Adaptive User Authentication on Mobile Devices

by

Jiayi Chen

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2022

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Dr. David Mohaisen Professor, Department of Computer Science, University of Central Florida
Supervisor:	Dr. Urs Hengartner Associate Professor, School of Computer Science, University of Waterloo
Internal Member(s):	Dr. N. Asokan Professor, School of Computer Science, University of Waterloo
	Dr. Yousra Aafer Assistant Professor, School of Computer Science, University of Waterloo
Internal-External Member:	Dr. Leah Zhang-Kennedy Assistant Professor, School of Interaction Design and Business University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

Chapters 3, 4, 5, and 6 are based on co-authored materials. I hereby declare that I am the first author and main contributor of these materials. The content in Chapter 3 is based on the published material [30] co-authored with Prof. Hassan Khan at the University of Guelph and Prof. Mohammad Mannan at Concordia University, who provided comments and did editing work. The contents in Chapters 4, 5, and 6 were co-authored with Prof. Hassan Khan, who provided the implementation of the Itus framework, offered comments, and did editing work. The rest of this thesis contains original content and was authored under the supervision of Prof. Urs Hengartner.

Abstract

Modern mobile devices allow users to access various applications and services anywhere. However, high mobility also exposes mobile devices to device loss, unauthorized access, and many other risks. Existing studies have proposed a variety of explicit authentication (EA) and implicit authentication (IA) mechanisms to secure sensitive personal and corporate data on mobile devices. Considering the limitations of these mechanisms under different circumstances, we expect that future authentication systems will be able to dynamically determine when and how to authenticate users based on the current context, which is called *adaptive authentication*. This thesis investigates adaptive authentication from the perspectives of context sensing techniques, authentication and access control adaptations, and adaptation modeling.

First, we investigate the smartphone loss scenario. Context sensing is critical for triggering immediate device locking with re-authentication and an alert to the owner before they leave without the phone. We propose Chaperone, an active acoustic sensing based solution to detect a user's departure from the device. It is designed to robustly provide a user's proximity and motion contexts in real-world scenarios characterized by bursting high-frequency noise, bustling crowds, and diverse environmental layouts. Extensive evaluations at a variety of real-world locations have shown that Chaperone has high accuracy and low detection latency under various conditions.

Second, we investigate temporary device sharing as a special scenario of adaptive authentication. We propose device sharing awareness (DSA), a new sharing-protection approach for temporarily shared mobile devices. DSA exploits natural handover gestures and behavioral biometrics as contextual factors to transparently enable and disable a device's sharing mode without requiring explicit input of the device owner. It also supports various access control strategies to fulfill sharing requirements imposed by an app. Our user study has shown the effectiveness of handover detection and demonstrated how DSA automatically processes sharing events to provide a secure sharing environment.

Third, we investigate the adaptation of an IA system to shared mobile devices to reject imposters and distinguish between legitimate users in real-time. We propose a multi-user IA solution that incorporates multiple modalities and supports adding new users and automatically labeling new incoming data for model updating. Our solution adopts a score fusion strategy based on Dempster-Shafer (D-S) theory to improve accuracy with considering uncertainties among different IA mechanisms. We also provide an evaluation framework to support IA researchers in the evaluation of multi-user, multi-modal IA systems. We present two sample use cases to showcase how our framework helps address practical design questions of multi-user IA systems.

Fourth, we investigate a high-level organization of different adaptation policies in an adaptive authentication system. We design and build a multi-stage risk-aware adaptive authentication and access control framework (MRAAC). MRAAC organizes adaptation policies in multiple stages to handle various scenarios and progressively adapts authentication mechanisms based on context, resource sensitivity, and user authenticity. We present three use cases to show how MRAAC enables various stakeholders (device manufacturers, enterprise and secure app developers) to provide adaptive authentication workflows on COTS Android with low processing and battery overhead.

In conclusion, this thesis fills the gaps in adaptive authentication systems for shared mobile devices and adaptation models for authentication and access control. Our frameworks and implementations also benefit researchers and developers to develop and evaluate their adaptive authentication systems efficiently.

Acknowledgements

I would like to thank my supervisor, Prof. Urs Hengartner, for the dedicated and constant support, guidance, and encouragement to make the completion of this thesis possible. I am very grateful that he spent a sheer amount of time reviewing my research work and discussing with me. His advice helped me quickly locate the problems in my research and work out possible solutions. He was always ready to assist me whenever I encountered any difficulty during my PhD. Despite the inconvenience brought by the COVID-19 pandemic, I was still able to receive prompt guidance and feedback from Prof. Hengartner, which has kept my research going smoothly.

I would like to thank Prof. Hassan Khan at the University of Guelph, who has also provided tremendous support and effort in this thesis. It is my great pleasure to collaborate with Prof. Khan on so many research topics during my PhD. His invaluable assistance and insights have greatly improved the quality of my research.

I am also grateful to Prof. Mohammad Mannan at Concordia University, who co-authored the first published paper of my PhD research. His insightful and critical comments give me much inspiration.

I am fortunate to have an outstanding and well-rounded committee: Prof. David Mohaisen, Prof. N. Asokan, Prof. Yousra Aafer, Prof. Leah Zhang-Kenndy. I am grateful to them for agreeing to serve on my committee.

I would also like to thank CrySP members for their great support on my research, particularly Steven Engler, Pierfrancesco Cervellini, and Matthew Rafuse, who provided help in improving my study protocols.

I am also thankful to my friends, Jiaqi Wang, Shufan Zhang, Andrew Wang, Zhao Zhang, Zebin Kang, Yamin Jahangir, Wen Cui, Matthew Kwok, for their help and support.

For Chapter 3, we gratefully acknowledge the support of NSERC for grant RGPIN-2014-0549. For Chapters 4, 5, and 6, we gratefully acknowledge the support of the Waterloo-Huawei Joint Innovation Laboratory for funding our research.

Dedication

Dedicated to my father, *Yuesheng Chen*, my mother, *Haijuan Guo*.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 User Authentication on Mobile Devices

Modern mobile devices, such as smartphones and tablets, have brought great convenience, strong connectivity, and high mobility to people's work and life. While people tend to use mobile devices for a variety of tasks, these devices also store sensitive personal or business data or provide access to sensitive operations. Despite benefiting from high mobility, mobile devices are also at high risks of unauthorized access and device loss. *User authentication* is usually required to unlock a mobile device when a user makes access attempts. Although password authentication is the most classic and widely adopted user authentication mechanism, it is poor in usability [42, 67] and suffers from shoulder surfing attacks [14, 165]. Researchers have devoted to replace password authentication with other authentication mechanisms [8, 19]. Biometrics-based authentication mechanisms, such as fingerprint and face recognition, have become popular in modern mobile devices since they make significant improvements in usability by reducing users' time and effort in doing user authentication. However, all these authentication mechanisms require a user's attention or explicit inputs. Moreover, once a device is unlocked, most authentication mechanisms cannot provide further identity verification to ensure the validity of the user identity throughout a session.

1.1.1 Implicit Authentication

Implicit authentication (IA) transparently authenticates a user's identity to improve the security and usability of user authentication. Note that we refer to the aforementioned

user authentication methods as Explicit Authentication (EA) in contrast. IA leverages users' distinct device usage or behavioral patterns to distinguish a user from others in a non-intrusive way. On the one hand, IA provides an additional authentication factor to supplement EA mechanisms. Many IA mechanisms [17, 100, 104] can continuously verify a user's identity in the background during the device usage. For example, an attacker may launch a shoulder surfing attack to obtain the PIN code to unlock a device. Behavioral biometrics based IA mechanisms can still block the attacker from accessing the device by comparing the attacker's touch patterns [37] or keystroke dynamics [148] to the device owner's. On the other hand, IA helps reduce unnecessary EA requests for alleviating a user's burden on doing user authentication. As long as an IA mechanism can verify the current user's identity, it can maintain the authentication status of the user and keep the device unlocked. Once it detects a mismatch in device usage or behavioral patterns, it can activate a specific EA mechanism to authenticate the user.

However, IA mechanisms still have the following limitations:

- **Accuracy.** According to the evaluation conducted by existing studies [52, 92], the error rate of behavioral biometrics based IA mechanisms is not negligible. A false rejection may interrupt a user's normal device usage and ask for re-authentication, while a false acceptance may temporarily expose sensitive resources to an attacker.
- **Availability.** IA mechanisms rely on certain patterns to verify a user's identity, while these patterns may be related to specific activities. The absence of the required activities and patterns will result in the unavailability of an IA mechanism.
- **Detection latency.** IA mechanisms may need to collect sufficient device usage data to determine if the current user is legitimate or not. Thus, it may take time for IA mechanisms to detect an attacker.
- **Power consumption.** To provide continuous authentication, an IA mechanism may need to continuously collect and process sensor data. The additional power consumption of IA computation and sensor activities is not negligible for mobile devices.

In spite of continuous effort in looking for new behavioral biometrics to address these limitations, a possible avenue is to combine different IA mechanisms, which is multi-modal IA. Compared to single-modal IA, multi-modal IA provides higher accuracy, broader coverage of continuous authentication, and defense against attacks targeting single modalities. However, multi-modal IA also leads to more overhead given that multiple IA mechanisms

collect and process sensor data continuously. In trade-off, the authentication system is supposed to control each IA mechanism to reduce unnecessary computation while maximizing the accuracy of combining the available modalities. We review the multi-modal IA techniques in § 2.4.2.

1.1.2 Adaptive Authentication

The emergence of IA shows a trend for adapting user authentication on mobile devices to use context to balance security and usability. More generally, a mobile authentication system is expected to dynamically choose the appropriate authentication method(s) or adjust its behaviors according to its operating circumstances, which is called an *adaptive authentication system*.

Adaptive authentication consists of two key elements: *context* and *adaptation*. According to Arias-Cabarcos et al. [10], context changes make an authentication system fail to satisfy the *security* or *usability* requirements. Consequently, the authentication system needs to apply changes to its authentication mechanisms to fit into the current circumstance. Taking Android Smart Lock [82] as an example, a user's *departure from the secure location* (secure context) implies higher risk of unauthorized access, which results in the *activation of EA for the next access* (adaptation).

Contexts are characterized by contextual factors [187], which involve:

- **Device-related** contextual factors describe the device status, including network connectivity, signal strength, nearby/connected devices, etc.
- **User-related** contextual factors describe the current device user, including physical movements, activities, user's proximity, social relationship, etc.
- **Physical** contextual factors describe the physical environment in which the device is located, including locations, ambient light, ambient noise, temperature, etc.
- **Temporal** contextual factors describe the time information, including date, time, etc.

Context sensing techniques leverage various sensors of mobile devices to obtain contextual factors and recognize the current context. For the example of Android Smart Lock, a mobile device can determine the current location via location sensors (e.g., GPS) and compare it to the user-defined trusted location to determine if it is secure or not.

The adaptation of an authentication system should fulfill the security and usability requirements of the new context. Intuitively, a less secure context needs stricter authentication to defend against higher risk of unauthorized access. Moreover, the adaptation scheme should target low authentication overhead and reduce time and effort of a user on authentication. The adaptation process involves all aspects of an authentication system. Based on the taxonomy of Arias-Cabarcos et al. [10], we divide the adaptation process into the following levels:

- **Parametric adaptation** is mainly about tuning the parameters of an authentication mechanism. For example, an IA mechanism for continuous authentication can tune down its frequency for less battery consumption.
- **Algorithmic adaptation** is to change the algorithm or model of an authentication mechanism for a different one when it does not work well (e.g., low accuracy) for the current context. For example, a face recognition based authentication mechanism may need to apply additional data pre-processing [76] to mitigate the accuracy degradation under a poor illumination condition.
- **Structural adaptation** is about activation and de-activation of authentication mechanisms. An authentication system may include a variety of authentication mechanisms, and need to determine which mechanism(s) to adopt for the current context. For example, if a user is using the device while walking, the authentication system can choose to activate a gait-based authentication mechanism for IA.
- **Systematic adaptation** changes the behaviors of an authentication system, including how to make decisions and how to respond to certain authentication results and context sensing results. For example, the system can choose to restrict access to sensitive resources instead of locking out a non-owner user when the device is temporarily shared with a guest.

1.1.3 Two Illustrative Examples

Adaptive authentication is a very broad concept that involves various context sensing techniques, authentication mechanisms, and adaptation methods. We start with two example scenarios to illustrate how context sensing techniques make mobile device aware of risks and how an authentication system adapts itself to these risks.

Smartphone loss. Smartphone loss is a common scenario where a smartphone is left unattended and incurs high risks of unauthorized access from nearby opportunistic attackers. Following the concept of adaptive authentication, the authentication system of the

device should be able to detect potential device loss and automatically lock the device (i.e., de-authenticate the user and trigger a specific authentication mechanism) to defend against unauthorized access. A user's proximity to a device is an important context factor for device loss detection since a user is usually physically far away from the device at a device loss event. However, proximity is not sufficient to determine a device loss event. We still need to incorporate other contextual factors. For example, a user's motion trajectory may imply that the user is moving away from the device. Besides, since smartphone loss is not likely to occur at secure locations (e.g., home), adding location restrictions can avoid unnecessary loss detection. We further investigate context sensing techniques for loss detection and provide a loss prevention solution in Chapter 3.

Device sharing. In practice, device owners share their devices with other people in daily life, where non-owner users are allowed to access the device. (We use the terms "*owner*" to refer to a smartphone owner sharing their device and "*sharee*" to refer to people a device is shared with.) The device sharing scenario breaks the assumption of many IA mechanisms that the device owner is the only legitimate user. Besides, most mobile devices and apps follow a single-user design and adopt all-or-nothing access control [4, 71] — once the device is unlocked, a non-owner may have access to most sensitive resources during sharing. In this scenario, a mobile device is supposed to detect a sharing event and adjust the behaviors of its IA mechanisms to prevent them from blocking the non-owner user upon a mismatch in device usage or behavioral patterns. It is also necessary to impose access control to prevent unauthorized access to sensitive resources on the device. In Chapter 4, we study context sensing techniques for the detection of temporary sharing with the incorporation of IA mechanisms for owner detection. Chapter 5 further investigates the adaptation of IA mechanisms to shared devices for multiple legitimate users.

1.2 Thesis Statement and Objectives

The thesis statement is stated as follows:

Context sensing and implicit authentication enable authentication systems of mobile devices to identify various risks resulting from real-world device usage practices and to adapt to varying security and usability requirements imposed by context changes.

This thesis includes four research objectives, which can be classified into two categories: 1) Proposing new adaptive authentication solutions to address practical scenarios, and 2) Designing new systems and frameworks to facilitate researchers and developers to develop

adaptive authentication systems. Each research objective consists of a series of research problems.

For the first category, we mainly investigate how to make mobile devices aware of a specific scenario and then decide what to adjust in their authentication systems to handle this scenario. We first need to determine the contextual factors that characterize the risk and design context sensing techniques to capture these factors. Then, we need to identify the risk of the scenario and apply adaptations to the authentication systems based on the security and usability requirements.

- **Objective 1 (Smartphone loss prevention):** Identify contextual factors that indicate potential smartphone loss, and design context sensing techniques accordingly. Determine when to automatically lock the device for re-authentication and how to alert a user to prevent smartphone loss.

  Smartphone loss detection is related to the user's proximity context. As smartphone loss happens when the device is not with the owner, a context sensing technique for loss prevention is required to detect a user's movement from a distance. Besides, to make context sensing reliable and robust, we need to consider possible environmental factors that may interfere with the sensing process. Evaluating a prevention solution should be performed under real-world conditions as well. Besides, a prevention solution should produce few false positives and effectively alert a user of smartphone loss in addition to locking the device.

- **Objective 2 (Device sharing awareness):** Identify contextual factors that indicate the starting and end of temporary device sharing, and design context sensing techniques accordingly. Enable mobile device to proactively detect device sharing with little to no inputs of users. Adapt authentication and access control of mobile devices to device sharing.

  Temporary device sharing allows a non-owner user to temporarily access the device, which is a special case for the authentication system. The goal of device sharing awareness is to adapt the authentication system to a detected sharing event swiftly. From the perspective of context sensing, the device needs to detect the handover event from the owner to another person as the starting of device sharing, which involves hand movement detection via motion sensors. Furthermore, it is critical to determine the end of a sharing event to ensure that the device has been returned to the owner. During sharing, the authentication system should not block the user from accessing non-sensitive resources.

For the second category, we investigate the complicated adaptations of an adaptive authentication system. On the one hand, an adaptive authentication system can involve adaptations at multiple levels. For example, adapting an IA system to multi-user scenarios requires algorithmic and systematic adaptations. On the other hand, an adaptive authentication system can adopt multiple adaptation policies to handle different scenarios. Thus, we need to design a framework to organize various adaptations.

- **Objective 3 (Multi-user IA system):** Adapt IA mechanisms to shared mobile devices for multiple users. Combine multiple IA mechanisms for multi-user authentication systems. Enable multi-user IA model update with new users and data. Provide a workflow for evaluating IA mechanisms for the multi-user scenarios.

