Providing Architectural Support for Building Privacy-Sensitive Smart Home Applications

Haojian Jin
haojian@cs.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Swarun Kumar
swarun@cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Jason Hong
jasonh@cs.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

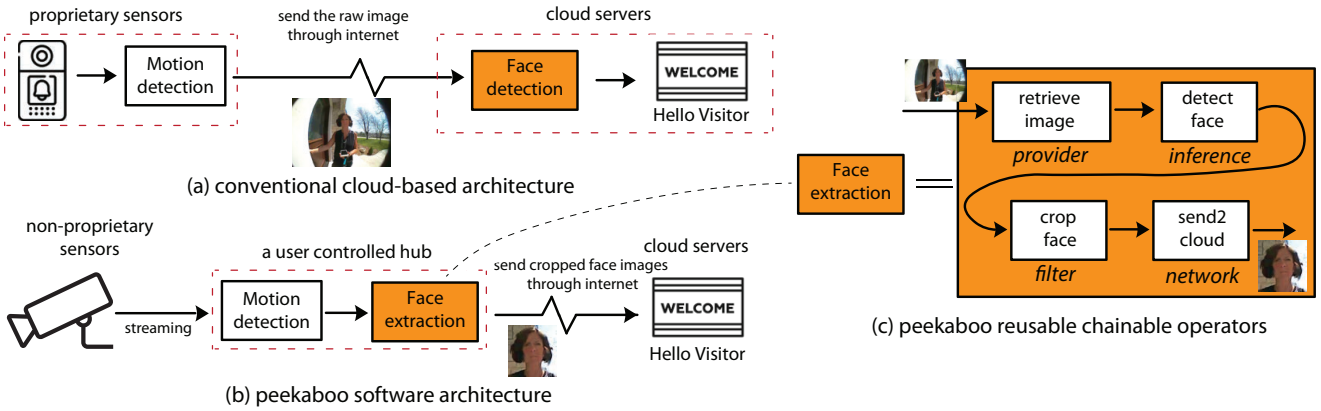

Figure 1: Peekaboo factors out the repetitive data preprocessing tasks (e.g., face detection) from the cloud side onto a smart home hub (a & b), supports them as a set of reusable, chainable operators (c), and then runs them over the sensor data before they flow to cloud services (and out of the users' control). In doing so, Peekaboo reduces unnecessary data egress, lowers the barriers for app developers, and offers users a unified and centralized nexus for privacy awareness and control.

ABSTRACT

In this thesis, we plan to introduce a new IoT app development framework named Peekaboo, which aims to make it much easier for developers to get the granularity of data they actually need rather than always requesting raw data, while also offering architecture support for building privacy features across all the apps. Peekaboo's architectural design philosophy is to factor out repetitive data pre-processing tasks (e.g., face detection, frequency spectrum extraction) from the cloud side onto a user-controlled hub, and support them as a fixed set of open source, reusable, and chainable operators. These operators pre-process raw data to remove unneeded sensitive user information before the data flow to the cloud (and out of the users' control), thus reducing data egress and many potential privacy risks for users. Further, all the IoT apps built with Peekaboo share a common structure of the chainable operators, making it possible to build consistent privacy features beyond individual apps.

CCS CONCEPTS

• **Human-centered computing** ■ *Human computer interaction (HCI)*; • **Security and privacy** ■ **Human and societal aspects of security and privacy**; • **Software and its engineering** ■ **Software architectures**.

KEYWORDS

Ubiquitous computing; privacy; toolkit; software architecture; smart home; IoT app development

ACM Reference Format:
Haojian Jin, Swarun Kumar, and Jason Hong. 2020. Providing Architectural Support for Building Privacy-Sensitive Smart Home Applications. In *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers (UbiComp/ISWC '20 Adjunct), September 12–16, 2020,*

ACM ISBN 978-1-4503-8076-8/20/09.
https://doi.org/10.1145/3410530.3414328

Virtual Event, Mexico. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3410530.3414328

1 PROBLEM STATEMENT

A major challenge for deploying smart home apps is earning the trust of end-users [7, 11, 16]. For many smart home apps, such as smart speakers, cameras, thermostats, and occupancy sensors, the "brains" of these systems are in the cloud. On-board sensors collect data and send them to remote servers, where primary functionality (such as commands, analytics, or sharing) typically happen. This kind of cloud-based software architecture reduces the cost of IoT hardware, allows running computationally expensive algorithms, eases system/feature upgrades, and allows for subscription-based business models. However, a major drawback of this kind of architecture is privacy. Users have little control over the data transmission process, and in many cases, must either accept the fact that the proprietary hardware may collect raw data continuously or simply not purchase these smart devices at all [7].

