed Khamis, Emanuel Von Zezschwitz, Heinrich Hussmann, and Florian Alt. Understanding shoulder surfing in the wild: Stories from users and observers. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 4254–4265, 2017. 7.2.2

[110] Pardis Emami-Naeini, Yuvraj Agarwal, Lorrie Faith Cranor, and Hanan Hibshi. Ask the experts: What should be on an iot privacy and security label? In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 447–464. IEEE, 2020. 6.6.1, 8.3.1, 8.4.2, 8.6

[111] Pardis Emami-Naeini, Yuvraj Agarwal, Lorrie Faith Cranor, and Hanan Hibshi. Ask the experts: What should be on an iot privacy and security label? In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 447–464, 2020. doi: 10.1109/SP40000.2020.00043. 5.1

[112] Pardis Emami-Naeini, Janarth Dheenadhayalan, Yuvraj Agarwal, and Lorrie Faith Cranor. An informative security and privacy "nutrition" label for internet of things devices. *IEEE Security & Privacy*, 20(2):31–39, 2021. 6.6.1

[113] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. Taintdroid: an information-flow tracking

system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):5, 2014. 2.2.1, 2.1, 2.8, 2.12

[114] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067, 2014. 6.8

[115] Martin Erwig and Markus Schneider. Developments in spatio-temporal query languages. In *Proceedings. Tenth International Workshop on Database and Expert Systems Applications. DEXA 99*, pages 441–449. IEEE, 1999. 4.2

[116] Benedict Evans. Privacy, ads and confusion —benedict evans. https://www.ben-evans.com/benedictevans/2021/8/27/understanding-privacy, 09 2022. (Accessed on 09/24/2022). 2

[117] Dean Eckles Eytan Bakshy and Michael Bernstein. Big experiments: Big data' s friend for making decisions | facebook. https://www.facebook.com/notes/facebook-data-science/big-experiments-big-datas-friend-for-making-decisions/10152160441298859/, 03 2014. (Accessed on 03/02/2020). 7.2.1

[118] Hossein Falaki, Dimitrios Lymberopoulos, Ratul Mahajan, Srikanth Kandula, and Deborah Estrin. A first look at traffic on smartphones. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 281–287. ACM, 2010. 2.1.1

[119] Florian M Farke, David G Balash, Maximilian Golla, Markus Dürmuth, and Adam J Aviv. Are privacy dashboards good for end users? evaluating user perceptions and reactions to google' s my activity. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, pages 483–500, 2021. 8.3.1

[120] Kassem Fawaz and Kang G. Shin. Location privacy protection for smartphone users. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, page 239–250, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450329576. doi: 10.1145/2660267.2660270. URL https://doi.org/10.1145/2660267.2660270. 3.1

[121] Adrienne Porter Felt, Serge Egelman, and David Wagner. I've got 99 problems, but vibration ain't one: A survey of smartphone users' concerns. In *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '12, pages 33–44, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1666-8. doi: 10.1145/2381934.2381943. URL http://doi.acm.org/10.1145/2381934.2381943. 7.5.2, 7.5, 7.5.3, 12, 7.6

[122] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the eighth symposium on usable privacy and security*, pages 1–14, 2012. 2.1, 5.2

[123] Rachel L Finn, David Wright, and Michael Friedewald. Seven types of privacy. In *European data protection: coming of age*, pages 3–32. Springer, 2013. 7.5.1, 7.4

[124] Fletchercross. Services and fees | fletchercross. https://www.fletchercrossconsulting.com/services-and-fees, 08 2022. (Accessed on 08/20/2022). 6.7.5

[125] Forcepoint. Forcepoint | human-centric cybersecurity. https://www.forcepoint.com/, 2017. (Accessed on 11/08/2017). 2.8

[126] Forcepoint. Tls inspection and how it works. https://help.stonesoft.com/onlinehelp/StoneGate/SMC/6.2.0/GUID-2B5CE217-C9FE-4D8F-915E-FEA1F8253BEC.html, 08 2021. (Accessed on 08/09/2021). 5.9

[127] Nuno Fortes, Paulo Rita, and Margherita Pagani. The effects of privacy concerns, perceived risk and trust on online purchasing behaviour. *International Journal of Internet Marketing and Advertising*, 11(4):307–329, 2017. 7.5

[128] Sheera Frenkel and Kate Conger. Facebook' s security chief to depart for stanford university - the new york times. https://www.nytimes.com/2018/08/01/technology/facebook-security-alex-stamos.html, 2018. (Accessed on 01/15/2019). 7.1

[129] Andrea Frome, German Cheung, Ahmad Abdulkader, Marco Zennaro, Bo Wu, Alessandro Bissacco, Hartwig Adam, Hartmut Neven, and Luc Vincent. Large-scale privacy protection in google street view. In *2009 IEEE 12th international conference on computer vision*, pages 2373–2380. IEEE, 2009. 6.4.2

[130] Simson Garfinkel, John M Abowd, and Christian Martindale. Understanding database reconstruction attacks on public data. *Communications of the ACM*, 62(3):46–53, 2019. 6.4.2

[131] David Garlan and Mary Shaw. An introduction to software architecture. In *Advances in software engineering and knowledge engineering*, pages 1–39. World Scientific, 1993. 5.3

[132] Jort F. Gemmeke, Daniel P. W. Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R. Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *Proc. IEEE ICASSP 2017*, New Orleans, LA, 2017. 5.3.1, 5.1, 5.6.3

[133] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210, 2016. 5.1, 5.7

[134] Ian Goldberg, David Wagner, Randi Thomas, Eric A Brewer, et al. A secure environment for untrusted helper applications: Confining the wily hacker. In *Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography*, volume 6, pages 1–1, 1996. 5.7

[135] Google. Aiy projects. https://aiyprojects.withgoogle.com/, 07 2021. (Accessed on 07/19/2021). 5.5

[136] Google. Guest mode: An easy privacy control for your home devices. https://blog.google/products/assistant/introducing-guest-mode/, 01 2021. (Accessed on 12/02/2021). 5.4

[137] Google AI. Google ai blog: Take your best selfie automatically, with photobooth on pixel 3. https://ai.googleblog.com/2019/04/take-your-best-selfie-automatically.html, 04 2019. (Accessed on 08/10/2020). 3.2

[138] Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. Checking app behavior against app descriptions. In *Proceedings of the 36th International Conference on Software Engineering*, pages 1025–1035. ACM, 2014. 2.12, 2.10

[139] Kevin Granville. Facebook and cambridge analytica: What you need to know as fallout widens. *The New York Times*, 2018. 1.1, 7.1

[140] Frances S Grodzinsky and Herman T Tavani. Privacy in the cloud: applying nissenbaum's theory of contextual integrity. *ACM SIGCAS Computers and Society*, 41(1):38–47, 2011. 7.2.1

[141] Jens Grossklags and Alessandro Acquisti. When 25 cents is too much: An experiment on willingness-to-sell and willingness-to-protect personal information. 7.2.3

[142] The Guardian. Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach | news. https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election, 2018. (Accessed on 09/26/2018). 7.1, 7.1

[143] Dominique Guinard, Iulia Ion, and Simon Mayer. In search of an internet of things service architecture: Rest or ws-*? a developers' perspective. In Alessandro Puiatti and Tao Gu, editors, *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pages 326–337, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-30973-1. 5.7

[144] Anhong Guo, Anuraag Jain, Shomiron Ghose, Gierad Laput, Chris Harrison, and Jeffrey P Bigham. Crowd-ai camera sensing in the real world. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(3):1–20, 2018. 6.7.4

[145] Bin Guo, Zhiwen Yu, Xingshe Zhou, and Daqing Zhang. From participatory sensing to mobile crowd sensing. In *2014 IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM WORKSHOPS)*, pages 593–598. IEEE, 2014. 6.1

[146] Qi Guo, Haojian Jin, Dmitry Lagun, Shuai Yuan, and Eugene Agichtein. Mining touch interaction data on mobile devices to predict web search result relevance. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 153–162. ACM,

2013. 7.4.3

[147] Seda Gürses. Can you engineer privacy? *Communications of the ACM*, 57(8):20–23, 2014. 1

[148] Seda Gürses, Carmela Troncoso, and Claudia Diaz. Engineering privacy by design. *Computers, Privacy & Data Protection*, 14(3):25, 2011. 1, 1.3.1

[149] Seungyeop Han, Jaeyeon Jung, and David Wetherall. A study of third-party tracking by mobile apps in the wild, 2012. 2.2.1, 2.1, 2.2.2, 2.12

[150] Eszter Hargittai. Facebook privacy settings: Who cares? *First Monday*, 2010. 8.2.2