  Another device sharing scenario is where multiple users mutually use a single mobile device. As most existing IA systems are designed for single-user scenarios, we need to design multi-user IA systems to prevent unauthorized access of strangers or other legitimate users proactively. For algorithmic adaptation, we need to extend single-user IA mechanisms for multi-user scenarios and make the IA models adapt to new incoming users and data. Besides, we also need to fuse various IA mechanisms with adaptation to their uncertainties, which change with model updates. For systematic adaptation, the system should be able to support new user enrollment, label the incoming data automatically, and process possible exceptions. In addition, an evaluation framework is essential to help IA researchers construct trace-based evaluation tasks to compare different multi-user IA schemes.

- **Objective 4 (Multi-stage adaptive authentication):** Model adaptive authentication systems for complicated adaptations. Design a framework to support multiple adaptive authentication schemes for different scenarios. Develop adaptive authentication libraries for developers to build adaptive authentication systems on mobile devices.

  When enabling multiple adaptations in an authentication system, it is essential to ensure that they do not conflict with each other or result in potential loopholes that enable attackers to bypass the authentication system. Also, an adaptive authentication system should be extensible to enable developers to handle different scenarios. Thus, our goal is to first propose an adaptation model to organize various adaptations. Possible problems include what factors to consider for risk modeling, how to model the behaviors of authentication mechanisms and context sensing techniques, and how to organize adaptations on multiple levels. Then, we aim to propose a framework that helps developers build adaptive authentication systems with low development and computation overhead.

1.3 Main contributions

This thesis contributes new methods and systems to implicit and adaptive authentication domains. We start with two practical scenarios, smartphone loss and device sharing, respectively, by investigating the related context sensing techniques. For a follow-up to device sharing, we investigate how to adapt an IA system to shared mobile devices with multiple users, which involves changes in both algorithmic and systematic levels. Finally, we provide a general adaptive authentication framework that supports adaptive authentication and access control for different risk types. Also, we provide the implementation of all the frameworks and systems presented in this thesis to benefit developers and researchers.

The main contributions of this thesis include the following four aspects:

1. We provide a novel context sensing technique that uses active acoustic sensing to detect a user's departure and absence from a mobile device without the help of additional hardware and devices. Based on the context sensing technique, we propose a smartphone loss prevention solution, Chaperone. According to extensive evaluation with lab experiments, real-world experiments, and user study, Chaperone can proactively detect potential device loss under complicated environmental conditions, and automatically lock the device and effectively alert the user.

   Related Publication:

   [30] Jiayi Chen, Urs Hengartner, Hassan Khan, and Mohammad Mannan. Chaperone: Real-time locking and loss prevention for smartphones. In 29th USENIX Security Symposium. USENIX Association, 2020.

2. We combine handover detection and IA-based owner detection to make smartphones proactively and implicitly detect device sharing events. We then propose device sharing awareness (DSA) to help secure sensitive data during device sharing with considering human factors of forgetfulness and mistrust. An extensive evaluation shows that DSA can accurately detect and process sharing events with low battery consumption.

3. We investigate multi-user IA for shared mobile devices and propose an architecture for multi-user, multi-modal IA systems with considering new incoming users and data. We also provide a Dempster-Shafer theory based score fusion method to fuse the scores from different multi-user IA mechanisms for better accuracy. In addition, we design an evaluation framework, SHRIMPS, to help IA developers and researchers to build trace-based tasks with real-world datasets and evaluate multi-user IA schemes.

4. We propose multi-stage adaptation modeling for adaptive authentication, which supports the design of complex and stateful adaptation schemes. We design a multi-stage framework, MRAAC, for adaptive authentication and access control for secure app and system developers. The framework supports automatic generation of multi-stage adaptation models. A demonstration of three use cases shows that MRAAC can help build adaptive authentication systems with low overhead and battery consumption.

5. We release the source code of our solutions, frameworks, and tools, as well as data sets collected for evaluation at the following URL: https://github.com/cryspuwaterloo/jiayi_thesis_code. It helps other researchers reproduce our evaluation results and also benefits developers to design their own adaptive authentication systems.

1.4 Thesis Organization

The remainder of the thesis is organized as follows: Chapter 2 provides the background and a literature review of context sensing, implicit authentication, device sharing, and adaptive authentication and access control. Chapter 3 presents an active acoustic sensing based device loss prevent solution, which addresses *Objective 1*. Chapter 4 addresses *Objective 2* by proposing a device sharing awareness solution that uses both handover detection and behavioral biometrics based owner detection to capture device sharing events. Chapter 5 first extends and adapts multi-modal IA systems for multi-user scenarios and then provides a simulation framework for evaluating multi-user IA, which address *Objective 3*. To address *Objective 4*, Chapter 6 proposes multi-stage adaptation models and provides a general adaptive authentication and access control framework for mobile devices. Chapter 7 concludes the thesis with a summary and future avenues.

Chapter 2

Background and Literature Review

In this chapter, we first provide background and existing solutions for the two illustrative examples, smartphone loss and device sharing, respectively. Then, we review the existing studies of context sensing techniques, implicit authentication, and adaptive authentication, which are related to our approaches and systems. Finally, we review the evaluation methodology and metrics.

2.1 Smartphone Loss

2.1.1 Background

Smartphone loss is a serious security risk that has affected millions of users. In 2018, Kaspersky Lab [91] reported that on average, 23,000 Android devices are being lost or stolen each month. In 2016, half a million UK residents had a mobile phone stolen, and 35% of these phones were stolen while they were being left out and unattended [140]. A 2017 study reported that 6% of Australians lost their device in the previous five years [50]. Most stolen phones are never recovered—e.g., 68% US users failed to retrieve their phones in 2014 [102]. Users were more likely to lose their smartphones in public places, e.g., coffee shops and bars, where strangers can steal them [39]. In 2019, smartphones were the most commonly lost item in the ride-hailing service Uber [170]. Wiese et al. [178] observed that 49% of office workers put their phone unattended on a desk, which incurs unauthorized access of co-workers to sensitive data [164]. Beyond privacy threats, stolen or lost devices can also significantly affect enterprise security [25, 134, 166]. Smartphone loss is highly

related to a user's absence. Furthermore, if a device does not get locked in time, sensitive or confidential data on the device will be vulnerable to unauthorized access.

2.1.2 Existing Smartphone Loss Solutions

Many solutions have been designed to secure a lost smartphone or its data. We term these solutions as *post-loss* solutions. These solutions aim to prevent unauthorized access to the sensitive data stored on lost devices. This goal is mostly achieved by locking the phone's screen after a configurable idle period. However, an adversary, like a co-worker, may be able to pick up the phone before it locks. Ideally, the phone screen should be locked as soon as its owner steps away. *Proximity-based* solutions [59, 137, 181] target this goal by making the owner carry an additional device, and use RFID or Bluetooth to detect proximity to the phone. However, these solutions do not provide a very accurate measure of distance [105]. Yang et al. [184] proposed *Surround-See*, a smartphone equipped with an omni-directional camera for peripheral vision. One suggested application is warning users when they leave their phone behind. However, such special purpose cameras are unavailable on current smartphones. IA is another option since it can detect when a non-owner is using the phone, and subsequently locks the phone. Mirsky et al. [130] investigated the scenario where an attacker picks up an unattended phone and starts using it. They showed that within seven seconds, continuous authentication could detect the change in behavior and lock the phone. However, it may fail in certain cases, in particular, against mimicry attacks [94].

Some other solutions assist with the recovery of lost devices. "Find My iPhone" and "Find My Device" are device tracking services available from Apple and Google, respectively. Once the device owner realizes that they have lost the device, they can use these services to locate, recover, or disable their smartphone. Yu et al. [185] presented a post-loss solution that uses emergency call mechanisms to allow the device owner to wipe their device remotely after a loss. This solution works even if a thief removes the SIM card from the device. However, the solution is not designed to prevent the physical loss of the device. Usually there is some delay between the device loss event and the owner's realization of it. For devices lost in public places, this delay is sufficient for strangers to steal the device and turn on airplane mode to render such solutions ineffective. Therefore, a phone that is about to become unattended in a public place should try to prevent this loss by alerting its owner (e.g., playing an alarm sound), in addition to locking its screen.

To address the limitations of post-loss solutions, we aim to enable mobile device to proactively detect and *prevent* potential device loss. Our solution is to use active acoustic

sensing to track a user's motion and determine whether a device loss event is likely to happen. We review acoustic sensing and other wireless sensing techniques in § 2.3.1.

2.2 Device Sharing

This section first reviews device sharing surveys to understand the nature of device sharing and then reviews existing device sharing solutions.

2.2.1 Device Sharing Surveys

According to recent surveys, mobile device sharing is common in people's daily life [124] and even a systemic practice in some regions (e.g., South Asia [4, 7, 151]). These survey have investigated common device sharing practices and suggested how mobile operating systems or apps should be designed to secure the device owner's private and sensitive data. Hence, we provide an overview of the nature of device sharing following the four aspects: *why to share*, *what to share*, *whom to share with*, and *how to share*.

Why to share. Prior research has shown that it is common for smartphone users to temporarily share their devices for trust and convenience [90, 124]. Matthews et al. [124] conducted a survey of 99 households and a diary study with 25 participants. The results have shown that most recorded sharing events were intentional and could be classified into five sharing types (note: accidental sharing is excluded): borrowing, mutual use, setup, helping, and broadcasting. Borrowing and helping types are usually unplanned, and convenience is the primary motivation. The setup type is a sharee performing device setup or configuration activities on behalf of the device owner. Broadcasting occurs when a device owner and a sharee (or multiple sharees) view specific content on the device together. Different from the other four types, mutual use is a long-term sharing practice where two or more users have regular access to the device.

What to share. In the early years (e.g., the surveys conducted by Kalson et al. [90] in 2009 and Hayashi et al. [71] in 2012), a sharee tended to borrow a phone for communication (e.g., making phone calls, sending messages) and entertainment (e.g., playing games, watching videos). According to the survey conducted by Matthews et al. [124], people have been sharing various apps, including social networking apps, camera apps, browsers, etc. Notably, many shared apps contain both essential functions and sensitive information. For example, a camera app provides an essential photo-taking function, while it also connects to the albums of the owner's photos.

Whom to share with. As household sharing is a common social and cultural practice, mobile devices are usually shared with family members and friends who are socially close to a device owner. It is driven by the need for maintaining social relationships and signaling trust among people [4,7,123]. We elaborate on the trust implication in the next section and how it impacts the design of device sharing solutions. Besides, shared mobile devices are also common in workspaces, especially for front-line workers and medical workers [12, 40], where the sharing type is mutual use.

How to share. While providing the required resources by sharees, device owners should take measures to protect sensitive resources from unauthorized access. For household sharing, a device owner tends to supervise a sharee's device use during sharing [124]. Besides, they may adopt app locks to protect sensitive apps with individual passwords [71]. However, extensive qualitative studies [4, 90, 124] have shown that people have privacy concerns over sharing their mobile devices because of possible device misuse and exposure of sensitive or private data. For example, a social networking app may keep a user logged in due to its single-user design. A sharee can move to that app during sharing and access restricted data or functionality. Hang et al. [65] reported that the majority of participants in their user study wanted the ability to share some apps, but not others. Furthermore, participants desired to grant sharees access to only specific features of a shared app.

2.2.2 Trust

Trust is an important motivation for device sharing. According to Matthews et al. [124], device owners are willing to share their devices unsupervised with trusted sharees. A device owner's trust in a sharee is usually based on their relationship and experience. Despite trust, accidental exposure of sensitive resources is still possible during sharing. Moreover, device owners also share their devices with others to signal trust, i.e., making sharees feel trusted [123,124]. Liu et al. [113] reported that 86% of the participants in their user study always kept their phone in their sight when sharing, which puts an extra burden on the owner and may make the sharee feel mistrusted. A guest account for socially close sharees has been deemed inappropriate since it signals mistrust [90,123]. Explicitly hiding certain apps or data may also arouse suspicion and imply a lack of trust [4,7]. Thus, it is necessary to take trust into account when designing device sharing solutions.

Recent device sharing proposals have been exploring how to protect sensitive resources while not compromising trust among people. Seyed et al. [155] proposed a modular smartphone comprised of multiple access-controlled hardware components to address the trust and convenience issues of device sharing. PrivacyShield [141] provided a subtle just-in-

time privacy provisioning system, which enables the owner to quickly configure an access control rule by entering pre-defined touch gestures. Ahmed et al. [5] adopted two accounts for shared use and secret space, respectively, which can be accessed via the same interface but with different passwords. To address the trust issue, we focus on taking control of the entire sharing process *proactively* and *automatically* so that smartphone users do not need to specify or enable access control rules in front of a sharee.

Note that we emphasize the subtlety of *enabling* a device sharing solution. We do not try to hide from a sharee that the device is currently in a restricted environment, which is a design problem [4]. Achieving this goal requires tremendous efforts of app developers to redesign their apps [5].

2.2.3 Device Sharing Control Solutions

Many technical solutions have been proposed to protect sensitive information from unauthorized access on a shared device. We classify these solutions into four categories based on their scopes and methods:

1. *Guest accounts* create an independent environment for sharees without access to the personal data of a device owner. However, it prevents sharees from accessing non-sensitive resources only available on the owner's account (e.g., non-sensitive photos, a public post on the owner's social networking app).

2. *App locks* (e.g., Samsung S Secure [45], Norton App Lock [103]) make an app require credentials (e.g., a PIN) for launching the app. App locks provide all-or-nothing access control: a device owner can only choose from sharing the entire app or nothing. It introduces unnecessary authentication overhead and does not apply to many common apps with personal data (e.g., browser apps, streaming apps).

3. *App pinning* (e.g., Android Screen Pinning [78], iOS Guided Access [77]) restricts a sharee's access to the current foreground app only. While it is handy for single app sharing, it fully blocks access to other apps but imposes no restrictions on accessing in-app content of the foreground app.

4. *Vaults* (e.g., Xiaomi's App Vault [85], Huawei's Private Space [115]) allow owners to hide apps and files from sharees. A common practice is to provide two interfaces for shared access and private access, respectively. It provides finer-grained control over the shared resources compared to the other methods. However, vault solutions have been found to provide limited stealth functionality [5]: 1) Most vault apps on the

market still provide an entry point that reveals the existence of a hidden vault. 2) They may only apply to specific file types (e.g., photos, text, videos).

Existing studies [65, 90, 124, 151] have studied owners' security and privacy concerns with sharing different apps and called for access control mechanisms for device sharing. Studies [65, 71, 90] have shown that all-or-nothing access control cannot meet the need for device sharing from both security and convenience aspects. xShare [113] enabled the owner to specify the resources to share and offered a restricted mode by modifying the mobile operating system. DiffUser [135] established a multi-user security model for Android by creating different accounts to apply different access control rules. SnapApp [24] adopted a time-constrained access control model where a short sliding gesture can activate a 30-second usage session. This scheme reduces the authentication overhead and enables quick device sharing, but the attacker can still launch an attack within the session. TreasurePhone [153] was proposed to combine both environmental and user contexts to realize context-dependent access control to groups of apps.

Overall, most existing systems need manual activation and lack interaction with third-party apps to secure sensitive resources. In Chapter 4, we propose *device sharing awareness (DSA)* as a novel solution to determine when to change the behavior of the authentication system (i.e., stop blocking a non-owner user) and impose access control (i.e., prevent a non-owner user from accessing sensitive data during sharing). DSA enables smartphones to proactively detect device sharing without being manually activated by an owner and provides flexible access control. It provides a new direction to ensure the subtlety of sharing control solutions for the consideration of trust, and prevents a user's failure in activating these solutions.

2.3 Context Sensing Techniques

This section reviews the literature regarding the context sensing techniques for human activity detection and gesture detection.

2.3.1 Wireless Sensing

Wireless sensing techniques leverage wireless signals to sense objects or human activities in the surroundings. Based on the availability consideration, acoustic signals and WiFi signals are possible candidates for mobile devices since most are equipped with speakers,

microphones, and WiFi modules. Although we mainly use active acoustic sensing for motion detection in this thesis, we also review WiFi sensing and passive acoustic sensing related studies and explain why we choose active acoustic sensing over them.

WiFi sensing. A common approach of using WiFi signals for activity detection is to analyze changes in the channel state information (CSI) [3,57,72,107,117,136,175,177,182,195]. Extensive studies have investigated using distinct patterns of CSI amplitude variations in the time domain to detect a user's presence [3, 136], hand gestures [72, 107], and motion [57, 182]. A typical WiFi sensing technique requires a transmitter (which is usually a wireless access point) and a receiver (which is a mobile device such as a laptop or a smartphone). When the transmitter sends a signal to the receiver, the signal can be reflected by floor, ceiling, walls, people, etc., and form multiple transmission paths (i.e., multipath effect [144]). If a person is walking in between the transmitter and the receiver, propagation paths are affected by the movements of that person's body, and consequently, the CSI values on the receiver change. The goal is to find out the correlations between human movements and CSI changes. For example, Xu et al. [182] adopted time-frequency analysis techniques to segment and recognize the walking movement, and applied wavelet decomposition to extract the speed information of different body parts. One advantage of WiFi sensing is its non-obtrusiveness, where WiFi signals can be easily transmitted though objects and obstacles. However, it also make it challenging to identify and track a specific user (e.g., the device owner) given that there might be several people moving within the range. Besides, it is difficult to extract CSI on commodity smartphones [195], which may require the root privilege or be not supported [60]. WiFi sensing techniques also require separate sender and receiver devices, and impose placement requirements, which makes them infeasible for context sensing in public places.