Earning the trust of end-users is challenging for at least two reasons. First, protecting privacy in a smart home is hard because of the fluid and dynamic nature [8, 15, 17]. For example, privacy can be leaked from any of the distributed and heterogeneous sensor sources. An elderly person may be willing to install a 24x7 camera, sacrificing the privacy for the chance to live independently and safely [1, 18]. Designing a practical privacy-sensitive app requires a deep understanding of the use scenario and significant efforts [12].

Second, while privacy is a fundamental right to users, we have to acknowledge that it is not the "core value" on par with functionality, security, reliability, and the cost for developers [9]. Privacy features in smart home apps are typically built using an ad-hoc process. App developers choose whatever technique is easiest to implement, usually dictated by the existing software architecture and sensors (see §2).

This thesis aims to make privacy-sensitive app development easier by offering a new kind of system architecture for smart home apps. The proposed framework, Peekaboo, will allow developers to expend less effort on the common privacy features across most smart home apps and focus their energies on the primary goal of these apps, that is, enabling the smart automation through the sensor data.

2 BACKGROUND & RELATED WORK

In this section, we review previously built smart home applications to understand how they incorporate privacy-relevant features. Through a comprehensive analysis of published academic and industrial solutions spanning three decades, we find three types of paradigms[1].

The **end-to-end development** (e.g., Nest Camera, responsive environment [14], smart microwave [13]) is the most common approach nowadays, in which developers build the proprietary hardware and a complete stack of software from the firmware to the cloud programs and mobile apps. The manufacturer can control how every aspect of the product works, making it possible to optimize user experience (e.g., setup and management), power consumption, and hardware utilization. Developers can profit from selling the physical hardware (e.g., Nest Camera) and offering add-on services (e.g., Nest Aware [5]). The privacy model for these products rely on the self-regulation. Products may offer some privacy control and users can operate based on the available options. For example, Nest Cameras allow users to turn off the camera during a specified time period.

An alternative approach is building apps through the **interoperability protocols** (e.g., IFTTT, Apple HomeKit), which stitches cross-platform services through network protocols (e.g., Apple MFi). Since this paradigm does not control the privacy policies of third parties, it offers limited privacy protection. For example, IFTTT encourages users to ask questions to the service providers before disclosing their personal information [10].

The **capability abstraction** approach (e.g., HomeOS [2], Samsung SmartThing is built upon the interoperability protocols approach, which aims to construct a unified interface for home technologies. HomeOS encapsulates the interoperability protocols into standard system APIs so that developers can develop smart home apps in a similar way as mobile/PC app development. The privacy model for the operating system approach still relies on app developer self-regulation. But it improves the full-stack method because it offers a centralized location for end-users to manage multiple apps.

As we can see, the privacy features in existing development frameworks are relatively limited, suffering from four types of deficiencies:

- End-users have to trust the developers but have little ways to check the real behavior of smart home products. All three paradigms rely on the assumption that users have to trust that companies will collect and use their data properly.
- Smart home products often need to be individually managed, and each comes with its unique privacy control interface. Distributed privacy control imposes a management challenge for users.
- Verifying the app behavior is a non-trivial tasks for third-party auditors. Third party auditors play an important role in today's data privacy ecosystem. For example, Electronic

[1]These paradigms are non-exclusive. Each paradigm comes with some trade-offs. To address some trade-offs, many smart home solutions use a combination of multiple paradigms. For example, full-stack products are often compatible with some interoperability protocols.

Frontier Foundation found that Ring shares a plethora of customers' personally identifiable information to third-parties [3] and pushed Ring to offer a privacy control interface for end users [4]. However, analyzing the privacy behavior is a daunting task for all three paradigms.

- There is no mechanism for users to control the data flow beyond the functionalities offered by the developers. A contrasting example is the modern mobile permission systems (e.g., Android, iOS), which allow users to configure whether an app has access to the data.

3 SYSTEM DESIGN

Our key insight is that cloud services do not need raw sensor data for many IoT scenarios. Indeed, developers often need to implement repetitive functions on the cloud back-end to extract the desired data, costing unnecessary development effort, bandwidth, and computational resources. For example, to implement a "HelloVisitor" app on a video doorbell, which recognizes the visitors in front of the door and personalizes welcome messages, developers need to implement an algorithm to extract the image area containing a human body before running a person recognition (Fig. 1).