[151] Drew Harwell. Ring, the doorbell-camera firm, has partnered with 400 police forces, extending surveillance reach - the washington post. https://www.washingtonpost.com/technology/2019/08/28/doorbell-camera-firm-ring-has-partnered-with-police-forces-extending-surveillance-reach/, 08 2019. (Accessed on 05/12/2022). 6.1

[152] Drew Harwell. The pregnancy-tracking app ovia lets women record their most sensitive data for themselves —and their boss - the washington post. https://www.washingtonpost.com/technology/2019/04/10/tracking-your-pregnancy-an-app-may-be-more-public-than-you-think/?arc404=true, 04 2019. (Accessed on 02/04/2020). 7.9

[153] Chris Hawblitzel, Jon Howell, Jacob R Lorch, Arjun Narayan, Bryan Parno, Danfeng Zhang, and Brian Zill. Ironclad apps:{End-to-End} security via automated {Full-System} verification. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 165–181, 2014. 6.8

[154] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015. 6.7.1

[155] Weijia He, Maximilian Golla, Roshni Padhi, Jordan Ofek, Markus Dürmuth, Earlence Fernandes, and Blase Ur. Rethinking access control and authentication for the home internet of things (iot). In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 255–272, 2018. 8.2.2

[156] Miguel Helft. Google alters buzz service over privacy concerns - the new york times. https://www.nytimes.com/2010/02/15/technology/internet/15google.html, 2010. (Accessed on 01/09/2019). 7.9, 10.13

[157] Kashmir Hill. How target figured out a teen girl was pregnant before her father did. https://www.forbes.com/sites/kashmirhill/2012/02/16/how-target-figured-out-a-teen-girl-was-pregnant-before-her-father-did/, 02 2012. (Accessed on 08/27/2019). 7.3

[158] Kashmir Hill. How target figured out a teen girl was pregnant before her father did. *Forbes, Inc*, 2012. 7.4.1, 7.1, 7.3, 7.4.4, 7.4.4, 7.9

[159] Kashmir Hill. Okcupid lied to users about their compatibility as an experiment. https://www.forbes.com/sites/kashmirhill/2014/07/28/okcupid-experiment-compatibility-deception/, 07 2014. (Accessed on 07/06/2018). 7.1, 7.9

[160] Chris Hoffman. Browser extensions are a privacy nightmare: Stop using so many of them. https://www.howtogeek.com/188346/why-browser-extensions-can-be-dangerous-and-how-to-protect-yourself/, 09 2022. (Accessed on 09/24/2022). 1

[161] Pi hole. Pi-hole –network-wide protection. https://pi-hole.net/, 09 2021. (Accessed on 09/04/2021). 8.1, 9.1

[162] LAURA M. HOLSON. Putting a bolder face on google - the new york times. https://www.nytimes.com/2009/03/01/business/01marissa.html?pagewanted=3&mtrref=undefined&gwh=2970AE389901E20D8913B8FB7ABB5DBE&gwt=pay, 2009. (Accessed on 01/30/2019). 7.1.1, 7.2, 7.1, 7.9

[163] Jason I Hong and James A Landay. An architecture for privacy-sensitive ubiquitous computing. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 177–189, 2004. 2.1

[164] Jason I Hong, Jennifer D Ng, Scott Lederer, and James A Landay. Privacy risk models for designing privacy-sensitive ubiquitous computing systems. In *Proceedings of the 5th conference on Designing interactive systems: processes, practices, methods, and techniques*, pages 91–100. ACM, 2004. 7.4.3, 7.4.3, 7.4.4, 7.5

[165] White House. Consumer data privacy in a networked world: A framework for protecting privacy and promoting innovation in the global digital economy. *White House, Washington, DC*, pages 1–62, 2012. 2.10

[166] Danny Yuxing Huang, Noah Apthorpe, Frank Li, Gunes Acar, and Nick Feamster. Iot inspector: Crowdsourcing labeled network traffic from smart home devices at scale. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(2):1–21, 2020. 5.7, 8.2.2

[167] Jeff Huang, Oren Etzioni, Luke Zettlemoyer, Kevin Clark, and Christian Lee. Revminer: An extractive interface for navigating reviews on a smartphone. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pages 3–12. ACM, 2012. 2.3.1

[168] Janet Shibley Hyde. Gender similarities and differences. *Annual review of psychology*, 65:373–398,

2014. 8.5

[169] Giovanni Iachello, Jason Hong, et al. End-user privacy in human–computer interaction. *Foundations and Trends® in Human–Computer Interaction*, 1(1):1–137, 2007. 7.2.1

[170] IBM' s Emerging Technology Services. Node-red. https://nodered.org/, 07 2020. (Accessed on 07/06/2020). 5.1, 5.5, 6.5

[171] IDEO. 7 simple rules of brainstorming –ideo u. https://www.ideou.com/blogs/inspiration/7-simple-rules-of-brainstorming, 01 2022. (Accessed on 01/06/2022). 8.6.3

[172] Intel IoT Devkit. intel-iot-devkit/sample-videos: Sample videos for running inference. https://github.com/intel-iot-devkit/sample-videos, 08 2021. (Accessed on 08/18/2021). 5.6.3

[173] Intersog. How many app developers are there in the world? - intersog. https://intersog.com/blog/how-many-developers-are-there-in-the-world, 09 2022. (Accessed on 09/24/2022). 2

[174] Yasha Iravantchi, Karan Ahuja, Mayank Goel, Chris Harrison, and Alanson Sample. *PrivacyMic: Utilizing Inaudible Frequencies for Privacy Preserving Daily Activity Recognition*. Association for Computing Machinery, New York, NY, USA, 2021. ISBN 9781450380966. URL https://doi.org/10.1145/3411764.3445169. 5.8

[175] Qatrunnada Ismail, Tousif Ahmed, Apu Kapadia, and Michael K Reiter. Crowdsourced exploration of security configurations. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 467–476, 2015. 7.2.4

[176] IT Pro Team. How to stay anonymous online | it pro. https://www.itpro.com/privacy/30584/how-to-stay-anonymous-online, 08 2018. (Accessed on 08/13/2021). 8.1

[177] Molly Jackman and Lauri Kanerva. Evolving the irb: Building robust review for industry research. *Washington and Lee Law Review Online*, 72(3):442, 2016. 7.10.3

[178] Suman Jana, Arvind Narayanan, and Vitaly Shmatikov. A scanner darkly: Protecting user privacy from perceptual applications. In *2013 IEEE Symposium on Security and Privacy*, pages 349–363, 2013. doi: 10.1109/SP.2013.31. 5.3.1, 5.7

[179] Stuart Jeffries. Mansion map: a guide to rich people's houses, until google blurs them out | homes | the guardian. https://www.theguardian.com/lifeandstyle/shortcuts/2014/jul/14/mansion-map-where-rich-and-famous-live, 07 2014. (Accessed on 07/25/2022). 6.9

[180] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Zhuoqing Morley Mao, Atul Prakash, and Shanghai JiaoTong Unviersity. Contexlot: Towards providing contextual integrity to appified iot platforms. 2017. 7.2.3

[181] H. Jin, G. Liu, D. Hwang, S. Kumar, Y. Agarwal, and J. Hong. Peekaboo: A hub-based approach to enable transparency in data processing within smart homes. In *2022 2022 IEEE Symposium on Security and Privacy (SP) (SP)*, pages 1571–1571, Los Alamitos, CA, USA, may 2022. IEEE Computer Society. doi: 10.1109/SP46214.2022.00142. URL https://doi.ieeecomputersociety.org/10.1109/SP46214.2022.00142. (document), 1, 1.1, 8.6.2

[182] Haojian Jin, Tetsuya Sakai, and Koji Yatani. Reviewcollage: a mobile interface for direct comparison using online reviews. In *Proceedings of the 16th international conference on Human-computer interaction with mobile devices & services*, pages 349–358. ACM, 2014. 2.3.1, 7.4.1

[183] Haojian Jin, Minyi Liu, Kevan Dodhia, Yuanchun Li, Gaurav Srivastava, Matthew Fredrikson, Yuvraj Agarwal, and Jason I Hong. Why are they collecting my data?: Inferring the purposes of network traffic in mobile apps. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(4):173, 2018. (document), 3.1, 8.3.1

[184] Haojian Jin, Swarun Kumar, and Jason I. Hong. Sensor as a company: On self-sustaining iot commons, 2021. URL https://arxiv.org/abs/2112.02775. 6.1

[185] Haojian Jin, Hong Shen, Mayank Jain, Swarun Kumar, and Jason I. Hong. Lean privacy review: Collecting users' privacy concerns of data practices at a low cost. *ACM Trans. Comput.-Hum. Interact.*, 28(5), aug 2021. ISSN 1073-0516. doi: 10.1145/3463910. URL https://doi.org/10.1145/3463910. 8.2.1