Acoustic sensing. Acoustic sensing techniques can be broadly classified into two categories based on their working principles: *passive acoustic sensing* and *active acoustic sensing*.

In passive acoustic sensing, the device continuously records the ambient sound and then detects ongoing events [32,146]. For example, Chu et al. [32] proposed the matching pursuit (MP) algorithm to extract features from the environmental sounds to recognize the audio context (e.g., to characterize a location). However, for human activity recognition, passive acoustic sensing can only sense activities characterized by certain sounds (e.g., human breathing sound while sleeping [146]), and cannot be used to detect user movements.

Active acoustic sensing techniques require mobile devices to send and receive acoustic signals. It can operate in a transmitter-receiver mode for multi-device applications similar to WiFi sensing: A transmitter plays acoustic signals via its speaker, and a receiver records

the sounds via its microphone and extracts acoustic signals. Due to the Doppler effect, the frequency of the received acoustic signal changes with the relative motion between a transmitter and a receiver. The receiver can estimate the speed of the transmitter accordingly. Zhang et al. [188] tracked the Doppler shift to detect the encountering of two people and then identify the user based on the acoustic signal. CAT [120] took advantage of both the Doppler effect and the frequency modulated continuous waveform (FMCW) signals to accurately estimate distance and speed for hand movement. However, all these techniques require another device to send acoustic signals, which is not always available in practice.

Active acoustic sensing can also work in a standalone mode where a single mobile device plays an acoustic signal and senses its echoes. Extensive studies [108, 109, 163, 169, 176, 186, 191, 192, 194] have investigated using single-device active acoustic sensing for a variety of acoustic sensing tasks on commodity off-the-shelf smartphones. Based on their applications, we classify these studies into the following five categories:

1. **Static object sensing.** Since a static object can reflect acoustic signals, it is feasible to sense the device-object distance: we can record the time of flight (ToF) of an acoustic signal, which makes a round trip between the device and the object, and calculate the distance For example, Zhou et al. [191] used a two-pulse signal to measure the distance from a smartphone to nearby objects and then identify the geometry of corridors and rooms. Tung et al. [169] proposed BumpAlert to detect nearby objects and alert a distracted pedestrian of a possible collision.

2. **User presence and activity detection.** Human movements can lead to dynamics in the reflected signals, making it possible to detect human activities using active acoustic sensing. Li et al. [108] used active acoustic sensing to detect a user's presence within a pre-defined distance and designed iLock, an automated device locking solution based on the user presence information. Lian et al. [109] proposed EchoSpot to locate a user in an indoor environment by extracting the echoes reflected from walls and the user's body. However, this solution require multiple devices to obtain a precise 2D location of the user.

3. **Gesture recognition and hand movement detection.** Active acoustic sensing is more sensitive to the human movements close to the device since there is less path loss [26]. Thus, researchers have also used active acoustic sensing to detect in-air hand movements and recognize gestures. Wang et al. [176] proposed LLAP that leverages the distance measurements between the device and a user's finger to obtain fine-grained finger movements. Sun et al. [163] considered both air-borne and

structure-borne (i.e., sound transmission through solid materials) signal transmission to track the finger movements on the back of smartphones. In summary, due to path loss and multipath effect, precise gesture detection is only possible when the user is very close to the device.

4. **Fingerprinting.** As an acoustic signal is reflected by multiple objects, the received signal is a combinations of various echoes from nearby objects. Thus, the difference between the original signal and the received signal is unique. Based on this idea, Tung et al. [168] investigated how to generate acoustic signatures to tag the indoor locations so that a mobile device can remember its current location with 1cm resolution. Zhou et al. [192] proposed EchoPrint that applied acoustic fingerprinting for user authentication. EchoPrint extracted unique features from the echoes reflected by a user's face, which are dependent on the user's 3D facial geometries.

In Chapter 3, we propose Chaperone that uses active acoustic sensing to detect potential smartphone loss proactively. Specifically, our solution aims to capture a user's departure from the device as a signal of potential smartphone loss. In terms of methodology, more close to Chaperone is iLock by Li et al. [108]. The basic idea of iLock is to automatically lock a device based on the user-device distance estimated by the FMCW-based sensing technique [2]. However, iLock was only evaluated in two relatively ideal environments: a lab and a library. Our experiments show that it fails to work reliably in some common scenarios due to environmental factors (see Figure 3.1, and details in §3.1.2). High-frequency noise, movement of nearby people, and the presence of obstacles may interfere with iLock's distance estimation and result in false positives. In comparison, the design of Chaperone takes these environment factors into consideration to make our solution robust under real-world scenarios.

2.3.2 Motion Sensor Based Sensing

Many mobile devices are equipped with motion sensors to track device movements. Common sensors include:

- An **accelerometer** sensor provides the acceleration measurements in relation to the three coordinate axes.
- A **gyroscope** sensor provides the rotation rate measurements for the three coordinate axes.

- A **magnetometer** sensor (or a **geomagnetic field** sensor) provides the magnetic field strength measurements for the three coordinate axes. Usually, it determines the orientation of a device with the help of accelerometer.

Mobile operating systems also provide software-based motion sensors, which aggregate raw sensor data from hardware-based sensors. For example, linear acceleration on Android [83] is derived from the measurements of an accelerometer, which excludes the gravity component using a gyroscope or magnetometer sensor.

Mobile device movements, measured by motion sensors, can reflect a user's input while the user is using the device in hand. It enables a user to control a device via motion and gestures. For example, a user can rotate the device to simulate steering in a racing game app. Device movements can also imply the physical environment or the user's activity even when a user is not interacting with the device. For example, Android Activity Recognition [84] can detect if a user is walking, running, or driving according to measurements of multiple motion sensors. Thus, the applications of motion sensors can be classified into two categories based on their goals: 1) motion detection and 2) activity recognition.

Motion detection. It usually involves detecting a series of meaningful movements of a specific body part (e.g., hand, arm, leg, and head). A representative application is gesture detection and recognition [6, 111, 116, 156]. Different from acoustic sensing, motion sensor based gesture detection requires a user to hold (e.g., smartphones) or wear (e.g., smartwatches) the device so that motion sensors can capture the user's hand movements through the device movements. Liu et al. [111] designed uWave to detect personalized gestures based on acceleration only. Since the uWave model is highly user-dependent, it requires a training stage for each user for each gesture pattern, which brings a non-negligible burden on the user side. Akl et al. [6] provided an accelerometer-only gesture recognition scheme to recognize 18 user-independent hand gestures using dynamic time warping and affinity propagation. Kim et al. [96] proposed deepGesture to use a recurrent neural network (RNN) to learn user-independent gesture models from both accelerator and gyroscope data. Avery et al. [11] used accelerometer, gravity, and orientation sensors to determine a user's handedness based on a moving gesture when the user picks up the device. In Chapter 5, we try to detect a user's natural handover gesture for triggering a device sharing solution. Different from the above existing studies, this handover gesture involves two people and does not follow a pre-defined hand movement trajectory.

Other applications of motion detection techniques include detecting a specific action of a user. Here, we focus on the actions related to our two example scenarios in § 1.1.3:

1. Device theft detection. Liu et al. [112] proposed a machine learning based method to detect pickpocket and grab-and-run phone theft events. The basic idea is to extract

features that are related to the motion patterns of the two theft events. However, their solution was limited to these two theft events and did not address unattended phone scenarios.

2. Gait detection and extraction. For gait-based authentication, it is necessary to determine the existence of gait data and then recognize each step to segment the motion data. Zou et al. [196] proposed a convolutional neural network (CNN) and Long short-term memory (LSTM) based solution to identify gaits from a piece of accelerometer and gyroscope data.

Activity recognition. Human activity recognition (HAR) is a time-series classification problem that determines the high-level activity (e.g., walking, running, riding bikes) of a user based on motion sensors [174]. Here, we formulate a simplified binary classification problem for detecting if a certain activity happens or not (i.e., $A = \{a_0, a_1\}$, where a_0 means that the activity does not happen, and a_1 means the activity happens):

Given a sequence of time-series sensor data with n data points: $\mathbf{s} = \{\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{n-1}\}$, where $\mathbf{d}_i$ is the sensor measurements at time t_i, train a model $\{y\}_{i=0}^{n-1} = \mathcal{F}(\mathbf{s}), y_i \in A$, where $\{y_i\}_{i=0}^{n-1}$ is the predicted labels (i.e., activities), to minimize the gap between the prediction and the ground-truth label denoted by $\{y_i^*\}_{i=0}^{n-1}$.

A typical HAR solution slides a window over the time-series sensor data (i.e., *signals*) and extracts features from each window. According to existing studies [22,49,101,162,172], we list common features used by HAR and categorize them into two main categories: 1) Time-domain features include:

- *Statistical features* provide a basic statistical summary of the sensor data, which are widely adopted as the most fundamental features [22, 73, 101, 172]. They include mean, variance, standard deviation, maximum, minimum, range, n^{th} percentile, etc.
- *Correlation features* describe the relationships between two signals from different sensors or different axes [22, 49]. Common correlation based features include correlation coefficient (for measuring the strength of the relationship between two signals) and cross-correlation (for measuring the similarity between two signals):

$$\rho_{x,y} = \frac{cov(x,y)}{\sigma_x \sigma_y}, \tag{2.1}$$

$$CC_{x,y} = \max_{j=0,1,\ldots,n-2} \frac{1}{n} \sum_{i=0}^{n-1} x_i y_{i-j}, \tag{2.2}$$

where $x = \{x_0, x_1, \ldots, x_{n-1}\}$ and $y = \{y_0, y_1, \ldots, y_{n-1}\}$ are the two signals, $cov(x, y)$ is the covariance between x and y, σ is the standard deviation of a signal.

- *Integration features* measure the area under the signal curve, which are usually applied to accelerometer and gyroscope data [62, 106]. They provide rough estimates of velocity (i.e., single integration of acceleration), distance (i.e., double integration of acceleration), and rotation angle (i.e., single integration of rotation rate). Note that the significant accumulative errors from sensor measurements make it unreliable to measure these metrics accurately without compensations from other sensors [149]. Nevertheless, they are still useful for learning patterns related to velocity, distance, and rotation angle.

2) Frequency-domtain features require applying the Fourier transform to the time-series signals. The common features include:

- *Spectral coefficients* reflect the magnitude and phase at their corresponding frequencies. The related features are about the spectral coefficients at the key frequencies related to the target activity (e.g., 0.5 to 3 Hz for detecting walking and running [49]).
- *Energy* is calculated as the normalized squared sum of the spectral coefficients, which is used to determine if a user is walking, running, riding, or driving [119].
- *Entropy* is a measure of uncertainty of a distribution. Spectral entropy measures the spectral power distribution [15]. It is applied for distinguishing signals that have similar energy level. The entropy of the time-series data can obtain reflect the sudden changes during a window [172].

After extracting the feature vectors, we need to label the data and then train a model with a specified machine learning algorithm.

To obtain more informative contexts, existing studies usually involve other sensors in addition to motion sensors. Vaizman et al. [171, 172] proposed a multi-modal system that uses various sensors (e.g., camera, microphone, location sensors, etc.) on smartphones and smartwatches to recognize a person's behavioral context in natural environments. Cruciani et al. [36] used data from both audio and motion sensors to train a convolutional neural network for high accuracy across different HAR tasks.

As introduced in § 1.2, it is possible to track users' hand movements and recognize sharing gestures to detect the starting of a sharing event. However, unlike behavioral contexts, such as walking and running, a sharing gesture lasts only several seconds and is not a repetitive or periodical activity. Nevertheless, we follow the feature selection from existing work [101, 171] to train our gesture detection model.

Type	Features
Time	inter-stroke time, stroke duration
Trajectory	start x, start y, stop x, stop y, direct end-to-end distance, mean resultant length, length of trajectory, mid-stroke area, ratio of end-to-end distance and length of trajectory, largest deviation from end-to-end line, 20th percentile of deviation from end-to-end line, 50th percentile of deviation from end-to-end line, 80th percentile of deviation from end-to-end line
Movement	average direction, average velocity, 20th percentile of pairwise velocity, 50th percentile of pairwise velocity, 80th percentile of pairwise velocity, 20th percentile of acceleration, 50th percentile of acceleration, 80th percentile of acceleration, direction of end-to-end line, median velocity at last 3 points, median acceleration at first 5 points
Other	phone orientation, mid-stroke pressure

Table 2.1: Features for Touch-based IA [52].

2.4 Implicit Authentication

The thesis investigates IA for shared mobile devices and incorporates IA into the framework of adaptive authentication. In this section, we review the behavioral biometrics based IA techniques that are involved in this thesis, and then introduce the existing literature about multi-modal IA and multi-user IA.

2.4.1 Behavioral Biometrics

One of the thesis objectives is to extend existing IA mechanisms for shared mobile devices instead of proposing new behavioral biometrics. Hence, we choose to exemplify the usage of our proposed frameworks and systems with existing IA techniques. Researchers have investigated various behavioral biometrics for IA, including touch [17, 52, 190], gait [38, 196], keystroke [104], breath [27], etc. Considering the availability of common behavioral biometrics, we mainly use touch and gait for this thesis.

Touch based IA. For mobile devices with a touch screen, touch events can be captured while a user is interacting with the device. By analyzing the patterns of each touch event, a touch-based IA mechanism can distinguish a user from others. Given that touch events

are very common during a session, touch-based behavioral biometrics can be used to authenticate a user continuously.

In this thesis, we choose Touchalytics [52] for touch-based IA. A touch event (i.e., a *stroke*) consists of a sequence of touch points, and each touch point is described with a tuple of its timestamp, position (x, y), pressure, area, and orientation. After connecting these touch points, we can observe the trajectory of this touch event. Touchalytics extracts 30 features for each touch event, and we select 28 of them in Table 2.1, which are available across different devices. Then, we apply the Random Forest machine learning algorithm [138] with 100 estimators for model training.

Gait based IA. When a user is walking with a mobile device, motion sensors can collect the user's motion data to obtain gait patterns. The basic idea is to first capture a periodic repetition every two steps (i.e., a *cycle*) and then extract features for each cycle. Given that the time intervals between every two steps are not always the same, time interpolation is required to make each cycle have the same sample number [38]. Other signal processing techniques, such as weighted moving average [38], low-pass filtering [35], etc., are also used for data smoothing and noise filtering. Then, a gait based IA scheme extracts features for each cycle and compares them to the owner's to determine if the current user is the owner.

We choose a deep neural network based gait-based IA scheme proposed by Zou et al. [196] (we call it DeepGait in this thesis). DeepGait trains a gait extraction model to determine if a sequence of motion data contains gaits. Then, it segments the motion data and extracts all gait cycles. For training and authentication, DeepGait uses an gait identification model that consists of a CNN network and an LSTM network. The CNN network adopts convolution kernels to abstract the feature of the gait curve along the time series, while the LSTM network is to memorize information interaction along the time series.

Sliding window strategy. To reduce false positives, we also use a (m, n)-sliding-window strategy [75] to aggregate the IA results: If m out of n instances are accepted as the owner's, the IA mechanism will accept the current user as the owner. However, the sliding window strategy may take more time (i.e., at least m data points) for the system to make decisions. Thus, it is necessary to include both accuracy and detection latency in the evaluation.

2.4.2 Multi-modal IA

Multi-modal authentication combines multiple biometric modalities to reduce false acceptances and false rejections. Most existing work on multi-modal authentication, with

the exception of DriverAuth [63], has focused on single-user scenarios. Extensive studies [44, 56, 86, 128, 161] have investigated combining multiple physiological biometrics such as face, voice, or fingerprint. For example, Gofman et al. [56] combined both face recognition and voice recognition to authenticate a smartphone user to achieve good overall accuracy even with poor quality face images and voice samples. Similarly, combining multiple behavioral biometrics enables IA to identify a user's identity with high confidence and lowers the chance of spoofing attacks.

The key problem of multi-modal IA is how to fuse different behavioral biometrics to achieve better performance. Abuhamad et al. [1] classified the fusion methods into three levels:

- *Feature-level* [63, 104, 173]. The basic idea of feature-level fusion is to construct a large feature vector combines the features of each modality. Vhaduri et al. [173] designed a multi-modal solution for wearable devices with feature-level fusion of step counts, heart rate, calorie burn and metabolic equivalent of task. Shrestha et al. [158] proposed ZEMFA to extract gait features from multiple devices to perform zero-effort authentication. Feature-level fusion is challenging when the modalities to be fused are not compatible with each other, i.e., their feature vectors are not computed in a similar way. For example, features of voice samples are usually in the form of Mel-Frequency Cepstral Coefficients (MFCC) [132], while features for face recognition are extracted from image pixels [87]. Combining the features of these two biometrics requires normalization and feature selection [56].