Peekaboo factors out these common data preprocessing tasks (e.g., audio activity detection, face detection, and frequency spectrum extraction) from the cloud service side onto the hub, supports them as a set of reusable, chainable operators and then runs them over the sensor data before they flow to cloud services (and out of the users' control) (Fig. 1). The sensors in Peekaboo first send collected data to a trusted hub, and the hub preprocesses the data before sending the data to developers' servers. Here, the hub can be any computing resource that is fully controlled by the end-users, e.g., a raspberry pi, an old computer in the home, or a shared EC2 instance in the cloud. This design offers several unique benefits:

- Peekaboo can lower the barrier for smart home app development, since developers can utilize these system-wide reusable operators, alleviating the need to re-implement functionality.
- Peekaboo reduces unnecessary data egress without preventing developers from implementing planned features. For example, "HelloVisitor" developers can request only the face images instead of the raw picture, which may capture other undesired information, e.g., the package on the floor, the home address, and a parked vehicle.
- The basic unit of each hub program is reusable operators with known semantics, making it possible to infer the real app behavior at the content level. When a user installs an app, Peekaboo can inform users of the required data content (e.g., face images). After installation, Peekaboo can show users what data content different apps are collecting (e.g., face images have been sent to the HelloVisitor app for X times).
- The hub becomes the gateway of all the outgoing network traffic, offering a unified and centralized privacy awareness and control for all the smart home apps.

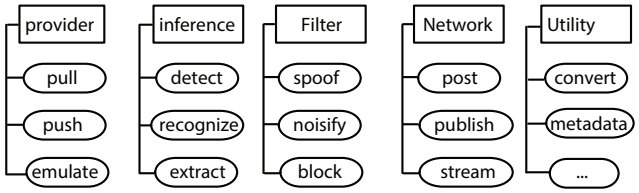

Figure 2: The taxonomy of action primitives. The operators are designed in a functional programming language fashion, where each operator is a data action (i.e., a "verb" statement). Here, we enumerate the common data actions across different operator categories. Note the actions are independent with the data types. For example, "face blurring" is the "noisify" action that applies to the face data.

The instantiation of Peekaboo software architecture will contain three main components: a programming interface allowing developers to build Peekaboo apps, a hub runtime that enables content-based late bind and system optimization, and program analyzers/rewriters to enable content-based privacy features.

Programming Interface

A Peekaboo application consists of a lightweight and elementary *hub program* running on the hub and a more computationally expensive and sophisticated optional *cloud program* running on developers' servers. With Peekaboo, a developer can implement a Peekaboo app in three steps: (1) connect the operators into flows; (2) configure the operator properties; and (3) implement the main program on the cloud.

The hub program is a connected graph of four types of *open-source* operators (Fig. 2): provider, inference, filter, and outgoing network, where each node corresponds to a verb statement (e.g., detect face) and the connections between operators are referring to the sensor data or derived data. Figure 1(c) illustrates a hub program for the "HelloVisitor" app. The provider operator (i.e., retrieve image) specifies the data retrieval behavior (e.g., pull v.s. push, frequency, resolutions) and gets raw sensor data from Peekaboo runtime. The inference operator (i.e., detect face) analyzes the raw data and derives content annotations. The filter operator (i.e., crop face) filters the sensor data based on the content annotation. The outgoing network (i.e., send2cloud) is the only operator that has outgoing network access, which sends the data to the remote cloud.

Runtime

Although Peekaboo is a software-intensive framework, it needs hardware to execute the hub programs, collect sensor data, and overcome the hardware constraints. We cannot ignore the hardware aspect of the total solution. This hardware dependency of Peekaboo imposes two major challenges.

First, how can developers build smart home apps without knowing the sensor placements? Setting up a smart home app for a designated sensor is relatively straightforward. However, many apps (e.g., home occupancy statistics) require collaborative input from sensors across the home [17]. Existing smart home solutions (e.g., Google Nest Aware) address this issue by constructing a closed ecosystem and ask users to configure the sensors through a location template (e.g., a camera at the living room), which corners users into vendor lock-in.