[186] Haojian Jin, Hong Shen, Mayank Jain, Swarun Kumar, and Jason I Hong. Lean privacy review: Collecting users' privacy concerns of data practices at a low cost. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 28(5):1–55, 2021. (document)

[187] Haojian Jin, Boyuan Guo, Rituparna Roychoudhury, Yaxing Yao, Swarun Kumar, Yuvraj Agarwal, and Jason I Hong. Exploring the needs of users for supporting privacy-protective behaviors in smart homes. In *CHI Conference on Human Factors in Computing Systems*, pages 1–19, 2022. (document)

[188] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, et al. Cloud programming simplified: A berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383*, 2019. 6.2

[189] Jaeyeon Jung, Anmol Sheth, Ben Greenstein, David Wetherall, Gabriel Maganis, and Tadayoshi Kohno. Privacy oracle: a system for finding application leaks with black box differential testing. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 279–288. ACM, 2008. 2.1, 2.11, 2.8

[190] Juniper. Ssl forward proxy. https://docs.paloaltonetworks.com/pan-os/8-1/pan-os-admin/decryption/decryption-concepts/ssl-forward-proxy, 08 2021. (Ac-

cessed on 08/19/2021). 5.9

[191] Kaggle. Test leakage | wiki. https://www.kaggle.com/wiki/Leakage, 2018. (Accessed on 02/15/2018). 2.6

[192] Rishabh Khandelwal, Thomas Linden, Hamza Harkous, and Kassem Fawaz. Prisec: A privacy settings enforcement controller. 2021. 5.7

[193] Dohyun Kim, Prasoon Patidar, Han Zhang, Abhijith Anilkumar, and Yuvraj Agarwal. Self-serviced iot: Practical and private iot computation offloading with full user control, 2022. URL https://arxiv.org/abs/2205.04405. 6.8

[194] Aniket Kittur, Ed H Chi, and Bongwon Suh. Crowdsourcing user studies with mechanical turk. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 453–456. ACM, 2008. 7.2.4

[195] Aniket Kittur, Boris Smus, Susheel Khamkar, and Robert E Kraut. Crowdforge: Crowdsourcing complex work. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 43–52, 2011. 7.2.4

[196] Ronny Ko and James Mickens. Deadbolt: Securing iot deployments. In *Proceedings of the Applied Networking Research Workshop*, pages 50–57, 2018. 5.7

[197] Ron Kohavi, Alex Deng, Brian Frasca, Toby Walker, Ya Xu, and Nils Pohlmann. Online controlled experiments at large scale. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1168–1176, 2013. 7.2.1

[198] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016. 5.1, 5.7

[199] Adam DI Kramer, Jamie E Guillory, and Jeffrey T Hancock. Experimental evidence of massive-scale emotional contagion through social networks. *Proceedings of the National Academy of Sciences*, page 201320040, 2014. 7.2.3, 7.1, 7.4.2, 7.3, 7.8.1, 7.9, 10.13

[200] Katharina Krombholz, Adrian Dabrowski, Matthew Smith, and Edgar Weippl. Exploring design directions for wearable privacy. 2017. 7.2.2

[201] John Krumm. Inference attacks on location tracks. In *International Conference on Pervasive Computing*, pages 127–143. Springer, 2007. 5.3.1

[202] Thomas Kubitza, Patrick Bader, Matthias Mogerle, and Albrecht Schmidt. Developing iot systems: It's all about the software. *Computer*, 53(4):58–62, 2020. 3.1

[203] Ponnurangam Kumaraguru and Lorrie Faith Cranor. *Privacy indexes: a survey of Westin's studies*.

2005. 7.2.3, 7.5.2, 7.5

[204] Michelle Kwasny, Kelly Caine, Wendy A Rogers, and Arthur D Fisk. Privacy and technology: folk definitions and perspectives. In *CHI'08 Extended Abstracts on Human Factors in Computing Systems*, pages 3291–3296. 2008. 7.2.2

[205] CMU CHIMPS Lab. Privacygrade: Grading the privacy of smartphone apps. `http://privacygrade.org/`, 2017. 2.1, 2.12, 2.8, 2.11

[206] Samuel Lampa. samuell/awesome-fbp: Awesome flow-based programming (fbp) resources. `https://github.com/samuell/awesome-fbp`, 06 2020. (Accessed on 12/02/2021). 5.8

[207] Eric C Larson, Mayank Goel, Gaetano Boriello, Sonya Heltshe, Margaret Rosenfeld, and Shwetak N Patel. Spirosmart: using a microphone to measure lung function on a mobile phone. In *Proceedings of the 2012 ACM conference on ubiquitous computing*, pages 280–289, 2012. 3.1, 5.6.1

[208] Walter Lasecki, Christopher Miller, Adam Sadilek, Andrew Abumoussa, Donato Borrello, Raja Kushalnagar, and Jeffrey Bigham. Real-time captioning by groups of non-experts. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pages 23–34, 2012. 7.2.4

[209] Josephine Lau, Benjamin Zimmerman, and Florian Schaub. Alexa, are you listening? privacy perceptions, concerns and privacy-seeking behaviors with smart speakers. In *Pro. ACM Human-Computer Interaction CSCW*. ACM, 2018. 8.2.2

[210] Anh Le, Janus Varmarken, Simon Langhoff, Anastasia Shuba, Minas Gjoka, and Athina Markopoulou. Antmonitor: A system for monitoring from mobile devices. In *Proceedings of the 2015 ACM SIGCOMM Workshop on Crowdsourcing and Crowdsharing of Big (Internet) Data*, pages 15–20. ACM, 2015. 2.3, 2.11, 2.8, 2.8

[211] Haein Lee, Hyejin Park, and Jinwoo Kim. Why do people share their context information on social network services? a qualitative study and an experimental study on users' behavior of balancing perceived benefit and risk. *International Journal of Human-Computer Studies*, 71(9):862–877, 2013. 7.2.2

[212] Taegyeong Lee, Zhiqi Lin, Saumay Pushp, Caihua Li, Yunxin Liu, Youngki Lee, Fengyuan Xu, Chenren Xu, Lintao Zhang, and Junehwa Song. Occlumency: Privacy-preserving remote deep-learning inference using sgx. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–17, 2019. 5.1, 5.7

[213] Sam Levin. Facebook told advertisers it can identify teens feeling 'insecure' and 'worthless' | technology | the guardian. `https://www.theguardian.com/technology/2017/may/01/facebook-advertising-data-insecure-teens`, 05 2017. (Accessed on 02/17/2020). 7.8.1,

7.9, 10.13

[214] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020. 5.7

[215] Tianshi Li, Yuvraj Agarwal, and Jason I. Hong. Coconut: An ide plugin for developing privacy-friendly apps. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2019. 2.10

[216] Tianshi Li, Elijah B Neundorfer, Yuvraj Agarwal, and Jason I Hong. Honeysuckle: Annotation-guided code generation of in-app privacy notices. *Proc. ACM IMWUT*, 5(3):1–27, 2021. 8.6.2

[217] Toby Jia-Jun Li and Oriana Riva. Kite: Building conversational bots from mobile apps. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 96–109. ACM, 2018. 2.8, 2.9

[218] Toby Jia-Jun Li, Amos Azaria, and Brad A Myers. Sugilite: creating multimodal smartphone automation by demonstration. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 6038–6049. ACM, 2017. 2.9

[219] Toby Jia-Jun Li, Jingya Chen, Brandon Canfield, and Brad A Myers. Privacy-preserving script sharing in gui-based programming-by-demonstration systems. *Proceedings of the ACM on Human-Computer Interaction*, 4(CSCW1):1–23, 2020. 8.2.1

[220] Yang Li, Jason I Hong, and James A Landay. Design challenges and principles for wizard of oz testing of location-enhanced applications. *IEEE Pervasive Computing*, 6(2):70–75, 2007. 7.2.2

[221] Yuanchun Li, Fanglin Chen, Toby Jia-Jun Li, Yao Guo, Gang Huang, Matthew Fredrikson, Yuvraj Agarwal, and Jason I. Hong. Privacystreams: Enabling transparency in personal data processing for mobile apps. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(3), September 2017. doi: 10.1145/3130941. URL `https://doi.org/10.1145/3130941`. 3.1, 5.7

[222] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. Droidbot: a lightweight ui-guided test input generator for android. In *Proceedings of the 39th International Conference on Software Engineering Companion*, pages 23–26. IEEE Press, 2017. 2.5

[223] Chieh-Jan Mike Liang, Nicholas D. Lane, Niels Brouwers, Li Zhang, Börje F. Karlsson, Hao Liu, Yan Liu, Jun Tang, Xiang Shan, Ranveer Chandra, and Feng Zhao. Caiipa: Automated large-scale mobile app testing through contextual fuzzing. In *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*, MobiCom '14, pages 519–530, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2783-1. doi: 10.1145/2639108.2639131. URL `http://doi.acm.org/10.1145/2639108.2639131`. 2.9