- *Score-level* [23, 34, 74, 150]. Score-level fusion collects and aggregates the classification scores from different models. One advantage of score-level fusion is that it relies on results only without changing the model of each modality. In general, a modality with higher confidence should have more weight in making decisions. Crawford et al. [34] proposed a score-level weighted average fusion method that gives more weight to more recent detection scores. Buriro et al. [23] calculated the weight based on the classifier performance for their weighted average fusion method. Smith et al. [160] adopted the Dempster-Shafer theory based score fusion for single-user scenarios. Our work extends the application of the D-S theory to cover multi-user scenarios. COR-MORANT [74] was designed to provide risk-aware continuous authentication for single-user cross-device scenarios. It proposed two weighted score threshold fusion methods and a Kalman filter based score fusion method to fuse the authentication score from different devices.

- *Decision-level* [53, 110]. Decision-level fusion collects decisions made by individual modalities separately and makes the final decision with adopting certain rules or

voting. Fridman et al. [53] used two low-level modalities (i.e., keystrokes and mouse movements) and a high-level modality of stylometry and applied decision-level fusion that minimizes the global Bayes' risk. Lin et al. [110] fused the decisions from six classifiers using a compound-voting mechanism to improve the accuracy.

2.4.3 Multi-user IA

Although most of the existing IA studies regarded the IA problem as a binary or one-class classification problem [63], a few studies conducted preliminary explorations of multi-user scenarios recently. Specifically, these existing studies have focused on addressing multi-user IA as a multi-class classification problem. Ehatisham et al. [43] leveraged physical activity patterns to identify the device owner and secondary users who have partial access to the device. However, the proposed IA scheme does not consider a general attacker class. DeepGait proposed by Zou et al. [196] slightly changed the deep neural networks of its gait authentication model to perform multi-user identification (i.e., identify the current user from a number of candidates). However, in practice, the system is expected to detect unauthorized access and track user switches simultaneously. ContAuth [28] adopted iCaRL [145] and EWC [97] to address the incremental learning problem for DNN-based single-modal IA mechanisms to improve cross-session performance for multi-user scenarios. It considers the attacker class for model training and adopts incremental learning techniques for new user enrollment. Gupta et al. [63] proposed DriverAuth to provider multi-user and multi-modal authentication for ride-sharing platforms. However, DriverAuth only authenticates a user at the beginning of a ride.

Chapter 5 investigates the challenges of designing a multi-user IA system, which is not limited to a multi-class classification problem. A multi-user IA system should be also able to 1) incorporate multiple modalities to ensure good accuracy for continuous authentication, 2) process and label new incoming data for existing users, and 3) support the enrollment of new users. Existing studies have not covered all these aspects.

2.5 Adaptive Authentication

According to Arias-Cabarcos et al. [10], an adaptive authentication system should be able to dynamically adjust its behavior in response to the operating environment. For example, Primo et al. [139] proposed a gait-based authentication mechanism that adapts to the position in which the phone is held to optimize the accuracy in the real-world scenarios.

		Smart Lock [82]	TreasurePhone [153]	CASA [70]	Prog. auth. [147]	PRISM [142]	ConXsense [129]	CORMORANT [74]	MRAAC (Chapter 6)
Condition	Env. factors	✓	✓	✓	✓	✓	✓	✓	✓
	User activities	✓			✓	✓			✓
	Biometrics				✓			✓	✓
Outcome	Structural	✓		✓	✓	✓	✓	✓	✓
	Parametric							✓	✓
Access control	App-level		✓		✓	✓	✓		✓
	In-app								✓
Structure	Single-stage	✓	✓	✓	✓	✓	✓	✓	✓
	Multi-stage								✓

Table 2.2: Comparison to existing adaptive authentication frameworks. "✓" denotes a supported feature.

Similarly, Crawford et al. [33] investigated the impact of movements on the accuracy of the keystroke dynamics based IA mechanisms and design. However, these studies only focus on the impact of context factors on a specific authentication mechanism.

This section mainly reviews the existing studies that provide a high-level architecture for adaptive authentication, which allows mobile app developers and device users to customize adaptation schemes. Table 2.2 lists the studies that satisfy the criteria and compares these studies with our adaptive authentication and access control framework, MRAAC (see Chapter 6). TreasurePhone [153] proposed "spheres" to manage access control rules based on location and user activity. It determines the visibility or availability of a certain file or app based on the current context, whose idea is close to the in-app control of MRAAC. Hayashi et al. [70] proposed CASA, a probabilistic framework for dynamically determining whether to explicitly authenticate a user and how to select an explicit authentication mechanism based on the user's location data. ConXsense [129] combined both locations and nearby Bluetooth devices to calculate the familiarity of the current context and decide whether to lock the device. However, it only considers three location tags (i.e., public, work, and private) and two safety levels (i.e., safe and unsafe), which lacks extensibility.

PRISM [142] proposed a context modeling approach that used HARD-BN [143] to automatically extract patterns from location, user activities, etc. It can automatically generate policies to determine if an EA mechanism is required to unlock a device. Besides, it also allows users to manually make policies. Progressive authentication [147] employed behavioral biometrics as a contextual factor to estimate the user authenticity level. However, it did not involve the adaptation of behavioral biometrics based IA mechanisms. Wojtowicz and Joachimiak [179] built a generic model for the contextual factors that affect the performance of biometrics-based authentication mechanisms. CORMORANT [74] is a multi-modal authentication framework that incorporates contextual factors into the fused score of several biometrics-based EA and IA mechanisms. It supports the adaptation of IA mechanisms — when fusing the different modalities, it dynamically adjusts their weights based on other contextual factors.

All above methods follow a single-stage design of adaptive authentication, which does not support complicated adaptation logic. In Chapter 6, we introduce a novel multi-stage adaptation framework to organize both IA and EA mechanisms as well as fine-grained access control.

2.6 Common Threat Model

User authentication primarily targets unauthorized access to sensitive resources on the devices. Sensitive resources include apps (e.g., banking apps, messaging apps), files (e.g., photos, videos), and related critical operations (e.g., modifying or deleting files, making transactions). We assume that there is only one primary user (i.e., owner) who has full access to a mobile device. Unauthorized access can be initiated by any non-owner user, including strangers, sharees, and secondary users (for multi-user scenarios, see Chapter 5). These attackers are physically close to the device so that they can directly operate the device. Most unauthorized access happens when the device owner does not pay attention to the device or the device is out of sight of the device owner. However, a sharee's accidental access to sensitive resources on a shared device is also considered unauthorized access even if a device owner supervises the device sharing. For example, when a device owner temporarily shares the device with a sharee for entertainment, a pop-up message notification may expose sensitive information.

We describe attackers based on their relationship with the owner and their knowledge about the authentication system:

Opportunistic attackers are usually nearby strangers in the same premises as the owner.

They have little knowledge about the security measures adopted by the device, and therefore, use the device following their usual habits. A common example scenario is that a stranger picks up an unattended mobile device and tries to access random resources on the device.

Informed attackers are mainly social insiders who socially close to the device owner (e.g., family members, friends, co-workers) and may have knowledge of the owner and the security measures adopted by the device. Compared to strangers, social insiders have more opportunities to access an unlocked device and launch targeted attacks. For example, Marques et al. [123] referred to the shower time attack where an individual tries to access their intimate partner's smartphone when the partner is temporarily absent. For temporary device sharing, a device owner usually needs to unlock their device before giving it to a sharee. Based on an attacker's knowledge, they may attempt to access sensitive resources directly and launch attacks before the detection. A knowledgeable attacker may try to bypass the authentication system or our proposed methods.

We make the following assumptions for the common threat model. We trust the device owner and focus on the attacks launched by non-owner users. we assume the operating systems running on mobile devices are trusted, and the device owner installs app from trusted sources. Therefore, we assume an attacker cannot install malicious apps, modify the operating system (e.g., tampering sensor measurements), or exploit the root privilege. Besides, we assume the EA mechanisms are not compromised by an attacker (e.g., an attacker does not know the master password of the owner). As noted by several surveys [7, 123, 124], a non-owner user may know the passcode for the lockscreen of a device through shoulder surfing or other means. We assume the EA mechanisms triggered by an IA rejection do not use the same credential as the one for the lockscreen.

2.7 Public Datasets

The thesis involves the evaluation of context sensing techniques and authentication systems via lab experiments, real-world experiments, trace-based evaluation, and user studies. For lab and real-world experiments and user studies, we collected data while participants were asked to complete a series of tasks. For trace-based evaluation, we ran our approaches or systems over our collected data and public datasets and analyzed the results. The public datasets used in this thesis include:

HMOG [159] collected sensor data from 100 Android smartphone users while they were reading, writing, or doing map navigation session with the phone in their hands. The

HMOG data provides multi-user, multi-modal, and multi-session data, which meets most data requirements of this thesis. Thus, we use the HMOG [159] dataset as the main public data source for Chapters 4, 5, and 6. We list the dataset details as follows:

- **Activities:** document reading (i.e., reading), text production (i.e., writing),navigation on a map to locate a destination (i.e., map navigation)
- **Motion conditions:** walking, sitting
- **# of tasks per user:** 24 (8 reading tasks, 8 writing tasks, and 8 map navigation tasks)
- **Range of task lengths:** 5 to 15 minutes
- **Sensor data:** accelerometer, gyroscope, magnetometer, touchscreen input (including raw touch events, tap gestures, scale gestures, scroll gestures, fling gesture), key press events

BB-MAS [16] collected sensor data from 117 users across desktops, smartphones, tablets while they were walking or typing. Different from the HMOG dataset, BB-MAS users did not perform typing/browsing tasks while walking, and therefore, gait events and touch events were not available at the same time. Besides, in comparison to HMOG, the length of each walking task in BB-MAS is significantly shorter (i.e., less then 10 minutes per user per device), which may be insufficient to provide training and testing data for good model accuracy. We list the dataset details as follows:

- **Activities:** typing, browsing, walking, climbing stairs
- **# of tasks per user:** 12 (3 typing/browsing tasks, 6 walking tasks, 3 climbing tasks).
- **Range of task lengths:** 25 to 50 minutes (typing/browsing tasks), 1 minute (walking/climbing tasks)
- **Sensor data:** accelerometer, gyroscope, magnetometer, raw touch events, key press events

IDNet [54] collected accelerometer data and gyroscope data from 50 users while they were walking with the device in the right front pocket of their trousers. Thus, the IDNet dataset contains gait data only. We list the dataset details as follows:

- **Activities:** walking
- **# of tasks per user:** 1 to 14 walking tasks
- **Range of task lengths:** 5 minutes
- **Sensor data:** accelerometer, gyroscope

Touchalytics [52]. collected touch events from 41 users while they were interacting with Android smartphones. Since the screen interaction tasks are mainly about navigation maneuvers, touch data collected by Touchalytics is mainly for up-down and left-right swipes. We list the dataset details as follows:

- **Activities:** article reading, image comparison game.
- **# of tasks per user:** 5 to 7 (3 to 4 reading tasks, 2 to 3 game tasks)
- **Range of task lengths:** not specified
- **Sensor data:** raw touch events

2.8 Common Metrics

We adopt a set of common metrics to evaluate the performance of our approaches and compare to the state of the art. Both authentication and context sensing problems can be regarded as classification problems. Each time when a feature vector is extracted from collected sensor data, the system classifies it to a certain category. For binary classification, the sign of an instance indicates whether the current user is legitimate (i.e., positive) or not (i.e., negative), or whether a specific context (e.g., device loss context) is happening (i.e., positive) or not (i.e., negative). There are four outcomes depending on whether a classification result is true or not:

- A true positive or true acceptance (TP/TA) means that an authentication system accepts a legitimate user, or a context sensing technique successfully detects a specific context.
- A true negative or true rejection (TN/TR) means that an authentication system rejects an attacker/stranger, or a context sensing technique correctly detects that a specific context is not happening.

- A false positive or false acceptance (FP/FA) means that an authentication system falsely accepts an attacker/stranger, or a context sensing technique falsely detects a specific context.
- A false negative or false rejection (FN/FR) means that an authentication system falsely rejects a legitimate user, or a context sensing technique misses a specific context.

We adopt different sets of metrics to measure the accuracy of context sensing techniques and authentication systems, respectively. For evaluating context sensing techniques, we focus on its ability of retrieving the positive instances, i.e., detect a context. Thus, we use precision and recall for accuracy measurement.

- **Precision:** the fraction of all detected instances that truly contains the target context:
$$\text{Precision} = \frac{\#\text{ of TP}}{\#\text{ of TP} + \#\text{ of FP}}. \tag{2.3}$$
- **Recall:** the fraction of all instances with the context target that are successfully detected:
$$\text{Precision} = \frac{\#\text{ of TP}}{\#\text{ of TP} + \#\text{ of FN}}. \tag{2.4}$$
- **F1-score:** a measure that combines precision and recall:
$$\text{Precision} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{2.5}$$

As authentication systems are supposed to make minimal false detections, we emphasize their error rate in terms of false acceptance rate and false rejection rate.:

- **False Acceptance Rate (FAR):** the fraction of an attacker or stranger being falsely accepted:
$$\text{FAR} = \frac{\#\text{ of FA}}{\#\text{ of TR} + \#\text{ of FA}}. \tag{2.6}$$
- **False Rejection Rate (FRR):** the fraction of a legitimate user being falsely rejected:
$$\text{FRR} = \frac{\#\text{ of FR}}{\#\text{ of TA} + \#\text{ of FR}}. \tag{2.7}$$

Besides, for multi-user scenarios, it is possible for an authentication system to accept the current legitimate user but mistakenly identify this user as another legitimate user, which is a false identification (FI). Although both FA and FI cause the security problem of exposing a user's sensitive resources to another user, we differentiate two error types given the difference in attacker roles. Thus, we introduce **False Identification Rate (FIR)** to measure the fraction of a legitimate user being falsely identified as another legitimate user:

$$\text{FIR} = \frac{\#\text{ of FI}}{\#\text{ of TA} + \#\text{ of FR} + \#\text{ of FI}}. \tag{2.8}$$

Also, FRR for multi-user scenarios is updated to:

$$\text{FRR} = \frac{\#\text{ of FR}}{\#\text{ of TA} + \#\text{ of FR} + \#\text{ of FI}}. \tag{2.9}$$

For comparisons among different classification models with various threshold settings, we also use a receiver operating characteristic (ROC) curve to evaluate their performance, where the x-axis is the false positive rate (i.e., FPR, also known as FAR) and the y-axis is the true positive rate (i.e., TPR, also known as recall). There are two related metrics to summarize the ROC curve:

- **Area under curve (AUC):** the area under the ROC curve, which is the probability of the classification score for a random attacker being lower than that for a random legitimate user [66].
- **Equal error rate (EER):** the intersection point on the ROC where FPR $= 1 -$ TPR. It determines the threshold where the FPR and the FNR have the same value.

Eberz et al. [41] propose to use the Gini coefficient (GC) to analyze the error distributions among users and quantify systematic errors. GC is calculated between the area between the Lorenz Curve and the Line of Equality. For evaluating IA systems, the Lorenz Curve plots the percentiles of the users on the x-axis according to an error rate and plots the cumulative error rate on the y-axis. A point (x, y) on the curve indicates the normalized total error rate y contributed by the bottom x users. The Line of Equality is a straight diagonal line with a slope of 1, which represents that all users contribute to the same error rates. A high Gini coefficient means that errors are concentrated in a small group of users. In this thesis, we use GC and Lorenz Curve to investigate how the accuracy of a system is improved and what is the bottleneck for further improvement.

Chapter 3

Active Acoustic Sensing Based Smartphone Loss Prevention

3.1 Introduction

In this chapter, we address Objective 1 by proposing a smartphone loss prevention solution. A smartphone loss prevention solution first needs to address a comprehensive context sensing problem that involves two user-related context factors: proximity and activity. Then, the solution should automatically lock the device and alert the owner of a potential loss event. The main challenge is how to make the phone track the user's departure in a *contactless* way, where the phone senses the user's motion without the user carrying it. Given that smartphones are equipped with at least a pair of microphone and speaker, they are capable of active acoustic sensing. Hence, we adopt active acoustic sensing to detect the user-related contextual factors for smartphone loss prevention. The basic idea is to make a device continuously send acoustic signals and analyze the reflected signals from nearby object to determine a user's proximity and movement.

3.1.1 Threat Model

Our focus is on the threat posed to an unattended smartphone by nearby opportunistic attackers. We assume the attackers do not have any knowledge of our loss prevention solution, and their target is to access the unattended smartphone. To start with, the smartphone is placed stationary on a surface intentionally (e.g., the owner puts it on a

table), or unintentionally (e.g., the phone slips from the owner's pocket). Its microphone and speaker are not covered by other objects so that the transmission of sound is not blocked. (We examine the impact of nearby objects that partially block sound transmission in §3.5.3.) We assume the device owner is initially closer to the phone than others, including nearby people and the attacker, and the initial distance between the owner and the device is under 1m. This condition ensures that the device is initially in a relatively *secure* context compared to the later unattended status. We discuss active attackers and other complicated situations in §3.7 (e.g., when a stranger is closer to the device than the owner).