In contrast, Peekaboo runtime addresses this challenge by offering content-based sensor late binding. Developers only need to specify the requested data content type (e.g., occupancy data) as the input of the hub program. Peekaboo offers a set of built-in drivers (also implemented through reusable operators) to extract the occupancy data from applicable sensors, such as a microphone, camera, accelerometers. Peekaboo defers the sensor data binding until the moment users install the app on the hub. If a user has a camera installed in her living room and wants to install an app that consumes the occupancy data, the Peekaboo runtime will extract the occupancy data from the camera automatically. By doing so, developers can build smart home apps on non-proprietary hardware without knowing the sensor placements.

Second, how much computing resources do users need to run hub programs for a mid-size home? While Peekaboo only runs the lightweight tasks on the hub, a smart home may involve hundreds of sensors and thousands of apps. Peekaboo would not be feasible if the computational resource can not scale. In responding to this challenge, we plan to develop a hub program aggregation technique to optimize the computation resource utilization. Since all the hub programs are built with reusable operators, we can merge the hub programs and reuse the computed results across different apps on the same sensor. So the computation cost for a given sensor is capped.

Program Analysis for New Privacy Features

The design of reusable operators also makes Peekaboo apps easier to analyze and rewrite, enabling many new privacy features. First, as each hub program is defined by built-in operators that have known semantics, we can statically analyze the program to understand how the sensor data is processed step by step. Through the static program analyzer, Peekaboo can offer a privacy permission system at the content level. For example, a user may configure a privacy policy that "any face should not appear in any video streams from the living room." The Peekaboo app store will prevent her from installing any app requesting face data on certain camera sensors.

Second, the functional programming model eliminates external states, so we can analyze the real hub program behavior by inspecting each operator's input and output through the runtime, e.g., whether the outgoing image contains a face. The dynamic program analyzer can further offer content-based privacy awareness. For example, if a user wants to protect her face privacy, Peekaboo can run a deep fake operator to replace her face with an artificial face [6].

Third, the hub program's well-indexed nature allows the runtime to rewrite the hub program, enabling a privacy negotiation between the users and developers. For example, family members may not want their faces repetitively to be collected by the "HelloVisitor" developer. The runtime can offer "faking face" features, by inserting a "deep fake" spoofing filter into the hub program. Meanwhile, this spoofing process is also transparent to the developers. Developers may offer extra incentives to users for removing the filter if they want to collect new data for machine training. In doing so, Peekaboo aims to offer an infrastructure to enable an implicit privacy negotiation.

4 IMPLEMENTATION & PLAN

Our current implementation of Peekaboo contains the programming supports to build hub programs and a basic hub runtime to execute the hub programs. We implemented built-in reusable operators as Node-Red (https://nodered.org/) compatible Javascript packages so that developers can use the Node-Red web browser-based flow editor as the programming environment. As the next step, we will continue to build the runtime described above and enable new privacy features by incorporating program analysis and rewriting techniques.

5 EVALUATION PLAN

The primary metric of success for any toolkit is if it can be used to create a useful and non-trivial subset of the full design space of applications in a manner that is faster, is higher quality, or has more useful features than without it [8]. We will conduct detailed experiments to evaluate Peekaboo from three perspectives:

- **The expressiveness and extensibility of the programming API**. We have implemented six different privacy-sensitive smart home applications using Peekaboo, covering various scenarios (e.g., sensor types, events, data flow structure) to demonstrate the expressiveness. Meanwhile, we are also implementing a set of template operators to

help developers extend the operator with open source libraries and the latest machine learning models.
- **The system deployment practicality**. We deployed Peekaboo on a Raspberry Pi in the past month. Our preliminary experiment shows that the runtime aggregation techniques can improve the deployment efficiency significantly. For example, our experiment shows that a standard Raspberry Pi can support up to 10 camera streams simultaneously, each running 100+ applications.
- **Privacy benefits**. We demonstrated the privacy benefits through the implemented applications. We run a preliminary test on a Peekaboo smart TV app, which aims to protect users from undesired speaker recognition without breaking the speech recognition. Our results show that Peekaboo can reduce identity recognition (a group of 8 people) accuracy from 100% to 27% while the speech recognition accuracy remains the same. We plan to expand the privacy benefits experiments to more categories, such as OCR, face recognition, etc.

6 EXPECTED CONTRIBUTIONS

We expect our technical contributions would be as follows:
- We propose a new software architecture to support the building of privacy-trustable smart home applications. The new architecture factors out the common data preprocessing tasks from the cloud service side onto to the hub, supports them as a set of reusable, chainable operators and then runs them over the sensor data before they flow to cloud services.
- We introduce a new privacy management mechanism, content-based privacy management, which offers a fine-grained privacy awareness and control that operates on the content of each sensor data request.
- We introduce a new data-centric smart home development toolkit, Peekaboo, along with multiple different demonstration apps that have been built with it. We demonstrate that Peekaboo can offer a great deal of flexibility and reduce the amount of potentially sensitive data going to cloud services.