[224] Libelium. 50 sensor applications for a smarter world - libelium. `https://www.libelium.com/libeliumworld/top_50_iot_sensor_applications_ranking/`, 09 2020. (Accessed on

04/26/2022). 4.1, 4.1

[225] Jialiu Lin, Shahriyar Amini, Jason I Hong, Norman Sadeh, Janne Lindqvist, and Joy Zhang. Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing. In *Proceedings of the 2012 ACM conference on ubiquitous computing*, pages 501–510, 2012. 2.1, 2.1.1, 2.8, 2.10, 2.11

[226] Jialiu Lin, Bin Liu, Norman Sadeh, and Jason I. Hong. Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings. pages 199–212, 2014. 2.2.1, 2.1, 2.2.2, 7.5

[227] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 5.3.1, 5.1, 6.7.1

[228] TensorFlow Lite. Pose estimation | tensorflow lite. `https://www.tensorflow.org/lite/models/pose_estimation/overview`. (Accessed on 11/14/2020). 5.3.1, 5.1

[229] Leib Litman, Jonathan Robinson, and Tzvi Abberbock. Turkprime. com: A versatile crowdsourcing data acquisition platform for the behavioral sciences. *Behavior research methods*, 49(2):433–442, 2017. 7.9.1, 8.3.2

[230] Di Liu, Randolph G Bias, Matthew Lease, and Rebecca Kuipers. Crowdsourcing for usability testing. *Proceedings of the American Society for Information Science and Technology*, 49(1):1–10, 2012. 7.2.4

[231] Yabing Liu, Krishna P Gummadi, Balachander Krishnamurthy, and Alan Mislove. Analyzing facebook privacy settings: user expectations vs. reality. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 61–70, 2011. 5.7, 7.10.1

[232] Yabing Liu, Han Hee Song, Ignacio Bermudez, Alan Mislove, Mario Baldi, and Alok Tongaonkar. Identifying personal information in internet traffic. In *Proceedings of the 2015 ACM on Conference on Online Social Networks*, pages 59–70. ACM, 2015. 2.8

[233] Lucidchart. Context data flow diagram template. `https://www.lucidchart.com/pages/templates/data-flow-diagram/context-data-flow-diagram-template`, 2018. (Accessed on 12/22/2018). 7.4, 7.4.3

[234] Sina Madani. Development and evaluation of an ontology-based quality metrics extraction system. 2013. 2.7.1

[235] Miguel Malheiros, Sören Preibusch, and M Angela Sasse. "fairly truthful" : The impact of perceived effort, fairness, relevance, and sensitivity on personal data disclosure. In *International Conference on Trust and Trustworthy Computing*, pages 250–266. Springer, 2013. 7.2.3

[236] Luka Malisa, Kari Kostiainen, and Srdjan Capkun. Detecting mobile application spoofing attacks by leveraging user visual similarity perception. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, CODASPY '17, pages 289–300, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4523-1. doi: 10.1145/3029806.3029819. URL http://doi.acm.org/10.1145/3029806.3029819. 2.9

[237] Nathan Malkin, Julia Bernd, Maritza Johnson, and Serge Egelman. "what can' t data be used for?" privacy expectations about smart tvs in the us. *European Workshop on Usable Security (EuroUSEC)*, 2018. 8.2.1

[238] Anna Maria Mandalari, Daniel J. Dubois, Roman Kolcun, Muhammad Talha Paracha, Hamed Haddadi, and David Choffnes. Blocking without breaking: Identification and mitigation of non-essential iot traffic. *Proceedings on Privacy Enhancing Technologies*, 2021(4):369–388, 2021. doi: doi:10.2478/popets-2021-0075. URL https://doi.org/10.2478/popets-2021-0075. 5.7

[239] Yunlong Mao, Shanhe Yi, Qun Li, Jinghao Feng, Fengyuan Xu, and Sheng Zhong. A privacy-preserving deep learning approach for face recognition with edge computing. In *Proc. USENIX Workshop Hot Topics Edge Comput.(HotEdge)*, pages 1–6, 2018. 5.7

[240] Dennis Mares and Emily Blackburn. Acoustic gunshot detection systems: a quasi-experimental evaluation in st. louis, mo. *Journal of Experimental Criminology*, pages 1–23, 2020. 6.9

[241] Kirsten Martin and Katie Shilton. Putting mobile application privacy in context: An empirical study of user privacy expectations for mobile devices. *The Information Society*, 32(3):200–216, 2016. 2.1, 2.1, 2.2.2, 2.10

[242] Mike Masnick. Court says scraping websites and creating fake profiles can be protected by the first amendment | techdirt. https://www.techdirt.com/articles/20180401/22565539541/court-says-scraping-websites-creating-fake-profiles-can-be-protected-first-amendment.shtml, 2018. (Accessed on 10/24/2018). 2.9

[243] GILES TURNER MATT DAY and NATALIA DROZDIAK. Thousands of amazon workers listen to alexa conversations | time. https://time.com/5568815/amazon-workers-listen-to-alexa/, 04 2019. (Accessed on 02/04/2021). 5.1

[244] Lucas Maystre and Matthias Grossglauser. Fast and accurate inference of plackett–luce models. In *Advances in neural information processing systems*, pages 172–180, 2015. 8.1, 8.3.3, 8.4.3, 8.5

[245] Scott McCloud. Understanding comics: The invisible art. *Northampton, Mass*, 1993. 7.4.4

[246] Aleecia M McDonald and Lorrie Faith Cranor. The cost of reading privacy policies. *Isjlp*, 4:543, 2008. 1.1

[247] Aleecia M. McDonald and Lorrie Faith Cranor. Americans' attitudes about internet behavioral ad-

vertising practices. In *Proceedings of the 9th Annual ACM Workshop on Privacy in the Electronic Society*, WPES '10, pages 63–72, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0096-4. doi: 10.1145/1866919.1866929. URL http://doi.acm.org/10.1145/1866919.1866929. 7.2.3

[248] Matt McFarland. Amazon go: No cashiers, hundreds of cameras, and lots of data - cnn. https://www.cnn.com/2018/10/03/tech/amazon-go/index.html, 03 2018. (Accessed on 02/04/2020). 7.9

[249] Alan McQuinn and Daniel Castro. The costs of an unnecessarily stringent federal data privacy law. Technical report, Information Technology and Innovation Foundation, 2019. 7.1, 7.3.1, 7.9.3

[250] Emily McReynolds, Sarah Hubbard, Timothy Lau, Aditya Saraf, Maya Cakmak, and Franziska Roesner. Toys that listen: A study of parents, children, and internet-connected toys. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 5197–5207. ACM, 2017. 8.2.1

[251] Frank D McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 19–30, 2009. 6.8

[252] MeetingKing. Calculate meeting cost. https://meetingking.com/meeting-cost-calculator/, 03 2020. (Accessed on 03/02/2020). 7.2.1, 7.10.3

[253] Metropolitan Area Planning Council. How to do a parking study –mapc. https://www.mapc.org/resource-library/how-to-do-a-parking-study/, 08 2022. (Accessed on 08/20/2022). 6.1

[254] Piotr Mirowski, Matt Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Andrew Zisserman, Raia Hadsell, et al. Learning to navigate in cities without a map. *Advances in neural information processing systems*, 31, 2018. 6.7.3

[255] Richard Mitev, Anna Pazii, Markus Miettinen, William Enck, and Ahmad-Reza Sadeghi. Leakypick: Iot audio spy detector. In *Annual Computer Security Applications Conference*, ACSAC '20, page 694–705, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450388580. doi: 10.1145/3427228.3427277. URL https://doi.org/10.1145/3427228.3427277. 5.7

[256] Rolf Molich and Jakob Nielsen. Improving a human-computer dialogue. *Commun. ACM*, 33(3): 338–348, March 1990. ISSN 0001-0782. doi: 10.1145/77481.77486. URL http://doi.acm.org/10.1145/77481.77486. 7.5.3

[257] Alex Morehead, Lauren Ogden, Gabe Magee, Ryan Hosler, Bruce White, and George Mohler. Low cost gunshot detection using deep learning on the raspberry pi. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 3038–3044. IEEE, 2019. 6.7.1, 6.7.3

[258] Hajime Morita, Tetsuya Sakai, and Manabu Okumura. Query snowball: a co-occurrence-based ap-

proach to multi-document summarization for question answering. *Information and Media Technologies*, 7(3):1124–1129, 2012. 2.10