After the initial placement, the owner may move away from the device, thereby exposing it to theft or unauthorized access. The attack may happen within a few seconds after the phone becomes unattended (i.e., when the owner moves away from the phone). A potential smartphone loss is defined as a smartphone owner leaving the phone behind in a public or untrusted place. We propose a preventive approach that can detect a potential smartphone loss situation, lock the phone, and generate an alert before the owner leaves the place. More than just putting a threshold on the distance from the smartphone, our approach detects the owner's departure and absence from the phone (i.e., the owner keeps moving away from the phone and is eventually absent). Therefore, in our experiments, we do not have a specific attacker role given that the detection should occur *before* the attack happens. Instead, we consider the influence of nearby people on our sensing approach, which captures the reflected signals from the owner (see §3.2) and other people and objects. Note that we use the terms *owner* and *user* interchangeably.

3.1.2 Design Goals

An effective smartphone locking and loss prevention solution should have the following desirable properties:

Standalone. While leveraging specialized hardware (e.g., Surround-See [184]) can provide the required sensing ability to detect a nearby owner's movement, a solution that works on common off-the-shelf smartphones is more likely to be adopted. Similarly, while an accessory (e.g., a smartwatch) connected to the smartphone can detect smartphone loss, a standalone solution relieves users from carrying an additional device.

Low detection delay with low energy consumption. We use the term detection delay to refer to the time period during which the owner is unaware of the device loss. For post-loss solutions, this delay may be large as they are dependent on the owner's realization of the device loss. In a loss prevention solution, the detection delay corresponds to the time duration between the device owner leaving, and the solution realizing that the

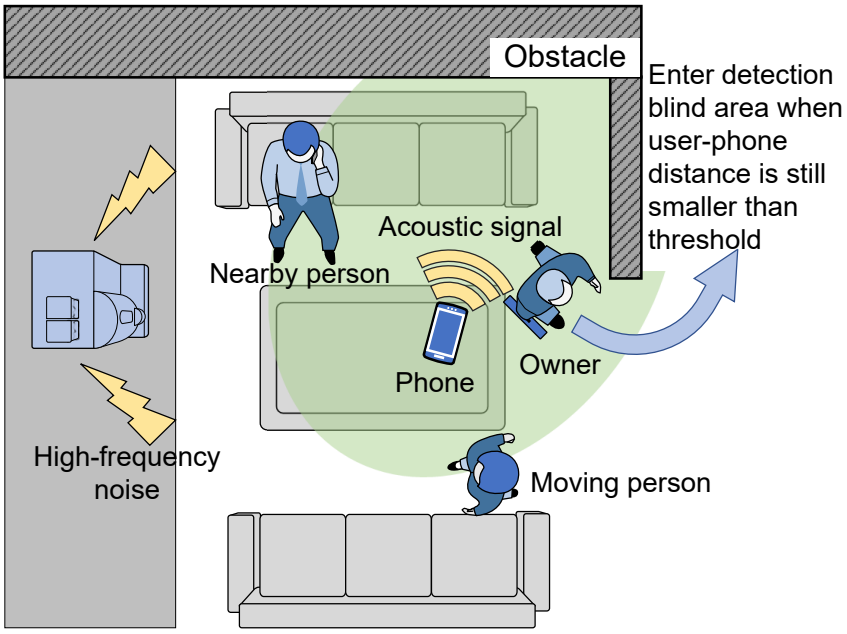


Figure 3.1: Potential factors that affect acoustic sensing. The green area depicts the detection range. The smartphone owner enters the detection-blind area caused by the obstacle while still being within the distance threshold, making the detector fail to follow the owner and track a nearby person instead.

owner is not present near the device, in turn, locking the phone. Thus, it is desirable to have low detection delay. However, low detection delay requires frequent sensing to ensure real-time detection. The local analysis of the acoustic data on the mobile device could be computationally intensive and consume significant battery power. Thus, we need to balance detection performance and energy consumption.

Few false positives and false negatives. A closely related usability aspect is the number of false positives. For example, the smartphone owner may move to grab something from across the table in a restaurant, which may be misconstrued as the owner leaving by a solution with low detection delay. False positives are inconvenient and may negatively affect the adoption of a solution. Therefore, the solution should notify the user in real-time, while limiting the number of false positives. Similarly, the solution should have few false negatives, i.e., failure to detect actual user leave events. False negatives may cause device loss; therefore, the system should minimize false negatives even at the cost of increasing false positives.

Robust. In practice, smartphones are lost at a variety of locations including coffee shops, restaurants, cars, etc. [39, 170]. Location diversity implies different levels of background noise, nearby moving people, and obstacles in the physical layout of the location. Figure 3.1 shows an example of these factors in a small lounge scenario. In terms of background noise, active acoustic sensing for smartphones usually uses the high-frequency band up to 24kHz (see §3.2), and as a result, high-frequency background noise poses a threat. Such noise is often encountered in real-world scenarios, e.g., slamming of a door. A high-frequency noise source may emit noise for a short period of time, but it is likely to happen more often at certain locations (e.g., a restaurant). Therefore, it is important for a robust system to deal with high-frequency noise. The movements of other nearby people introduce more reflections of sound signals, and thus require careful consideration. In terms of layout, a location's physical layout may introduce obstacles, limiting the effective operational range of acoustic sensing. In Figure 3.1, the range where active acoustic sensing can effectively receive the echoes is limited by the lounge layout since the acoustic signal is blocked or reflected by the obstacles. If the owner follows the blue arrow, the phone fails to track the echo from the owner after the owner moves behind the obstacle. In summary, the solution should robustly operate across a variety of locations, and require minimal or no location and environment-specific tweaking.

3.1.3 Contributions

We present `Chaperone`, a real-time smartphone locking and loss prevention solution using active acoustic sensing. Chaperone focuses on capturing a user's departure patterns and addresses the aforementioned challenges by tracking the departure procedure of the device owner across three dimensions (in reference to the smartphone): the motion state of the owner, the intensity of the motion, and the distance of the owner from the device. By incorporating multiple factors, Chaperone provides a robust real-time mechanism to detect when the user is *about to* leave the premise. The contributions include:

1. We design and implement Chaperone, a standalone, active acoustic sensing-based system that detects possible smartphone loss incidents in real-time on commodity smartphones. Chaperone requires no per-user training to operate in a new situation. Although it needs access to the device's microphone and speaker, Chaperone's standalone nature preserves privacy of the device owner and bystanders, as our carefully designed implementation does not offload any computation from the smartphone.

2. We conduct 1,345 experiments to demonstrate Chaperone's ability to operate under different conditions (including device orientations and positions, user leaving

speeds, distances to nearby stranger, close objects, and concurrent sensing by multiple devices), and cover various real-world scenarios characterized by high-frequency ambient noise, crowded locations, and diverse layouts (including academic venues, restaurants, offices, cars, and transit stations). This is the first such comprehensive evaluation of active acoustic sensing in real-world scenarios compared to existing literature [26, 108, 120, 121, 146, 163, 169, 176, 186, 188, 191, 192, 194].

3. Chaperone provides an overall precision of 93% and an overall recall of 96%, outperforming iLock [108] (see §3.4 for details) by 14% in both precision and recall scores. Specifically, in complex real-world scenarios (e.g., lounge and bus stop), the performance gain is up to 32% in the recall score. For 95% of the successful loss detection experiments, Chaperone can lock the phone and alert the owner within 0.5 seconds. The experimental results provide strong indication that Chaperone is robust and effective in many everyday scenarios.

4. We conduct a user study (n = 17) to investigate people's smartphone loss experiences, collect feedback on using Chaperone, and study user perceptions of different alert methods for smartphone loss prevention. The results indicate that the participants are satisfied with the detection performance of Chaperone. We also report on the suitability of five alert methods for different locations.

5. We release Chaperone as an opensource, standalone Android app, and our collected dataset from both lab and real-world experiments, to help reproduce our findings, and improve acoustic sensing-based device loss prevention solutions. The project link is https://github.com/cryspuwaterloo/chaperone.

3.2 Chaperone

We leverage active acoustic sensing based on a high-frequency acoustic signal, which is inaudible to most humans and is not interfered by common noise in the lower-frequency band. The speed of sound is orders of magnitude greater than the speed of a person moving away from the device—sufficient for real-time detection. Chaperone consists of four main modules: trigger, acoustic sensing, user tracking, and decision making module; see Figure 3.2.

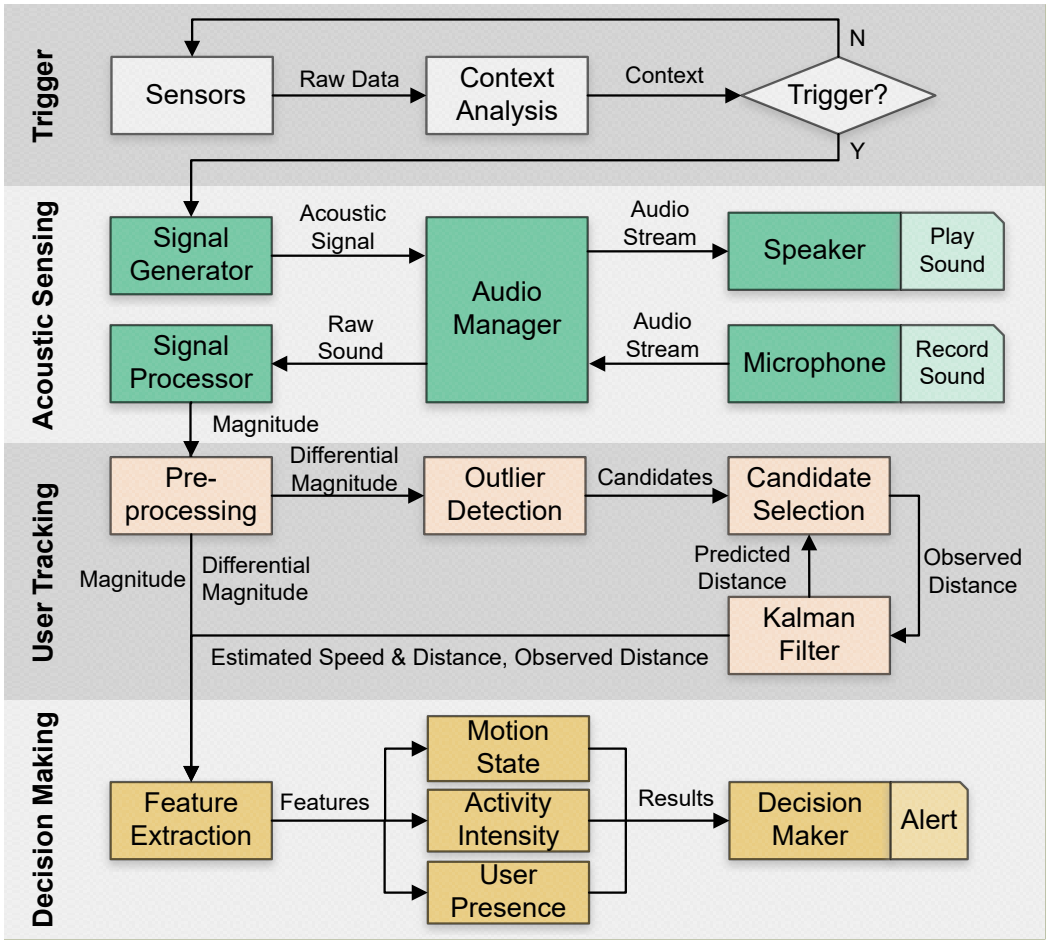


Figure 3.2: Workflow of Chaperone.

3.2.1 Trigger Module

Chaperone does not need to continuously perform active acoustic sensing for many scenarios including the following:

1. The user is holding the device, or it is on the user's body. Google's Activity Recognition API provides this information using low-power sensors [84].

2. The user is using the device while it is lying on a surface, e.g., playing a video while the device is on a desk. This can be determined by querying the device state to establish whether the device screen is off and it is in idle state.

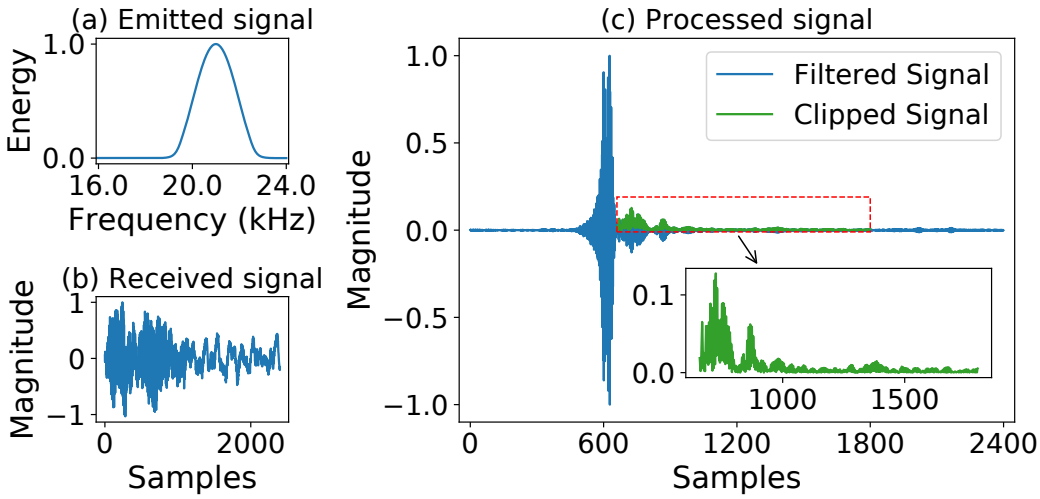


Figure 3.3: Signal processing in the acoustic sensing module (note: magnitude in the figure is normalized).

3. The device is at a *trusted* location, e.g., the user's home; such locations can be configured by the device owner.

In summary, the trigger module invokes active acoustic sensing only when the device is not in use (i.e., idle), not on the user's body, and in a potentially *untrusted* or *public* environment. This reduces the acoustic sensing overhead.

3.2.2 Acoustic Sensing Module

This module performs active acoustic sensing to keep track of the user's movement. It sends a particular acoustic signal, and processes the received echo signal to make meaningful conclusions about the user's movement (if any). It consists of an acoustic signal generator, audio manager (controlling the speaker and the microphone), and a signal processor.

The signal generator produces an inaudible acoustic signal based on sampling rate, frequency, length, and signal type, and then passes the audio data to the audio manager. We use a sampling rate of 48kHz for supporting the acoustic signal up to 24kHz and a sensing period (i.e., a frame) of 50 milliseconds for real-time detection. In the first phase of the sensing period, the device emits a 1,200-sample acoustic signal and keeps recording the sound; see Figure 3.3a. In the following 1,200-sample idle phase, the device emits no signal but continues to sense for the reflection of the signal emitted during the first phase.

The default acoustic signal used in Chaperone is a frequency sweep from 19–23kHz with fading at the start and the end of the signal, which is inaudible to most humans [46].

The audio manager interfaces with the smartphone's speaker and microphone. It simultaneously uses the speaker to periodically play the acoustic signal and the microphone to record the sound; see Figure 3.3b for an example of the raw sound. Since the recorded sound covers the whole frequency range, including environmental noise, the audio manager continuously passes the raw sound data to the signal processor to extract the reflected acoustic signal.