7 BIOGRAPHICAL SKETCH

Haojian Jin is a forth-year Ph.D. student in the Human-Computer Interaction Institute at Carnegie Mellon University, advised by Dr. Jason Hong and Dr. Swarun Kumar. His research is focused on building new IoT applications while protecting people's privacy. Haojian is expected to graduate at the May of 2022.

REFERENCES

[1] George Demiris, Brian K Hensel, Marjorie Skubic, and Marilyn Rantz. 2008. Senior residents' perceived need of and preferences for" smart home" sensor technologies. *International journal of technology assessment in health care* 24, 1 (2008), 120.

[2] Colin Dixon, Ratul Mahajan, Sharad Agarwal, AJ Brush, Bongshin Lee, Stefan Saroiu, and Paramvir Bahl. 2012. An operating system for the home. In *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*. 337–352.

[3] Electronic Frontier Foundation. 2020. Ring Doorbell App Packed with Third-Party Trackers | Electronic Frontier Foundation. https://www.eff.org/deeplinks/2020/01/ring-doorbell-app-packed-third-party-trackers. (Accessed on 07/06/2020).

[4] Stephen Gandel. 2020. Ring to change privacy settings after study showed it shared personal information with Facebook and Google - CBS News. https://www.cbsnews.com/news/ring-facebook-google-personal-information-privacy-settings-change/. (Accessed on 07/04/2020).

[5] Google. 2020. Nest Aware - Video Recording Subscription for Nest Cams - Google Store. https://store.google.com/us/product/nest_aware. (Accessed on 07/06/2020).

[6] David Güera and Edward J Delp. 2018. Deepfake video detection using recurrent neural networks. In *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, 1–6.

[7] Kashmir Hill. 2020. Activate This 'Bracelet of Silence,' and Alexa Can't Eavesdrop - The New York Times. https://www.nytimes.com/2020/02/14/technology/alexa-jamming-bracelet-privacy-armor.html. (Accessed on 04/16/2020).

[8] Jason I Hong and James A Landay. 2004. An architecture for privacy-sensitive ubiquitous computing. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*. 177–189.

[9] Giovanni Iachello and Jason Hong. 2007. *End-user privacy in human-computer interaction*. Vol. 1. Foundations and Trends in Human–Computer Interaction.

[10] IFTTT. 2018. Privacy policy - IFTTT. https://ifttt.com/terms. (Accessed on 07/04/2020).

[11] Anick Jesdanun. 2018. How to plan your smart home — and weigh privacy risks | The Seattle Times. https://www.seattletimes.com/business/how-to-plan-your-smart-home-and-weigh-privacy-risks/. (Accessed on 05/03/2020).

[12] Haojian Jin, Minyi Liu, Kevan Dodhia, Yuanchun Li, Gaurav Srivastava, Matthew Fredrikson, Yuvraj Agarwal, and Jason I Hong. 2018. 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 (2018), 1–27.

[13] Haojian Jin, Jingxian Wang, Swarun Kumar, and Jason Hong. 2019. Software-Defined Cooking using a Microwave Oven. In *The 25th Annual International Conference on Mobile Computing and Networking*. 1–16.

[14] Haojian Jin, Jingxian Wang, Zhijian Yang, Swarun Kumar, and Jason Hong. 2018. Wish: Towards a wireless shape-aware world using passive rfids. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. 428–441.

[15] Marc Langheinrich. 2002. A Privacy Awareness System for Ubiquitous Computing Environments. In *Proceedings of the 4th International Conference on Ubiquitous Computing* (Göteborg, Sweden) *(UbiComp '02)*. Springer-Verlag, Berlin, Heidelberg, 237–245.

[16] Alison DeNisco Rayome. 2019. Why consumers still don't trust IoT devices - TechRepublic. https://www.techrepublic.com/article/why-consumers-still-dont-trust-iot-devices/. (Accessed on 05/03/2020).

[17] William Noah Schilit. 1995. *A system architecture for context-aware molbil computing*. Ph.D. Dissertation. Columbia University New York, NY.

[18] Daphne I. Townsend, Frank Knoefel, and Rafik A. Goubran. 2011. Privacy versus autonomy: A tradeoff model for smart home monitoring technologies. *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society* (2011), 4749–4752.