[259] Vivian Genaro Motti and Kelly Caine. Users' privacy concerns about wearables. In *International Conference on Financial Cryptography and Data Security*, pages 231–244. Springer, 2015. 7.2.2

[260] Rubén Mulero, Aitor Almeida, Gorka Azkune, Patricia Abril-Jiménez, Maria Teresa Arredondo Waldmeyer, Miguel Páramo Castrillo, Luigi Patrono, Piercosimo Rametta, and Ilaria Sergi. An iot-aware approach for elderly-friendly cities. *IEEE Access*, 6:7941–7957, 2018. doi: 10.1109/ACCESS.2018.2800161. 4.1, 4.1

[261] Stuart Myerscough, Ben Lowe, and Frank Alpert. Willingness to provide personal information online: The role of perceived privacy risk, privacy statements and brand strength. *Journal of Website Promotion*, 2(1-2):115–140, 2008. 7.5

[262] Arvind Narayanan and Bendert Zevenbergen. No encore for encore? ethical questions for web-based censorship measurement. 2015. 7.1

[263] Mohammad Nauman, Sohail Khan, and Xinwen Zhang. Apex: extending android permission model and enforcement with user-defined runtime constraints. In *Proceedings of the 5th ACM symposium on information, computer and communications security*, pages 328–332, 2010. 5.2, 5.7

[264] Rob Neuhaus. rrenaud/gibberish-detector: A small program to detect gibberish using a markov chain. `https://github.com/rrenaud/Gibberish-Detector`, 2018. (Accessed on 01/15/2018). 2.3.1

[265] Annalee Newitz. Dangerous terms: A user's guide to eulas | electronic frontier foundation. `https://www.eff.org/wp/dangerous-terms-users-guide-eulas`, 2005. (Accessed on 10/24/2018). 2.9

[266] Lily Hay Newman. The privacy battle to save google from itself | wired. `https://www.wired.com/story/google-privacy-data/`, 2018. (Accessed on 01/10/2019). 7.1, 7.2.1, 7.9, 10.13

[267] David H Nguyen and Elizabeth D Mynatt. Privacy mirrors: understanding and shaping sociotechnical ubiquitous computing systems. Technical report, Georgia Institute of Technology, 2002. 8.3.1, 8.4.2, 8.6

[268] Jakob Nielsen. Heuristic evaluation: How-to: Article by jakob nielsen. `https://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/`, 1995. (Accessed on 01/04/2019). 7.7

[269] Jakob Nielsen. Applying discount usability engineering. *IEEE software*, 12(1):98–100, 1995. 7.1

[270] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '90, pages 249–256, New York, NY, USA,

1990. ACM. ISBN 0-201-50932-6. doi: 10.1145/97243.97281. URL http://doi.acm.org/10.1145/97243.97281. 7.1, 7.1.1, 7.5.1, 7.5.3, 7.7, 7.7, 7.9, 7.9.3

[271] Helen Nissenbaum. Privacy as contextual integrity. *Wash. L. Rev.*, 79:119, 2004. 1.3.1, 2.10

[272] Helen Nissenbaum. *Privacy in context: Technology, policy, and the integrity of social life*. Stanford University Press, 2009. 2.1, 2.10, 7.2.3, 7.4.4

[273] NoFlo. Noflo | flow-based programming for javascript. https://noflojs.org/, 12 2021. (Accessed on 12/19/2021). 6.1

[274] NoFlo. Flow-based programming for javascript. https://noflojs.org/, 08 2022. (Accessed on 08/16/2022). 6.5

[275] Peter Norvig. Natural language corpus data. 2009. 2.3.1

[276] Yasunari Obuchi. Pda speech database. http://www.speech.cs.cmu.edu/databases/pda/README.html, 03 2003. (Accessed on 11/15/2020). 5.6.2

[277] National Institute of Standards and Technology. Draft nistir 8062: An introduction to privacy engineering and risk management in federal systems. 15:2015, 2015. 7.1, 7.2.1, 4, 7.3.1, 7.4.3, 8, 7.5.2, 7.5, 7.6, 7.6.2

[278] National Institute of Standards and Technology. Nistir 8062: An introduction to privacy engineering and risk management in federal systems. 15:2015, 2017. 7.2.1, 7.5

[279] Information Commissioner's Office. Data protection impact assessments (dpias) | ico. https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/data-protection-impact-assessments-dpias/, 12 2020. (Accessed on 12/30/2020). 7.2.1

[280] Hyeontaek Oh, Sangdon Park, Gyu Myoung Lee, Hwanjo Heo, and Jun Kyun Choi. Personal data trading scheme for data brokers in iot data marketplaces. *IEEE Access*, 7:40120–40132, 2019. 6.1

[281] Cathy O'Neil. China knows how to take away your health insurance - bloomberg. https://www.bloomberg.com/opinion/articles/2019-06-14/china-knows-how-to-take-away-your-health-insurance, 06 2019. (Accessed on 02/04/2020). 7.9

[282] Mark O'Neill, Scott Ruoti, Kent Seamons, and Daniel Zappala. Tls inspection: How often and who cares? *IEEE Internet Computing*, 21(3):22–29, 2017. doi: 10.1109/MIC.2017.58. 5.9

[283] Stacey A Page and Jeffrey Nyeboer. Improving the process of research ethics review. *Research integrity and peer review*, 2(1):1–7, 2017. 7.10.3

[284] Shoumik Palkar and Matei Zaharia. Diy hosting for online privacy. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pages 1–7, 2017. 5.7, 6.8

[285] Valentina Palladino. Teardown shows nest cam is "always-on" even when you think it's off | ars technica. https://arstechnica.com/gadgets/2015/11/teardown-shows-nest-cam-is-always-on-even-when-you-think-its-off/, 11 2015. (Accessed on 01/24/2021). 5.1, 5.7

[286] R. Michael Varney Pamela S. Hrubey. Privacy and data protection: Internal audit's role in establishing a resilient framework. https://www.crowe.com/-/media/Crowe/LLP/folio-pdf-hidden/Privacy_and_Data_Protection_Crowe_IIA_IAF_Joint_Report_CC2015-006.pdf?la=en-US&modified=20200407161139&hash=4F7360FDC3C61D820622DA34FC448C5B1E6F7877, 01 2020. (Accessed on 12/28/2020). 7.1

[287] Jeff Pan. Reverse engineering the airbnb api. https://medium.com/switchcm/reverse-engineering-the-airbnb-api-c0f5cb609b, 2017. (Accessed on 11/04/2017). 2.1.1, 2.6

[288] Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. Whyper: Towards automating risk assessment of mobile applications. 2.12, 2.10

[289] Seung-min Park, Daeyoun D Won, Brian J Lee, Diego Escobedo, Andre Esteva, Amin Aalipour, T Jessie Ge, Jung Ha Kim, Susie Suh, Elliot H Choi, et al. A mountable toilet system for personalized health monitoring via the analysis of excreta. *Nature biomedical engineering*, pages 1–12, 2020. 3.1

[290] Edoardo Patti and Andrea Acquaviva. Iot platform for smart cities: Requirements and implementation case studies. In *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, pages 1–6, 2016. doi: 10.1109/RTSI.2016.7740618. 4.1, 4.1

[291] Clare Payne. Bad people, or bad decisions? | investment magazine. https://www.investmentmagazine.com.au/2016/11/bad-people-or-bad-decisions/, 11 2016. (Accessed on 04/26/2020). 7.1, 7.11.1

[292] Dan Pearson. Riot uses lol chatlogs to weed out toxic employees | gamesindustry.biz. https://www.gamesindustry.biz/articles/2016-06-10-riot-uses-lol-chatlogs-to-weed-out-toxic-employees, 06 2016. (Accessed on 02/04/2020). 7.9

[293] Riccardo Petrolo, Valeria Loscri, and Nathalie Mitton. Towards a smart city based on cloud of things, a survey on the smart city vision and paradigms. *Transactions on emerging telecommunications technologies*, 28(1):e2931, 2017. 4.1, 4.1

[294] Andreas Pfitzmann and Marit Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management, 2010. 1, 1.3.1

[295] Angelisa C Plane, Elissa M Redmiles, Michelle L Mazurek, and Michael Carl Tschantz. Exploring user perceptions of discrimination in online targeted advertising. In *USENIX Security*, 2017. 7.5.2

[296] PolarProxy. Polarproxy - a transparent tls proxy created primarily for incident responders and malware researchers. https://www.netresec.com/?page=PolarProxy, 08 2021. (Accessed on 08/19/2021). 5.9

[297] Alisha Pradhan, Kanika Mehta, and Leah Findlater. " accessibility came by accident" use of voice-controlled intelligent personal assistants by people with disabilities. In *Proceedings of the 2018 CHI Conference on human factors in computing systems*, pages 1–13, 2018. 8.2.2