The signal processor is designed to obtain a magnitude vector $\mathbf{m}$ of the echoes. It first applies two filters, a band-pass filter and a matched filter, to the raw sound data to match the original acoustic signal. The band-pass filter keeps the dedicated frequency band, and the matched filter highlights the original acoustic signal by calculating the convolution of the filtered sound signal and the reversed original acoustic signal. Since it is impossible for an echo to occur before the direct transmission, we only keep the samples after the first peak (i.e., the sample with the locally highest magnitude caused by the direct transmission from the speaker to the microphone), and then obtain the processed acoustic data; see Figure 3.3c. The signal processor then calculates the magnitude vector $\mathbf{m}$ for the clipped signal. Since the delay of an echo is the round-trip time of sound traveling between the phone and an object (or user), each index of the vector can be mapped to the corresponding distance d according to the following time-of-flight distance measurement formula:

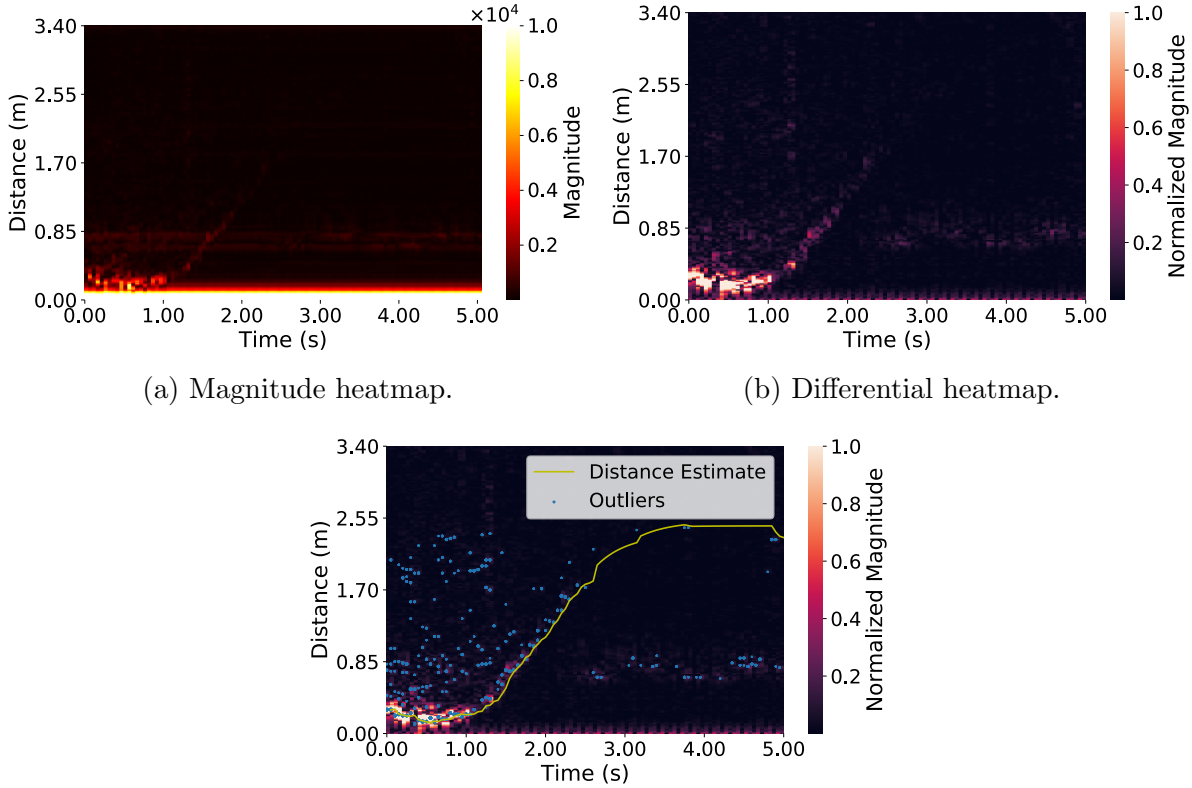
$$d = \frac{Mc}{2f_s} \cdot i, \tag{3.1}$$

where c is the speed of sound, f_s is the sampling rate of the acoustic signal and M is the downsampling rate. For example, given that $c = 340$ m/s, $f_s = 48$kHz and $M = 4$, the 10th element of vector m is the magnitude of the matched signal that is approximately 0.142m away from the phone. Finally, the signal processor passes the magnitude vector $\mathbf{m}$ for the current frame to the user tracking module.

3.2.3 User Tracking Module

This module locates the user by filtering echoes reflected from surrounding objects and background noise, and tracking the user among other moving bodies.

Pre-processing. In the first step, the pre-processing sub-module filters the echoes reflected from other objects. Figure 3.4a shows that the magnitude vectors capture echoes



(a) Magnitude heatmap.

(b) Differential heatmap.

(c) Outliers and distance estimate.

Figure 3.4: Distance estimation procedure: (a) the bright part represents the captured echoes from nearby objects and people; (b) after excluding echoes from static objects, the user's movement from time 0–2.5s is highlighted, but we can still observe the echo from nearby people, e.g., 85cm away from the device during the time period 2.5–5s; (c) by using our candidate selection algorithm, we can track the user's movement and predict the movement when there is no valid observation (e.g., at time 3s).

from the user as well as objects. We remove echoes from static objects by using the differential magnitude vector $\Delta\mathbf{m}_t = \mid \mathbf{m}_t - \mathbf{m}_{t-1} \mid, t \in \mathbb{N}^*$, which is the absolute difference between the current and the previous magnitude vectors. Figure 3.4b shows that this step excludes static objects and highlights echoes from the user. The pre-processing sub-module also determines if the current frame is affected by background noise. The overall magnitude of the differential magnitude vectors at the corresponding moments may become irregularly large due to high-frequency noise (see §3.1.2); we thus set a threshold on the average value

Algorithm 1 Candidate Selection Algorithm

Input: All m candidate tuples $C_m = \{(s_m, h_m, l_m)\}$ where s is starting distance, h: peak magnitude, l: cluster size; $\hat{d}$: predicted distance ; n history speeds $\tilde{\mathbf{v}} = \{v_0, v_1, \ldots, v_{n-1}\}$; $R_{\max}$: max range; q: base discount

Output: Observed distance *obs*

function CANDIDATE_SELECTION($C, \hat{d}, \tilde{\mathbf{v}}$)
  $obs \leftarrow -1, p_{\max} \leftarrow -1, e \leftarrow 0$ ▷ Initialization
  $\kappa_0 \leftarrow$ `getDirection`(v_{n-1})
  for $i \leftarrow n-2$ to 0 **do**
    $\kappa \leftarrow$ `getDirection`(v_i)
    if $\kappa = \kappa_0$ and $\kappa \neq 0$ **then** ▷ If direction changes
      $e \leftarrow e + 1$ ▷ Add to discount exponent
    else break
  for $i \leftarrow 0$ to $m-1$ **do**
    $s_i, h_i, l_i \leftarrow C_i$, $r \leftarrow q^e R_{\max}$ ▷ discounted range r
    if $\mid s_i - \hat{d} \mid \leq r$ or $\mid s_i + l_i - \hat{d} \mid \leq r$ **then**
      if $h_i > p_{\max}$ **then**
        $obs \leftarrow s_i, p_{\max} \leftarrow h_i$
  return *obs*

to exclude such noisy frames. Note that if a frame is regarded as noisy, there is no valid observation at that moment. This error is adjusted by predicting the current distance based on the values from the previous frames using a Kalman filter.

Outlier detection. This sub-module detects potential dynamic movements of the user. Intuitively, an outlier (i.e., an exceptionally large magnitude) in a differential magnitude vector implies the existence of motion at the corresponding distance. We use median-absolute-deviation (MAD) outlier detection to obtain the outliers in the current frame. However, our outlier detection may be negatively affected by the motion of the user's body parts and the motion of other nearby people. Specifically, the intense motion of a user's body parts results in a non-trivial number of outliers; see the blue dots in Figure 3.4c. We handle these outliers by clustering them based on their relative distance, so that they are merged into a single candidate.

Candidate selection and Kalman filter. From the clustered candidates, we choose the candidate that corresponds to the user and use it to estimate the user-device distance and the user's speed. For the first frame (at $t = 1$), we choose the candidate closest to the

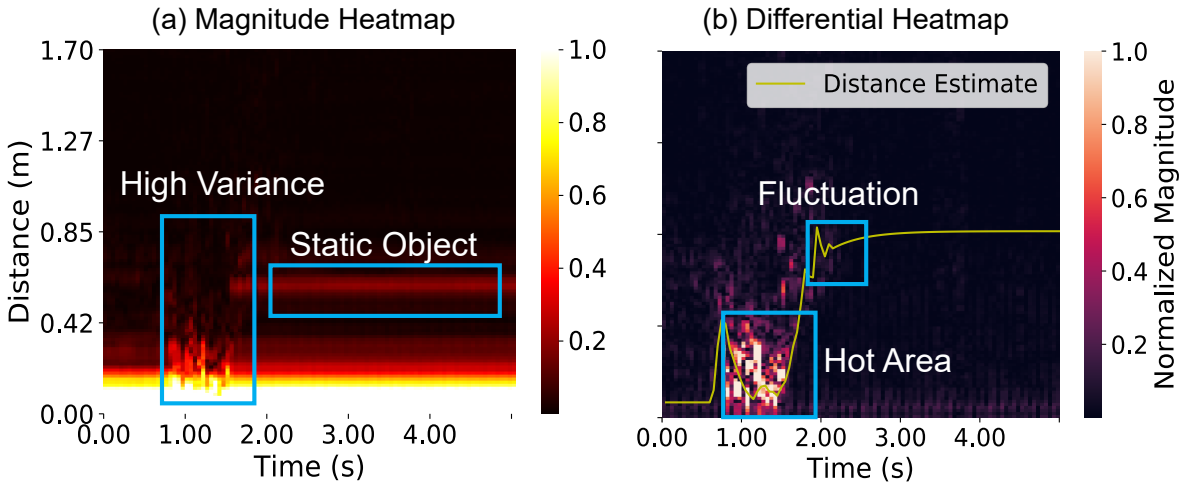


Figure 3.5: Example of distance tracking failure: the user tracking module can only track the user up to about 85cm.

phone, assuming that the user is the closest, and then feed the corresponding distance into the Kalman filter as the initial distance. Once the user is in motion, our assumption that the user is closest to the device may no longer be valid. For example, in Figure 3.4c, we can observe movement of another person at the distance mark of 0.8m (and at time 2.5s), while the user is actually 1.7m away from the phone. To address this scenario, we make the candidate selection and the Kalman filter work together to decide which candidate point to choose as the observed distance d_t at time t. The Kalman filter is also used to estimate both distance and speed. For candidate selection among the following frames, we reduce the candidate selection range if the user keeps the previous motion state; see Algorithm 1.

Since the Kalman filter itself predicts the current distance and speed at each round, we incorporate the a priori estimate of distance $\hat{d}_{t|t-1}$ (i.e., "predicted distance") from the Kalman filter to calculate the possible range for the next distance. The candidate selection module chooses the most consistent candidate based on the magnitude and uses its corresponding distance as the observed distance. Then, the Kalman filter updates the a posteriori estimate of distance $\hat{d}_{t|t}$ and speed $\hat{v}_{t|t}$ at time t. (We denote them as "estimated distance" and "estimated speed".) Note that if the user is stationary, or out of the detection range, there might be no matching candidate points. In that case, the Kalman filter is fed with the previous distance as the observation, assuming that the user is idle. After combining multiple frames, we obtain a trace of the user's movement based on the distance estimated by the Kalman filter (the yellow line in Figure 3.4c). All the distance and speed

values, together with the magnitude vectors, are passed to the decision making module to determine whether to generate an alert.

3.2.4 Decision Making Module

This module detects whether the user is about to leave the device, based on the information obtained from the acoustic sensing and user tracking modules. As noted in §3.1.2, several environmental factors can limit the detection capabilities in real-world scenarios; see Figure 3.5, where the user tracking module fails when a distance-only approach is employed with the distance threshold set at 1m. As a result, a simple distance-only approach is unable to determine whether the user is stationary at that point or is behind the wall. Therefore, dealing with obstacles requires a more comprehensive analysis than relying on the estimated distance alone.

Classifiers for user state estimation. We rely on three classifiers: the motion state classifier determines whether the user is approaching, leaving, or stationary; the activity intensity classifier determines whether the user's activity is intense or moderate; and the user presence classifier determines whether the user is close to the device or far away. The features for these classifiers are derived from distance, speed, magnitude vector and differential magnitude vector estimates of the user tracking module; Table 3.1 lists our features and their usage in the classifiers. Feature values are populated by combining data from multiple continuous frames into one window. The window size w is set to five frames (i.e., 250ms), containing sufficient information to perform meaningful analysis without affecting the real-time capability of Chaperone. Within each window, we denote the first frame as t_1 and the last frame as t_w. As for the (differential) magnitude vectors, we focus on movements in the 15cm–1m range. A lower bound of 15cm excludes any direct transmissions from the speaker to the microphone, and our experiments show that an upper bound of 1m provides sufficient data to reliably detect smartphone loss.

Features for classification. Intuitively, speed and distance features are correlated to the user's motion and presence state. From the user tracking module, we know whether it has a valid observation on the user's motion, and then we can obtain both observed and estimated distances and speeds. We also calculate the relative distance to the median of historical user-device distances, approximating the user's initial distance to reduce fluctuations caused by the user's activity. Besides, we employ the average speed, which is the slope of the line connecting the distances of the first and the last frames.

We also consider intensity-related features. Figure 3.5b shows that when the user is performing activities, such as typing or standing up, the movement of different body parts

Features	Formula or Description	C1	C2	C3
IsObserved	whether user tracking module makes a valid observation	●	●	●
Distance	Observed distance: $d_{\text{obs}} = \sum_{i=t_1}^{t_w} d_i/w$ Estimated distance: $d_{\text{est}} = \sum_{i=t_1}^{t_w} \hat{d}_{i\|i}/w$ Difference from median: $\Delta d_{\text{est}} = d_{\text{est}} - \text{median}\{d_t\}$ $\Delta d_{\text{est}} = d_{\text{est}} - \text{median}\{d_{t\|t}\}$	●	●	●
Speed	Observed speed: $v_{\text{obs}} = (d_{t_w} - d_{t_1})/w$ Est. speed: $v_{\text{est}} = (\hat{d}_{t_w\|t_w} - \hat{d}_{t_1\|t_1})/w$ Numerical avg. speed: $\bar{v} = \sum_{i=t_1}^{t_w} \hat{v}_{i\|i}/w$	●	●	
Fluctuation	# of direction changes in est. speed	●	●	
Magnitude	$\bar{m} = \sum_{i=t_1}^{t_w} \sum_{j=d_0}^{r} \Delta\mathbf{m}_{i,j}/wr$	●	●	●
Hot area rate	$h = \sum_{i=t_1}^{t_w} \sum_{j=d_0}^{r} \mathbb{1}\{\Delta\mathbf{m}_{i,j} > \theta\}/wr$	●	●	●
Row variance	$\sigma_d = \sum_{i=t_1}^{t_w} (\mathbf{m}_{i,d} - \mu_d)^2/w$, $\mu_d = \sum_{i=t_1}^{t_w} \mathbf{m}_{i,d}/w$		●	●
Static object	# of static objects changed in $\mathbf{m}$			●

Table 3.1: Features for three classifiers. C1: motion state; C2: activity intensity; C3: user presence. A circle means a classifier uses the corresponding feature (empty indicates no use).

leads to the average differential magnitude close to the phone being dramatically larger (called a "hot area") than the ambient magnitude. Therefore, to describe the user's activity intensity, we use the average differential magnitude and the hot area rate, the proportion of the area whose magnitude is larger than a threshold θ. Besides, these activities may result in some fluctuations in the speed and distance estimation, which can be observed in frequent changes of the direction.

The magnitude vector also provides information about user presence; see Figure 3.5a.

Even slight movement of the user can still cause an increase of variance in magnitude at the corresponding distance, implying the user's presence. Furthermore, it is possible to infer the user's presence based on the static objects nearby. When the user is near the phone, parts of the acoustic signal will be blocked by the body, and the objects behind the user may not appear on the spectrum. But after the user has left, these objects will begin to reflect the signal, and thus change the raw magnitude vector.

Decision maker. This sub-module determines the user state, and reacts based on the classification results of the three classifiers. We adopt a sliding window mechanism to make a decision across three windows, which improves the detection accuracy without sacrificing the real-time nature of the system. The decision maker uses the following criteria to decide whether a departure activity of the user happens: The user is leaving (i.e., the motion state classified as "leaving"), the activity intensity is fading (i.e., the activity intensity changed from "intense" to "moderate"), and lastly the user is no longer close to the device (i.e., the user presence state changed from true to false). Only when the user's movements satisfy all criteria, Chaperone will make a positive detection. This strategy helps reduce false positives by a distance-only approach.

As a reaction to a potential smartphone loss, Chaperone locks the phone immediately and triggers an appropriate alert method using, e.g., a ringtone, vibration, notification sound, or screen flashing. The alert scheme is chosen based on the contextual information collected by the trigger module. For example, if the environmental noise level is low, a gentle ringtone will be sufficient to get the user's attention. In §3.6, we systematically investigate user preferences for alert methods in terms of effectiveness and annoyance in different scenarios.

3.2.5 Implementation

We implement a Chaperone prototype as a standalone Android app. To help reproducibility, we also implement a remote-mode option, where the smartphone is responsible only for acoustic sensing, and a remote server stores and analyzes the raw acoustic data for user tracking and decision making.

For acoustic sensing, we use LibAS [167], an opensource framework for the rapid development of acoustic sensing apps. LibAS outputs the acoustic signal used by Chaperone and performs acoustic sensing. The operations required for user tracking and decision making (see Figure 3.2) are not provided by LibAS, so we had to implement them ourselves. The minimum SDK supported by Chaperone is API level 21. Audio data is collected in the

raw audio mode for Android 7.0 and up or using the microphone audio source for below Android 7.0.