[298] CMU PrivacyGrade. Grading the privacy of smartphone apps, 2015. 7.5

[299] Zhengyang Qu, Vaibhav Rastogi, Xinyi Zhang, Yan Chen, Tiantian Zhu, and Zhong Chen. Autocog: Measuring the description-to-permission fidelity in android applications. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1354–1365. ACM, 2014. 2.12, 2.10

[300] Amir Rahmati, Earlence Fernandes, Kevin Eykholt, Xinheng Chen, and Atul Prakash. Heimdall: A privacy-respecting implicit preference collection framework. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '17, page 453–463, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349284. doi: 10.1145/3081333.3081334. URL https://doi.org/10.1145/3081333.3081334. 5.8

[301] Leena Rao. Sexual activity tracked by fitbit shows up in google search results | techcrunch. https://techcrunch.com/2011/07/03/sexual-activity-tracked-by-fitbit-shows-up-in-google-search-results/, 07 2011. (Accessed on 02/04/2020). 7.9

[302] Vaibhav Rastogi, Yan Chen, and William Enck. Appsplayground: automatic security analysis of smartphone applications. In *Proceedings of the third ACM conference on Data and application security and privacy*, pages 209–220. ACM, 2013. 2.12

[303] Nisarg Raval, Animesh Srivastava, Ali Razeen, Kiron Lebeck, Ashwin Machanavajjhala, and Landon P Cox. What you mark is what apps see. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 249–261, 2016. 5.7, 6.1, 6.9

[304] Abbas Razaghpanah, Rishab Nithyanand, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Mark Allman, Christian Kreibich, and Phillipa Gill. Apps, trackers, privacy, and regulators: A global study of

the mobile tracking ecosystem. 2018. 2.8, 2.9, 6.2

[305] Reddit. Anyway to limit guests access to a shared device ie they can't hear sound, cams only available on a schedule i set etc? : wyzecam. https://www.reddit.com/r/wyzecam/comments/eqqqtq/anyway_to_limit_guests_access_to_a_shared_device/, 09 2021. (Accessed on 09/09/2021). 8.3.1

[306] Reddit. Guest mode: An easy privacy control for your home devices. https://www.reddit.com/r/googlehome/comments/kwm1k2/guest_mode_an_easy_privacy_control_for_your_home/, 09 2021. (Accessed on 09/04/2021). 8.4.2

[307] Reddit. Is there a robot vacuum cleaner that does not violate privacy? : privacy. https://www.reddit.com/r/privacy/comments/eruwpe/is_there_a_robot_vacuum_cleaner_that_does_not/, 09 2021. (Accessed on 09/09/2021). 8.3.1

[308] Reddit. Recommendations for a home camera respecting my privacy. https://www.reddit.com/r/homesecurity/comments/ctuvnx/recommendations_for_a_home_camera_respecting_my/, 09 2021. (Accessed on 09/07/2021). 8.1

[309] Reddit. Wyze cam into a smart plug? https://www.reddit.com/r/wyzecam/comments/7u9jy6/wyze_cam_into_a_smart_plug/, 08 2021. (Accessed on 08/10/2021). 8.1, 8.3.1, 8.4.2

[310] Reddit. use of wyze cams inside your home. https://www.reddit.com/r/wyzecam/comments/kochtt/use_of_wyze_cams_inside_your_home/, 09 2021. (Accessed on 09/07/2021). 8.1

[311] Elissa M Redmiles, Sean Kross, and Michelle L Mazurek. How well do my results generalize? comparing security and privacy survey results from mturk, web, and telephone samples. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1326–1343. IEEE, 2019. 8.5

[312] Michael Reilly. Is facebook targeting ads at sad teens? - mit technology review. https://www.technologyreview.com/s/604307/is-facebook-targeting-ads-at-sad-teens/, 05 2017. (Accessed on 02/17/2020). 7.8.1, 7.9, 10.13

[313] Eric Reis. The lean startup. *New York: Crown Business*, page 27, 2011. 7.1

[314] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David Choffnes. Recon: Revealing and controlling pii leaks in mobile network traffic. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 361–374. ACM, 2016. 2.1, 2.3, 2.11, 2.8, 2.8

[315] RescueTime. Rescuetime: Automatic time-tracking software. https://www.rescuetime.com/, 11 2020. (Accessed on 11/15/2020). 5.4

[316] Irwin Reyes, Primal Wiesekera, Abbas Razaghpanah, Joel Reardon, Narseo Vallina-Rodriguez, Serge

Egelman, and Christian Kreibich. " is our children's apps learning?" automatically detecting coppa violations. 2017. 2.9

[317] Matthew Richardson, Ewa Dominowska, and Robert Ragno. Predicting clicks: estimating the click-through rate for new ads. In *Proceedings of the 16th international conference on World Wide Web*, pages 521–530. ACM, 2007. 7.4.3

[318] Luke Rolka. Infographic: How to solve the online registration challenge | janrain. https://www.janrain.com/blog/infographic-how-solve-online-registration-challenge, 2012. (Accessed on 10/23/2018). 2.9

[319] Mary Beth Rosson, John M Carroll, and Natalie Hill. *Usability engineering: scenario-based development of human-computer interaction*. Morgan Kaufmann, 2002. 2

[320] Spencer Rothwell, Steele Carter, Ahmad Elshenawy, and Daniela Braga. Job complexity and user attention in crowdsourcing microtasks. In *Third AAAI Conference on Human Computation and Crowdsourcing*, 2016. 7.5.4

[321] Indrajit Roy, Srinath TV Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. Airavat: Security and privacy for mapreduce. In *NSDI*, volume 10, pages 297–312, 2010. 6.8

[322] Richard L Rutledge, Aaron K Massey, and Annie I Antón. Privacy impacts of iot devices: A smarttv case study. In *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, pages 261–270. IEEE, 2016. 8.2.1

[323] Samsung. One simple home system. a world of possibilities. | smartthings. https://www.smartthings.com/, 08 2021. (Accessed on 08/19/2021). 5.3.1

[324] SAN FRANCISCO COUNTY TRANSPORTATION AUTHORITY. Parking_supply_final_report_11.29.16.pdf. https://www.sfcta.org/sites/default/files/2019-03/Parking_Supply_final_report_11.29.16.pdf, 08 2022. (Accessed on 08/08/2022). 6.1

[325] Ruben Sánchez-Corcuera, Adrián Nuñez-Marcos, Jesus Sesma-Solance, Aritz Bilbao-Jayo, Rubén Mulero, Unai Zulaika, Gorka Azkune, and Aitor Almeida. Smart cities survey: Technologies, application domains and challenges for the cities of the future. *International Journal of Distributed Sensor Networks*, 15(6):1550147719853984, 2019. 4.1, 4.1

[326] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 5.5, 5.6.3

[327] Shruti Sannon, Natalya N Bazarova, and Dan Cosley. Privacy lies: Understanding how, when, and why people lie to protect their privacy in multiple online contexts. In *Proc. CHI*, pages 1–13, 2018.

8.1, 8.2.2

[328] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4):14–23, 2009. 5.7

[329] Stuart Schechter and Cristian Bravo-Lillo. Ethical-response survey report: Fall 2014. Technical report, Technical Report MSR-TR-2014-140, 2014. 3

[330] Stuart Schechter and Cristian Bravo-Lillo. Using ethical-response surveys to identify sources of disapproval and concern with facebook's emotional contagion experiment and other controversial studies. 2014. 7.2.2, 7.2.3, 7.3.1, 1, 7, 7.4.2, 7.5.3, 7.6, 7.10.1

[331] NOAM SCHEIBER. How uber uses psychological tricks to push its drivers' buttons. https://www.nytimes.com/interactive/2017/04/02/technology/uber-drivers-psychological-tricks.html, 2017. (Accessed on 07/08/2018). 7.1

[332] Johann Schleier-Smith, Vikram Sreekanti, Anurag Khandelwal, Joao Carreira, Neeraja J Yadwadkar, Raluca Ada Popa, Joseph E Gonzalez, Ion Stoica, and David A Patterson. What serverless computing is and should become: The next phase of cloud computing. *Communications of the ACM*, 64(5): 76–84, 2021. 6.1, 6.4.1

[333] Awanthika Senarath and Nalin Asanka Gamagedara Arachchilage. Understanding software developers' approach towards implementing data minimization. *arXiv preprint arXiv:1808.01479*, 2018. 1, 1.3.1

[334] Kim Bartel Sheehan. Toward a typology of internet users and online privacy concerns. *The Information Society*, 18(1):21–32, 2002. 7.1, 7.5.2, 7.5

[335] Gunnar A Sigurdsson, Gül Varol, Xiaolong Wang, Ali Farhadi, Ivan Laptev, and Abhinav Gupta. Hollywood in homes: Crowdsourcing data collection for activity understanding. In *European Conference on Computer Vision*, pages 510–526. Springer, 2016. 5.6.2

[336] Michael Skirpan and Tom Yeh. Designing a moral compass for the future of computer vision using speculative analysis. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*, pages 1368–1377. IEEE, 2017. 1

[337] Smartdraw. Data flow diagram - everything you need to know about dfd. https://www.smartdraw.com/data-flow-diagram/, 2018. (Accessed on 12/22/2018). 7.4, 7.4.3

[338] Objective Development Software. Little snitch 4: Makes the invisible visible! https://www.obdev.at/products/littlesnitch/index.html, 2017. (Accessed on 07/28/2017). 2.8

[339] Daniel J Solove. A taxonomy of privacy. *U. Pa. L. Rev.*, 154:477, 2005. 1.3.1, 7.4.3, 7.4.3, 7.5.2, 7.5

[340] Yihang Song and Urs Hengartner. Privacyguard: A vpn-based platform to detect information leakage

on android devices. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 15–26. ACM, 2015. 2.1, 2.3, 2.11, 2.8

[341] Sarah Spiekermann. The challenges of privacy by design. *Communications of the ACM*, 55(7):38–40, 2012. 7.1

[342] Gaurav Srivastava, Saksham Chitkara, Kevin Ku, Swarup Kumar Sahoo, Matt Fredrikson, Jason Hong, and Yuvraj Agarwal. Privacyproxy: Leveraging crowdsourcing and in situ traffic analysis to detect and mitigate information leakage. *arXiv preprint arXiv:1708.06384*, 2017. 2.1, 2.1.1, 2.11, 2.8, 2.9

[343] Daniel H Stolfi, Enrique Alba, and Xin Yao. Predicting car park occupancy rates in smart cities. In *International Conference on Smart Cities*, pages 107–117. Springer, 2017. 6.1, 6.1

[344] Peter Story, Daniel Smullen, Yaxing Yao, Alessandro Acquisti, Lorrie Faith Cranor, Norman Sadeh, and Florian Schaub. Awareness, adoption, and misconceptions of web privacy tools. *UMBC Faculty Collection*, 2021. 8.1

[345] Android Studio. Ui/application exerciser monkey. `https://developer.android.com/studio/test/monkey.html`, 2018. (Accessed on 02/02/2018). 2.1, 2.1.1

[346] Madiha Tabassum, Tomasz Kosiński, Alisa Frik, Nathan Malkin, Primal Wijesekera, Serge Egelman, and Heather Richter Lipford. Investigating users' preferences and expectations for always-listening voice assistants. *Proc. ACM IMWUT*, 3(4):1–23, 2019. 8.2.2

[347] Madiha Tabassum, Jess Kropczynski, Pamela Wisniewski, and Heather Richter Lipford. Smart home beyond the home: A case for community-based access control. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2020. 8.2.2

[348] Joshua Tan, Khanh Nguyen, Michael Theodorides, Heidi Negrón-Arroyo, Christopher Thompson, Serge Egelman, and David Wagner. The effect of developer-specified explanations for permission requests on smartphone user behavior. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 91–100, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2473-1. doi: 10.1145/2556288.2557400. URL `http://doi.acm.org/10.1145/2556288.2557400`. 2.1

[349] Adam Tanner. *What Stays in Vegas: The World of Personal Data—Lifeblood of Big Business—and the End of Privacy as We Know It*. PublicAffairs, 2014. 7.1

[350] Stéphane Cédric Koumetio Tekouabou, Walid Cherif, Hassan Silkan, et al. Improving parking availability prediction in smart cities with iot and ensemble-based model. *Journal of King Saud University-Computer and Information Sciences*, 2020. 6.1

[351] template.com. Eula template and generator - free and for 2018. `https://eulatemplate.com/`,

2018. (Accessed on 10/24/2018). 2.9

[352] Prachi Thacker. Graphql server with node.js. over the past decade, rest has become⋯| by prachi thacker | the startup | medium. https://medium.com/swlh/graphql-server-with-node-js-5a031881d4b8, 10 2020. (Accessed on 09/24/2022). 9.2

[353] Gwen Thomas. The dgi data governance framework. *The Data Governance Institute, Orlando, FL (USA)*, 20, 2006. 7.4.3

[354] Kurt Thomas, Devdatta Akhawe, Michael Bailey, Dan Boneh, Elie Bursztein, Sunny Consolvo, Nicola Dell, Zakir Durumeric, Patrick Gage Kelley, Deepak Kumar, et al. Sok: Hate, harassment, and the changing landscape of online abuse. 2021. 6.2

[355] Ken Thompson. Reflections on trusting trust. *Communications of the ACM*, 27(8):761–763, 1984. 6.3.3

[356] ThoughtWorks. Security sandwich | technology radar | thoughtworks. https://www.thoughtworks.com/radar/techniques/security-sandwich, 01 2015. (Accessed on 04/25/2020). 7.1

[357] Tripadvisor. Number of characters for review - tripadvisor support forum. https://www.tripadvisor.com/ShowTopic-g1-i12105-k12375287-Number_of_characters_for_review-Tripadvisor_Support.html, 03 2019. (Accessed on 02/17/2020). 7.5.3

[358] Khai N Truong, Elaine M Huang, Molly M Stevens, and Gregory D Abowd. How do users think about ubiquitous computing? In *CHI'04 Extended Abstracts on Human Factors in Computing Systems*, pages 1317–1320. ACM, 2004. 2

[359] Khai N Truong, Gillian R Hayes, and Gregory D Abowd. Storyboarding: an empirical determination of best practices and effective guidelines. In *Proceedings of the 6th conference on Designing Interactive systems*, pages 12–21. ACM, 2006. 2, 7.4.2, 7.4.4, 7.12.3

[360] Blase Ur, Pedro Giovanni Leon, Lorrie Faith Cranor, Richard Shay, and Yang Wang. Smart, useful, scary, creepy: Perceptions of online behavioral advertising. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 4:1–4:15, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1532-6. doi: 10.1145/2335356.2335362. URL http://doi.acm.org/10.1145/2335356.2335362. 7.2.3

[361] U.S. Office of Victim Services and Justice Grants. Private security camera system incentive program | ovsjg. https://ovsjg.dc.gov/service/private-security-camera-system-incentive-program, 10 2021. (Accessed on 10/22/2021). 6.1

[362] Lorenzo Valerio, Andrea Passarella, and Marco Conti. Accuracy vs. traffic trade-off of learning iot data patterns at the edge with hypothesis transfer learning. In *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, pages 1–6. IEEE, 2016. 5.7

[363] Max Van Kleek, Ilaria Liccardi, Reuben Binns, Jun Zhao, Daniel J Weitzner, and Nigel Shadbolt. Better the devil you know: Exposing the data sharing practices of smartphone apps. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 5208–5220, 2017. 2.1, 2.2.1, 2.1, 2.2.2, 2.5, 2.10, 2.11

[364] Lawrence G Votta Jr. Does every inspection need a meeting? In *Proceedings of the 1st ACM SIGSOFT symposium on Foundations of software engineering*, pages 107–114, 1993. 7.2.1, 7.10.3

[365] Vox. The facebook and cambridge analytica scandal, explained with a simple diagram. `https://www.vox.com/policy-and-politics/2018/3/23/17151916/facebook-cambridge-analytica-trump-diagram`, 2018. (Accessed on 09/26/2018). 7.1, 7.1

[366] Frank Wang, Ronny Ko, and James Mickens. Riverbed: Enforcing user-defined privacy constraints in distributed web services. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, pages 615–630, 2019. 5.1, 6.8

[367] Haoyu Wang, Jason Hong, and Yao Guo. Using text mining to infer the purpose of permission use in mobile apps. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 1107–1118. ACM, 2015. 2.1, 2.1, 2.6, 2.7.1, 2.7.2, 2.12, 2.8, 2.10

[368] Jiayu Wang and Qigeng Chen. Aspg: Generating android semantic permissions. In *Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on*, pages 591–598. IEEE, 2014. 2.12, 2.10

[369] Junjue Wang, Brandon Amos, Anupam Das, Padmanabhan Pillai, Norman Sadeh, and Mahadev Satyanarayanan. A scalable and privacy-aware iot service for live video analytics. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, pages 38–49. ACM, 2017. 5.1, 5.7, 6.1, 6.9

[370] Xu Wang, Srinivasa Teja Talluri, Carolyn Rose, and Kenneth Koedinger. Upgrade: Sourcing student open-ended solutions to create scalable learning opportunities. In *Proceedings of the Sixth (2019) ACM Conference on Learning@ Scale*, pages 1–10, 2019. 7.12.5