Support for different smartphones. For most experiments, we use a Google Pixel (2.15GHz quad-core CPU, 2016) for data collection to train the classifiers in the decision making module. We successfully tested the prototype on Samsung S8, Huawei AL-10, and Google Pixel, Pixel 3, Nexus 5x, and Nexus 6P phones. Because of hardware differences, the magnitude scales of acoustic signals vary on different devices. To make Chaperone work on different devices, an additional configuration step is needed. First, we adjust the volume of the target phone to approximate the original acoustic signal strength to the Pixel. Then, we sample the received signal and map the magnitude scale of the target phone to it. This one-time configuration step is needed before deployment so that the classifiers can be used on other devices without retraining.

Latency. To balance detection performance and signal processing overhead, we set the sensing period to 50 ms (see §3.2.2) and implement filters in native C for efficiency. It takes 25–35ms on the Pixel to generate raw magnitude vectors from the acoustic signal. User tracking considers echoes only within two meters from the device, which is sufficient for device loss detection, and takes less than 1ms to extract features. The decision making module uses pre-trained models and takes about 1–2ms for classification (see §3.4 for details). As a result, the overall latency of Chaperone for each sensing period is about 45ms on the Pixel. On the Nexus 5x (1.8GHz hexa-core CPU, 2015), processing takes 60ms, while on the Pixel 3 (2.5+1.6GHz octa-core CPU, 2018), it takes only 35ms. Therefore, Chaperone is effective for new and old devices.

Silent mode. When acoustic sensing is triggered on a device in silent mode, the media volume is set to high for exclusively sending inaudible acoustic signals. The ringtone volume remains on silent. Since silent mode implies that the user is in a quiet environment, Chaperone can adopt vibration or flashing for alerts, instead of a ringtone. When acoustic sensing is terminated, the device resumes the normal silent mode.

3.3 Evaluation Setup

Logistics. To evaluate the detection performance of Chaperone, we conducted experiments that simulated different smartphone loss scenarios. For the ground truth, we need labeled acoustic data that indicates when a user is at a certain distance from the device. This requires at least an experimenter and an observer. The experimenter acted as the device

owner and performed a series of departing and everyday activities. We include scenario-specific everyday activities as they may introduce false positives (see §3.5 for details). The observer was responsible for real-time labeling of the departure events, t_d, and absence events, t_a. The departure event indicates that the experimenter is leaving the device, and the absence event indicates that the experimenter is 1m away from the device. The observer also labeled the user state information, which is used for the model training for the three classifiers. In total, eight experimenters (one undergraduate student and one graduate student who have no security background, six graduate students who have security background) simulated the device loss events in the experiments and one observer labeled the events for consistency.

Data collection. Our objective is to collect data from a diverse set of evaluation conditions and scenarios. We first controlled device orientation and the user's departing speed in lab experiments. Intuitively, when the microphone is facing the user, the echo reflected from the user is most effectively captured. But if a user puts the phone horizontally (i.e., 90°) on a table, the received echo signal is likely weak. As for departing speed, if the user departs quickly, the system's reaction time may be inadequate for real-time alerts. We collected 135 departure and 135 everyday activity events from an experimenter to evaluate nine combinations of these conditions (see §3.4.1). Another aspect that requires careful control is the effect of a nearby stranger on Chaperone—e.g., whether the departure of a nearby stranger results in a false positive, or the existence of the stranger when the user has departed results in a false negative. We collected 54 user-departure events and 54 nearby stranger-departure events with an experimenter and a stranger separated by three distances in a lab-based setup (see §3.4.2). Finally, we evaluated real-world conditions with varying factors (e.g., crowd, noise, and physical layout) at eight locations (library, office, restaurant, coffee shop, lounge, bus stop, in-vehicle, and academic venue). Eight volunteers helped to collect 366 departure events and 391 idle events; see Table 3.2. We comment on the environmental conditions of each location when we present the results in §3.5. In addition, we evaluated the effects of other interference factors by collecting 75 departure events in close-object experiments and 135 departure events in concurrent sensing experiments.

Each data collection experiment consists of two parts. In each scenario, the experimenter put the phone on a surface (e.g., dining table at a restaurant) within one-arm distance from the body. In the first part, the experimenter performed some everyday activities matching the given scenario. In the second part, the experimenter left at the requested speed. Each activity is about 2.5–10 seconds long. For layouts with multiple departing paths, the experimenter also took different paths. The observer was far away from the experimenter (more than 2m for lab experiments, at least 1m for real-world ex-

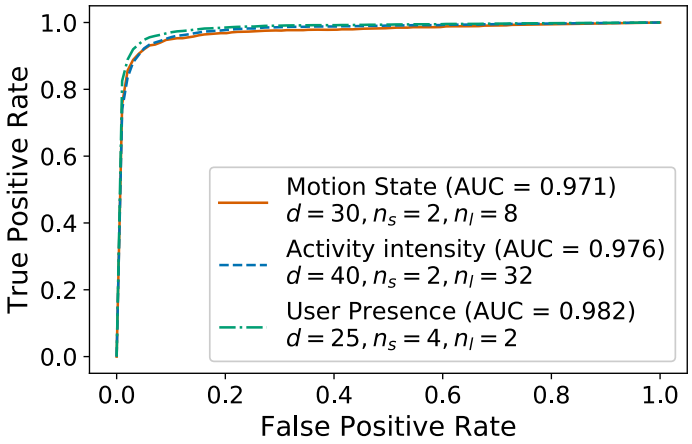


Figure 3.6: The ROC curve of the three classifiers.

periments) to capture the departing procedure. Finally, to measure the performance of Chaperone over longer idle periods, we collected 15–20 minutes of data in locations where the user stayed for a long time, such as libraries, meeting rooms, or restaurants. For these experiments, we count the total number of false positives in the given time duration.

Algorithms for comparison. We compare Chaperone with iLock's user-phone distance estimation approach [108]. We contacted the iLock authors for their implementation. Although we did not receive it, they provided implementation details missing from the paper. Combining with details from the related papers [2, 108], we implement iLock's distance estimation approach including background subtraction, peak finding, and a Kalman filter with outlier rejection. Given the available details, our implementation is close to the one by Li et al., although there may be minor differences. We label this algorithm as "iLock" for simplicity. We assume the phone will be locked and an alert will be raised whenever the estimated distance exceeds the threshold of 1m, as set by iLock [108]. iLock is prone to raise a positive detection for more involved scenarios to avoid false negatives. For example, when more than two users' movements are detected but only one exceeds the threshold, iLock locks the device without knowing whether it was the owner who crossed the threshold; this causes many false positives in the real-world experiments (see §3.5). To reduce false positives, we merge the candidate selection strategy from Chaperone into iLock, which we label as "iLock++". This change improves the peak selection of iLock to better track the owner's movement.

Metrics. We use precision and recall to measure the detection performance. We denote

a departing activity as a *positive instance* and an idle activity as a *negative instance*. We define a successful detection as one made after the moment t_d when the user starts to leave. Note that if a positive detection is made before t_d due to false tracking, it is counted as both a false positive and false negative (i.e., it creates a false alarm and fails to detect the true event). We also evaluate the time delay of the alerts. We use human observations as reference points and correct them based on acoustic sensing to offset the human reaction time. An alert is deemed valid if it is sent after t_d. We use the moment t_a when the user is observed to pass the 1m line as the zero-point for calculating the delays. Then the detection delay can be calculated as $\hat{t_d} - t_a$, where $\hat{t_d}$ is the time when Chaperone detects the departure. A negative delay means an early detection before the human observation of the user passing the 1m line.

Hyper-parameter tuning. Our three classifiers are responsible for interpreting the user's current status from a variety of features. The performance of the classifiers is critical for the final decision making. Therefore, it is necessary to tune the hyper-parameters of these classifiers before conducting the experiments. We adopt Randomized Search Cross Validation [138] to tune the three hyper-parameters of the Random Forest algorithm: tree size d, minimum sample number for splits n_s, and minimum sample number of each leaf n_l. We use the lab experiment dataset for tuning, and manually label 6,118 data points (i.e., windows) with the current user state. This dataset is also used to train the model used in real-world experiments. The tuning objective is to maximize the area under the receiver operating characteristic curve (AUROC). Figure 3.6 shows the average ROC curve of 20-fold cross-validation with the best hyper-parameter settings for the three classifiers. In the following experiments, we always adopt the hyper-parameters for model training listed in Figure 3.6.

3.4 Lab Experiments

3.4.1 Device Orientation and Departure Speed

We conducted experiments on nine different combinations of the following two factors—three phone orientation angles: 0° (vertical), 45°, and 90° (horizontal); and three departure speeds for the experimenter: *slow*, *normal*, *fast*. The logged departing speeds were experimenter dependent, and the average speeds were 1.25m/s (slow), 1.67m/s (normal), and 2.22m/s (fast). These experiments were conducted in a lab with a 70cm high desk. For each experiment, the phone was placed at the given angle on the desk in front of the ex-

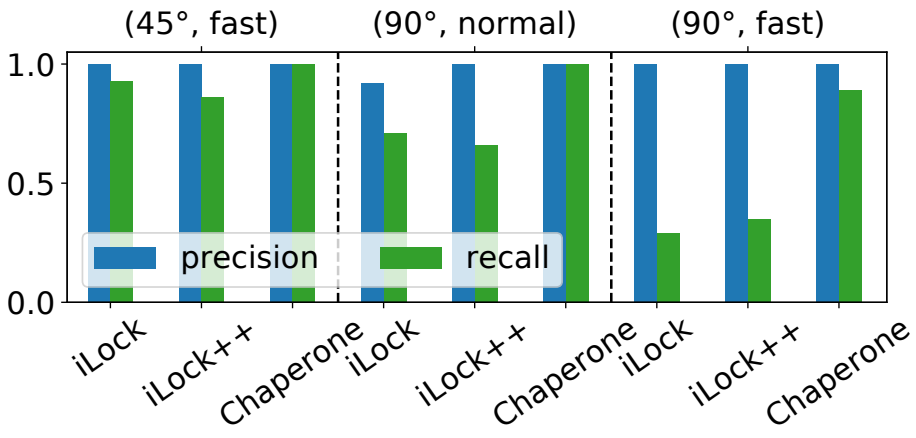


Figure 3.7: Precision and recall of iLock, iLock with Chaperone's candidate selection strategy (iLock++), and Chaperone.

perimenter and the experimenter stood at the desk about 20cm away from the phone. For each angle-speed combination, we logged 15 departure and 15 idle events.

Since Chaperone requires training the three classifiers, we use ten-time four-fold cross-validation to evaluate its detection performance. Namely, we split the data for all combinations into four subsets where data samples from different combinations are evenly distributed. We use three subsets for training and the fourth one for testing. The splitting is repeated for ten times, and eventually, we calculate the average precision and recall for each angle-speed combination. For iLock and iLock++, which are model-free, we directly evaluate their performance over each combination.

For angle-speed combinations (0°, fast/normal/slow), (45°, normal/slow), (90°, slow), all three algorithms achieved both 100% precision and 100% recall. Figure 3.7 shows the evaluation results for the three algorithms under the other three combinations. Even when the user departed at a fast speed and the phone orientation angle was 90°, the precision and the recall of Chaperone are 100% and 89%, respectively—a strong indication of robustness against different phone orientation angles and departure speeds. In comparison, if the user left at a normal or fast speed and the phone orientation angle was 90°, the recall scores of iLock and iLock++ decrease significantly. For the (90°, fast) combination, the recall score of iLock drops to only 29%, and iLock++'s is about 35% with successfully tracking two more departing activities based on the improved tracking strategy. The reason for the drop is that the strength of echoes from the user becomes weaker when the angle is larger, and

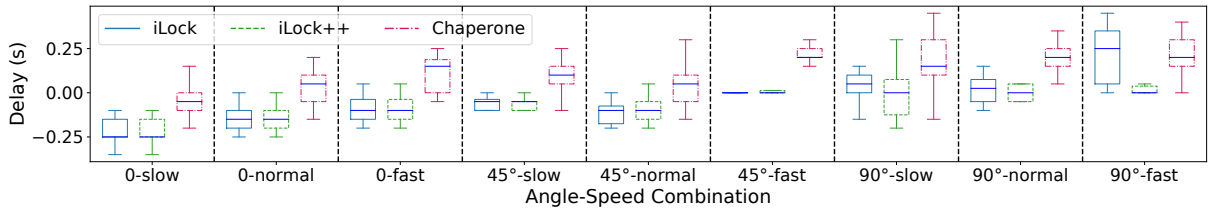


Figure 3.8: Detection delay (in seconds) for three algorithms.

the detection window is reduced due to the fast departing speed, where few valid measurements can be made by iLock and iLock++. They lost track of the user's trace before the 1m threshold under these conditions, which shows the ineffectiveness of the distance-only approach. Chaperone can still detect such situations based on the user state classifiers of the decision making module. All three algorithms have a very high precision score (i.e., no false positives for iLock++ and Chaperone), because the idle events performed in the ideal experiments were always close to the phone, which are easy to differentiate from a departure event. The experiment has shown that Chaperone, with the help of both the user tracking and decision making modules, outperforms iLock and iLock++ when handling a more complicated situation.

Figure 3.8 shows the detection delay for the three algorithms. Ideally, we expect an alert to be emitted within 1–2 seconds after leaving the phone to get the user's attention on time, i.e., while the user is still close. Chaperone consistently reacts within 400ms (95^{th} percentile) for all nine combinations after the user passed the 1m line; in contrast, iLock and iLock++ can react within 200ms due to their simpler detection strategy. Chaperone's window-based decision mechanism incurs a delay of 400ms, but it is still fast enough for real-time use.

In summary, Chaperone performs significantly better under several angle-speed combinations. Both iLock and iLock++ perform poorly when the orientation angle is large and the user's departing speed is high. Chaperone handles this situation well by tracking the user's departure pattern instead of relying on user-phone distance only. All three algorithms manage to detect departure events in real time.

3.4.2 Effects of a Nearby Stranger

We conducted controlled lab experiments to investigate how a nearby stranger affects the detection performance. We used the same layout as in §3.4.1, and conducted the

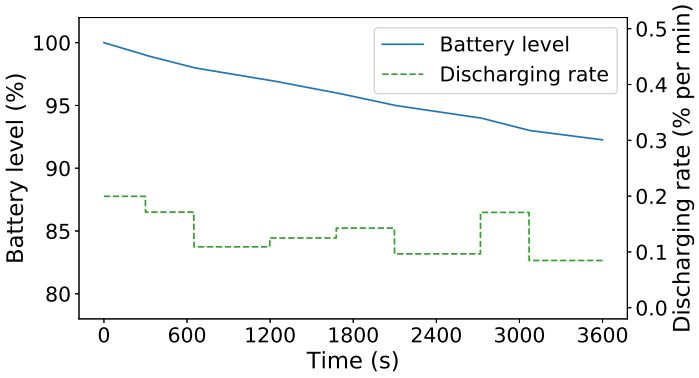


Figure 3.9: One-hour energy consumption when Chaperone is continuously sensing.

experiments as follows: Both the stranger and the user initially stood at the desk, and kept distances of 30cm, 75cm, and 100cm between them for different tests. The phone was vertically placed 20cm in front of the user on the desk. The user and the stranger were asked to alternatively depart from their positions.

For all three distance settings, Chaperone is able to detect all departure events with no false positives (precision and recall of 100%). iLock and iLock++, which are designed to defend against nearby attackers, also perform very well: among the 108 events, both algorithms had two false positives and one false negative for the 75cm user-stranger distance, and one false positive and one false negative for the 100cm user-stranger distance. The results show that interference from a nearby stranger has little impact on the detection in the lab environment. However, in real-world scenarios, there may be more than one person near the user. In addition, the activities from nearby people are unpredictable in terms of direction, intensity, timing, etc. Therefore, we further studied the potential of false positives/negatives resulting from nearby people in the real-world experiments; see §3.5.

3.4.3 Energy Consumption

Active acoustic sensing of Chaperone is triggered only when the Trigger module detects a potentially *vulnerable context* (e.g., at a bus stop). If the phone is in a private environment, e.g., home, Chaperone's processing needs will be negligent (i.e., no active acoustic sensing). However, Chaperone may still be occasionally triggered for a long period of time—e.g., the user is attending a conference, while leaving the phone on a table. Therefore, we use Android Battery Historian [58] to profile Chaperone's energy consumption on the Pixel with

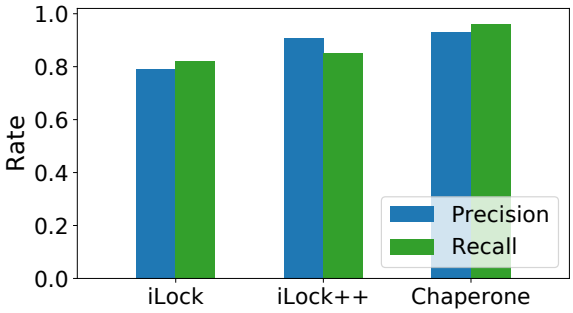


Figure 3.10: Overall performance of three algorithms across all locations.