[371] Yang Wang, Gregory Norcie, Saranga Komanduri, Alessandro Acquisti, Pedro Giovanni Leon, and Lorrie Faith Cranor. " i regretted the minute i pressed share" a qualitative study of regrets on facebook. In *Proceedings of the seventh symposium on usable privacy and security*, pages 1–16, 2011. 7.2.2

[372] Yang Wang, Gregory Norice, and Lorrie Faith Cranor. Who is concerned about what? a study

of american, chinese and indian users' privacy concerns on social network sites. In *International Conference on Trust and Trustworthy Computing*, pages 146–153. Springer, 2011. 7.2.3

[373] Fengguo Wei, Sankardas Roy, Xinming Ou, et al. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1329–1341. ACM, 2014. 2.12

[374] Zhou Wei. What your face may tell lenders about whether you're creditworthy - wsj. https://www.wsj.com/articles/what-your-face-may-tell-lenders-about-whether-youre-creditworthy-11560218700, 06 2019. (Accessed on 02/04/2020). 7.9

[375] Ryan Welton. Fuzion24/justtrustme: An xposed module that disables ssl certificate checking for the purposes of auditing an app with cert pinning. https://github.com/Fuzion24/JustTrustMe, 2018. (Accessed on 02/11/2018). 2.9

[376] Western City. The "front page" test: An easy ethics standard - western city magazine. https://www.westerncity.com/article/front-page-test-easy-ethics-standard, 02 2012. (Accessed on 04/25/2020). 7.1

[377] Monica T Whitty and Siobhan E Carville. Would i lie to you? self-serving lies and other-oriented lies told across different media. *Computers in Human Behavior*, 24(3):1021–1031, 2008. 8.2.2

[378] Wikipedia. Card sorting - wikipedia. https://en.wikipedia.org/wiki/Card_sorting, 2018. (Accessed on 08/15/2018). 2.2.2

[379] Wikipedia. Discounted cumulative gain - wikipedia. https://en.wikipedia.org/wiki/Discounted_cumulative_gain, 03 2020. (Accessed on 03/03/2020). 7.9.3

[380] Wikipedia. Separation of duties. https://en.wikipedia.org/wiki/Separation_of_duties, 04 2020. (Accessed on 04/23/2020). 7.4.4

[381] Wikipedia. Levenshtein distance. https://en.wikipedia.org/wiki/Levenshtein_distance, 02 2021. (Accessed on 02/05/2021). 5.6.2

[382] Wikipedia. Von restorff effect - wikipedia. https://en.wikipedia.org/wiki/Von_Restorff_effect, 09 2021. (Accessed on 09/08/2021). 8.4.3, 8.6.4

[383] Wikipedia. Matter (standard) - wikipedia. https://en.wikipedia.org/wiki/Matter_(standard), 08 2022. (Accessed on 08/10/2022). 6.2

[384] Wikipedia. Serverless computing - wikipedia. https://en.wikipedia.org/wiki/Serverless_computing, 05 2022. (Accessed on 05/31/2022). 6.4.1

[385] Wikipedia. Simple network management protocol - wikipedia. https://en.wikipedia.org/wiki/Simple_Network_Management_Protocol, 01 2022. (Accessed on 01/06/2022). 8.6.2

[386] Wikipedia. True-range multilateration - wikipedia. https://en.wikipedia.org/wiki/True-range_multilateration, 08 2022. (Accessed on 08/20/2022). 6.7.1

[387] Jenifer S Winter. Privacy and the emerging internet of things: using the framework of contextual integrity to inform policy. In *Pacific telecommunication council conference proceedings*, volume 2012, 2012. 7.2.3

[388] Wireshark. Wireshark · go deep. https://www.wireshark.org/, 2017. (Accessed on 08/28/2017). 2.8

[389] Yongkang Wong, Shaokang Chen, Sandra Mau, Conrad Sanderson, and Brian C Lovell. Patch-based probabilistic image quality assessment for face selection and improved video-based face recognition. In *CVPR 2011 WORKSHOPS*, pages 74–81. IEEE, 2011. 5.6.2

[390] Yongkang Wong, Shaokang Chen, Sandra Mau, Conrad Sanderson, and Brian C. Lovell. Patch-based probabilistic image quality assessment for face selection and improved video-based face recognition. In *IEEE Biometrics Workshop, Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 81–88. IEEE, June 2011. 5.6.3

[391] Molly Wood. Okcupid plays with love in user experiments - the new york times. https://www.nytimes.com/2014/07/29/technology/okcupid-publishes-findings-of-user-experiments.html?ref=technology&_r=1, 2014. (Accessed on 07/06/2018). 7.1, 7.1, 7.9

[392] Allison Woodruff, Vasyl Pihur, Sunny Consolvo, Lauren Schmidt, Laura Brandimarte, and Alessandro Acquisti. Would a privacy fundamentalist sell their dna for $1000... if nothing bad happened as a result? the westin categories, behavioral intentions, and consequences. In *Symposium on Usable Privacy and Security (SOUPS)*, volume 5, page 1, 2014. 7.1.1, 7.2.2, 7.2.3, 7.4.1, 1, 7.4, 7.4.3, 10, 7.4.4, 7.5.4, 7.9, 7.10, 7.9, 7.10.1

[393] Peter Worthy, Ben Matthews, and Stephen Viller. Trust me: doubts and concerns living with the internet of things. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*, pages 427–434. ACM, 2016. 8.2.1

[394] David Wright and Paul De Hert. *Privacy impact assessment*, volume 6. Springer Science & Business Media, 2011. 7.1, 7.2.1, 7.3.1, 7.5.2, 7.6

[395] Yaxing Yao, Davide Lo Re, and Yang Wang. Folk models of online behavioral advertising. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, pages 1957–1969, 2017. 8.2.2

[396] Yaxing Yao, Justin Reed Basdeo, Smirity Kaushik, and Yang Wang. Defending my castle: A co-design study of privacy mechanisms for smart homes. In *Proc. CHI*, pages 1–12, 2019. 8.2.2

[397] Anil Yelam, Shibani Subbareddy, Keerthana Ganesan, Stefan Savage, and Ariana Mirian. Coresident evil: Covert communication in the cloud with lambdas. In *Proceedings of the Web Conference 2021*, pages 1005–1016, 2021. 6.2

[398] Hyunwoo Yu, Jaemin Lim, Kiyeon Kim, and Suk-Bok Lee. Pinto: enabling video privacy for commodity iot cameras. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1089–1101, 2018. 5.1, 5.7

[399] Jinyan Zang, Krysta Dummit, James Graves, Paul Lisker, and Latanya Sweeney. Who knows what about me? a survey of behind the scenes personal data sharing to third parties by mobile apps. *Technology Science*, 30, 2015. 2.1, 2.11

[400] Eric Zeng and Franziska Roesner. Understanding and improving security and privacy in multi-user smart homes: a design exploration and in-home user study. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 159–176, 2019. 8.2.2

[401] Eric Zeng, Shrirang Mare, and Franziska Roesner. End user security & privacy concerns with smart homes. In *Symposium on Usable Privacy and Security (SOUPS)*, 2017. 8.2.1

[402] Han Zhang, Abhijith Anilkumar, Matt Fredrikson, and Yuvraj Agarwal. Capture: Centralized library management for heterogeneous iot devices. In *USENIX Security Symposium*, 2021. 5.7

[403] Hao Zhang, William Banick, Danfeng Yao, and Naren Ramakrishnan. User intention-based traffic dependence analysis for anomaly detection. In *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on*, pages 104–112. IEEE, 2012. 2.1, 2.11, 2.8

[404] Serena Zheng, Noah Apthorpe, Marshini Chetty, and Nick Feamster. User perceptions of smart home iot privacy. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW):1–20, 2018. 8.2.1

[405] Jingya Zhou, Jianxi Fan, and Jin Wang. Task scheduling for mobile edge computing enabled crowd sensing applications. *International Journal of Sensor Networks*, 35(2):88–98, 2021. 6.7.4

[406] Verena Zimmermann, Merve Bennighof, Miriam Edel, Oliver Hofmann, Judith Jung, and Melina von Wick. 'home, smart home'–exploring end users' mental models of smart homes. *Mensch und Computer 2018-Workshopband*, 2018. 8.2.1

[407] Zoom. Setting up calendar and contacts integration for the desktop client - zoom help center. https://support.zoom.us/hc/en-us/articles/360000488243-Setting-up-calendar-and-contacts-integration-for-the-desktop-client, 09 2021. (Accessed on 09/29/2021). 1.1

[408] Yixin Zou, Kevin Roundy, Acar Tamersoy, Saurabh Shintre, Johann Roturier, and Florian Schaub. Examining the adoption and abandonment of security, privacy, and identity theft protection practices. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–15, 2020. 8.1