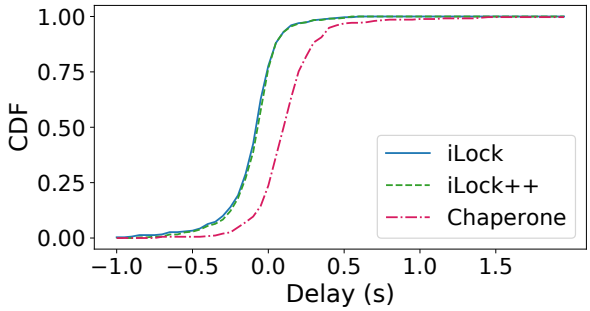


Figure 3.11: Detection delay. Negatives indicate detection before crossing 1m.

a 2770mAh battery. We fully charged the phone and kept it idle with no other applications running, except Chaperone and system services. We placed the phone on a table with the maximum volume, while Chaperone was continuously conducting detection; the battery level dropped from 100% to 92.3% in an hour—see Figure 3.9. The peak discharging rate was about 0.2% per minute, with an average of 0.13% per minute. For comparison, one-hour music playing with the same volume consumed about 4% of the battery, while one-hour movie playing consumed about 9% (the idle phone took about 0.3%). Although Chaperone's battery consumption during active acoustic sensing is significant, it is still acceptable for daily use with help of the trigger module — low-frequency sensing with motion and location sensors can help avoid unnecessary acoustic sensing and save battery. Our survey (see § 3.6) also showed that the extra battery consumption was acceptable for most participants considering their smartphone usage habits and Chaperone's trigger mechanism.

3.5 Real-World Experiments

We evaluated Chaperone against a variety of real-world scenarios. We did not employ scenario-specific data for training the classifiers. Instead, we trained them using the data that we collected from one experimenter during the lab experiments (§3.4), following our "Robust" design goal (§3.1.2, require minimum tweaking for unseen scenarios). Figure 3.10 shows the overall detection performance of the three algorithms over 366 departing activities and 391 idle activities in real-world scenarios. The precision and recall scores of Chaperone are 93% and 96%, respectively, compared to iLock's 79% and 82%, respec-

tively. With using Chaperone's candidate selection strategy, the precision of iLock++ increases up to 91% and the recall is slightly improved to 85%. Figure 3.11 shows the cumulative distribution function of the delay for the three algorithms; over 95% successful detection instances happen within 500 ms after the user crosses the 1m threshold. Although Chaperone has a longer delay than iLock, the delay gap is still acceptable. These results demonstrate Chaperone's efficacy in previously unseen real-world scenarios. Table 3.3 shows the precision and recall scores of eight experimenters in different locations. The three classifiers were trained with only one experimenter's (i.e., #3) data collected during the lab experiments. From the results, we can see that the pre-trained classifiers worked well for all eight experimenters, indicating that Chaperone is user-independent. We now discuss the results for the individual scenarios (summarized in Table 3.2).

3.5.1 Evaluation under Different Scenarios

Library. The experimenter shared a group study table with two or three students at our university library. Occasionally, strangers passed by near the table. The background noise came from people's chatting and the building's ventilation. The everyday activities involved reading and writing by the experimenter. In this environment, the detection rates of the three algorithms are mostly identical. As this scenario is close to the setting in the ideal experiments but with a few nearby strangers, iLock and iLock++ can also handle it well. The three algorithms shared a common false negative, caused by the simultaneous movements from both the user and a passer-by. The false positives were caused by interference from a nearby stranger's abrupt movements.

Office. The experimenter was alone in a narrow office cubicle and performing activities, such as using the keyboard and monitor, and standing up to fetch documents from a shelf. There was background noise from computers, typing, and a regular office swivel chair. The cubicle has a semi-open structure, and we placed the phone at different positions on the table. When it was placed close to a cubicle divider, the acoustic signals were partially blocked. iLock has significantly lower precision and recall scores since it failed to handle partially blocked signals well, or was misled by changes in the magnitude of the echoes from the chair. With Chaperone's tracking strategy, iLock++ has the same recall score as Chaperone and a slightly higher precision score. For Chaperone, ten false positives were a result of the user's movements matching the preset departure pattern of Chaperone. For example, three false positives came from the eight document fetching cases—when the experimenter momentarily came close to the 1m line and then returned. All the three false negatives were related to the false positives in departure activities where Chaperone sent alerts before t_d (see "Metrics" in §3.3).

Location	Departure	Idle	iLock		iLock++		Chaperone	
			Precision (FP)	Recall (FN)	Precision (FP)	Recall (FN)	Precision (FP)	Recall (FN)
library	46	50	0.98 (1)	0.98 (1)	1.00 (0)	0.96 (2)	0.96 (2)	0.96 (2)
office	54	87	0.70 (20)	0.85 (8)	0.89 (6)	0.94 (3)	0.84 (10)	0.94 (3)
restaurant	71	93	0.89 (8)	0.90 (7)	0.95 (3)	0.89 (8)	1.00 (0)	0.99 (1)
coffee shop	36	42	0.79 (8)	0.86 (5)	0.94 (2)	0.86 (5)	0.92 (3)	0.94 (2)
lounge	50	59	0.78 (9)	0.64 (18)	0.81 (7)	0.60 (20)	0.87 (7)	0.96 (2)
bus stop	45	27	0.68 (15)	0.71 (13)	0.88 (5)	0.78 (10)	0.92 (3)	1.00 (0)
in-vehicle	58	33	0.72 (18)	0.79 (12)	0.88 (7)	0.91 (5)	1.00 (0)	0.91 (5)
acad. venue	6	0	-	0.8 (1)	-	0.8 (1)	-	1.0 (0)

Table 3.2: Precision and recall of three algorithms in eight locations (FP: # of False Positives, FN: # of False Negatives).

User #	Departure	Idle	Precision (FP)	Recall (FN)	Location (# of cases)
1	50	67	0.87 (7)	0.90 (5)	office (69), in-vehicle (39), bus stop (9)
2	31	21	1.00 (0)	0.97 (1)	in-vehicle (52)
3	39	44	0.93 (3)	0.95 (2)	coffee shop (50), lounge (33)
4	17	31	0.84 (3)	0.94 (1)	library (20), coffee shop (28)
5	50	52	0.98 (1)	0.98 (1)	library (20), restaurant (44), lounge(38)
6	108	110	0.93 (8)	0.99 (1)	library (39), restaurant (92), bus stop (49), lounge (38)
7	29	30	0.96 (1)	0.93 (2)	library (17), restaurant (28), bus stop (14)
8	42	36	0.93 (3)	0.95 (2)	office (72), acad. venue (6)

Table 3.3: Per-user results of Chaperone and case distribution in eight locations (FP: # of False Positives, FN: # of False Negatives).

Restaurant and coffee shop. Since the layouts and results for the restaurant and coffee shop scenarios (one restaurant and two coffee shops) are similar, we present them together. For these scenarios, the experimenters were eating/drinking at different tables (e.g., round, corner, bar counter), and shared the tables with one or two nearby people (within 1m). Both types of places were noisy, crowded, and other customers were passing by. There was high-frequency noise from the entrance door, dragging of chairs, and dining carts in the restaurant; an espresso machine also sometimes produced high-pitched noise in the coffee shops. Chaperone performs very well in the restaurant: precision 100% and recall 99%. In coffee shops, three false positives from two experimenters have been observed when they temporally moved away from the counter seat, but the precision score is still 92%. iLock's precision is lower in the coffee shop than in the restaurant because of the interference from the occasional high-frequency noise from the espresso machine, while iLock++ is less affected. However, both iLock and iLock++ do not perform well in tracking the departure activities in some specific layouts where the experimenters passed by a near obstacle (e.g., a pillar) on their departure trace.

Lounge. We used a spacious, quiet lounge, where the experimenter was sitting on a couch, and the phone was placed either on a coffee table, or a couch (to simulate the phone being dropped from the pocket). The couch was shared with a stranger, and occasionally, there were people passing by. iLock and iLock++ do not perform well in the lounge with low recall scores of 64% and 60%. Due to the combination of the environmental factors (signal partially blocked by the furniture), and the user's departure trace (walking in a direction where the signal transmission is weak), iLock and iLock++ can hardly capture the user's movement as they highly rely on distance estimation. In contrast, Chaperone detects 96% of the departure activities. We record six false positives (including two in actual departure activities but where Chaperone sent alerts before t_d) for situations where the user reclined on the couch while the smartphone was on the coffee table. Similar to the document fetching cases in the office scenarios, the body reclining movement pattern is similar to the departure pattern, which misled Chaperone. One false positive is recorded when the smartphone was left on the couch and the user stood up from the couch, where a significant moving-away event was captured by Chaperone.

Bus stop. We experimented at two types of bus stops: an open-air bench and a small glass-enclosed waiting area. The experimenter left the phone either on the bench or a seat in the waiting area. There was high-frequency noise from passing vehicles and alert signals from trams. Several other people were also waiting for a bus or passing by. In this scenario, Chaperone significantly outperforms iLock and iLock++, detecting all the departure activities (recall: 100%). We note four false positives for Chaperone (the precision is still 92%) when the phone was placed between a stranger and the user, where the stranger-phone dis-

tance was very close to the user-phone distance. When the stranger moved away, Chaperone tracked their movement and resulted in a false positive. iLock and iLock++ were prone to be misled by the stranger's movement, especially when the user's movement range was intersected by the stranger's. In addition, the high-frequency noise sometimes interfered with the detection of iLock and iLock++ and produced false positives. Chaperone was unaffected by such high-frequency noise thanks to its noise handling strategy. Results from this scenario strongly suggest that Chaperone can operate reliably in such noisy environments.

In-vehicle. Since a significant number of smartphone losses happen during ride hailing [170], we specifically target this scenario, which includes several challenges: the car space is much smaller than other scenarios, and the leaving procedure is very short—the user opens the car door, steps out, and closes the door. Also, when exiting the car, friction noise is produced by clothes and the seat, as well as the clunking noise from the car door. We simulated the cases where the user leaves the phone on either the front or back seat in a sedan with different noise conditions for the state of the engine, radio, and air-conditioner. Chaperone has no false positives, and the recall reached 91%, outperforming iLock and iLock++. The false positives for iLock and iLock++ were the result of noise in the narrow car space when the user was stationary. However, the common noise in the car did not affect Chaperone's user tracking (due to the incorporation of noise detection, candidate selection algorithms and three user state classifiers). The false negatives for Chaperone primarily came from the short leaving procedure, and the movement of the car door when the user was closing it. To reduce false negatives, one possible solution is to shorten the decision window when the phone detects that it is in a vehicle. Nevertheless, Chaperone provides overall good performance for the car scenario.

Academic venue. We collected data at a workshop (a lecture room with over 50 people), and a conference keynote (a large hall with over 900 people). We tested the keynote scenario at the end of the talk when the conference participants were leaving from the hall (crowded and very noisy). Due to the limited data collection time, we only collected three departing activities for each place. Chaperone worked well without any false negatives. One common false positive for iLock and iLock++ in the keynote hall is that they lost track of the user because the echo strength dropped quickly to the same level as the noise before the user was reaching the 1m line.

Summary. iLock resulted in more false positives for real-world conditions than in lab experiments. Using Chaperone's candidate selection strategy, iLock++ offers higher precision, especially in restaurant and coffee shop scenarios where environmental factors introduced more noise; iLock++ also improved in detecting more departure activities in office and in-vehicle scenarios. Chaperone outperforms both iLock and iLock++ in complicated scenarios like the lounge and bus stop. The decision making module determines the user's

departure activities based on user's motion state and activity intensity, rather than estimating user-phone distance only. In general, Chaperone consistently performs well in terms of recall rate. However, among the eight locations, Chaperone has lower precision scores in the office and lounge scenarios, apparently, because users have a large movement range in these scenarios, and some specific activities (e.g., document fetching) are similar to the preset departure pattern of Chaperone. A possible solution is to enable different departure patterns for different types of locations.

3.5.2 Evaluation under Longer Idle Periods

The experiments in §3.5.1 evaluated false positives during everyday activities of short duration. In some scenarios, acoustic sensing may be triggered for a longer period of time while the phone is idle on a table with the user around, e.g., in a meeting room at an untrusted place. In this case, a false positive can be quite annoying. Therefore, we evaluated false positives with Chaperone running for 15–20 minutes in the following scenarios: office, library, restaurant, and meeting room. We configured Chaperone to continue to run after detecting a departure event. The experimenter performed everyday activities matching the scenarios. We observed no false positives in the office, library, and restaurant scenarios, while two false positives occurred in the meeting room scenario. Both false positives happened when the user was stationary, and the closest colleague, sitting around 30cm away, did some movements (e.g., adjusted their seat), misleading the detection. Overall, the false positive rate of Chaperone is acceptable for longer acoustic sensing sessions under different situations.

3.5.3 Effects of Other Interference Factors

As Chaperone relies on acoustic sensing, it may be affected by the following scenarios:

1. The sound transmission is partially blocked by an object very close to the speaker and microphone of a smartphone.
2. Multiple nearby Chaperone-enabled smartphones are conducting acoustic sensing concurrently.

We evaluate these cases by running Chaperone in the standalone mode with the trained classifier models used in the real-world experiments. In each experiment, the smartphone(s) are placed in front of the experimenter with 0 degree orientation angle (i.e., vertical) and the experimenter moves at a normal speed. For each setting, we conduct 15 experiments.

Close-object Experiments

For the real-world experiments, we did not control the environment, including the presence/absence of nearby objects. We further perform controlled experiments to study the effect of nearby objects that partially block transmission. The Pixel phone that we use in these experiments, utilizes the bottom speaker and microphone for acoustic sensing. (We discuss smartphones with different hardware layouts in § 3.7.) Intuitively, if an object that is wider and thicker than the smartphone is placed very close to the bottom of the smartphone, the sound transmission will at least be partially blocked.

Although many factors, such as object numbers, surface materials, and placements, may affect the sound transmission, our main focus is to test Chaperone under different blocking effects. Therefore, we change the distance between the object and the phone to study the blocking effects. We conducted the close-object experiments in an office, and placed the Pixel (8.5mm thick) on a desk with a laptop on its left side and two books on its right (within 50cm to the phone). We investigated the effect of a single object in front of the bottom speaker and microphone. We used two objects—a 200-page notebook (landscape-oriented, height: 19mm, width: 266mm) and a 16-oz steel coffee mug (height: 198mm, width: 84mm), and phone-object distances of 5cm and 15cm. Besides, we tested the situation where the notebook was stacked on top of the Pixel (placed at the notebook's centre) with an 8mm gap between the desk and the notebook. When the notebook was placed 5cm away from the phone, the departure of the experimenter was detected in 13/15 cases; for the coffee mug at the same distance, 11/15 cases were detected. Since the coffee mug has a larger surface than the notebook to reflect the signal, it becomes more difficult to track the user's movement. However, when the mug was placed 15cm away, 14/15 cases were detected. When the phone was covered by the notebook, Chaperone detected 12/15 cases. Overall, Chaperone can still function when signal transmission is partially blocked.

Concurrent Sensing Experiments

Another situation of interest is when multiple Chaperone-enabled devices conduct acoustic sensing with the same acoustic signal at the same time. Intuitively, the interference caused by the direct transmission (from the speaker of a phone to the microphone of the other) can be offset by the differential magnitude, since both acoustic signals have identical period, and the overlying signals are constant in each sensing period. Our pilot tests show that the acoustic signal generated by another phone, together with its echoes, could also be detected as additional noisy frames by Chaperone, which is not considered in our design. One solution is to adopt a higher noise threshold when Chaperone detects another identical

acoustic signal close-by. In the experiments, we tuned the threshold and made no other changes in the noise detection sub-module to handle concurrent sensing.

We follow the basic setting used in the nearby stranger experiments: two Chaperone-enabled phones (the Pixel and Pixel 3, using the same classifiers) are placed in parallel on the desk and two experimenters stand in front of the two phones and leave alternatively. Each experimenter repeats the departure activity for 15 times. The shoulder-to-shoulder distance between the two users was 30cm, while the distance between the two phones was 75cm. No false positives were detected, while one false negative on the Pixel 3 and three false negatives on the Pixel were observed. To simulate the case where two Chaperone users are very close to each other, we reduced the distance between the phones and the distance between the users by 10cm: to 65cm and 20cm, respectively. Both phones detected 11/15 cases without any false positives. In comparison, when only the Pixel was conducting acoustic sensing, and the other conditions remained the same, we still observed five false negatives. Since the two users shared a largely overlapping activity range, it became more difficult to distinguish them. Nevertheless, there is no significant change in detection performance brought by concurrent sensing.

We added another Pixel (and an experimenter) to conduct concurrent sensing experiments with three devices by placing them in parallel, 75cm apart, with the Pixel 3 in the middle. For the 45 experiments, there was one false negative on each Pixel and three false negatives on the Pixel 3, with no false positives on any device. These results indicate that Chaperone can function concurrently on multiple devices with limited performance penalty. Similar to the close-object experiments, we cannot exhaust all possible settings and related factors, such as more smartphones (and users) and different placements. However, our experimental results have shown the feasibility of Chaperone in common concurrent sensing situations.

3.6 User Study

Our real-world device loss experiments show promising results for Chaperone. To validate the subjective nature of some of the results (e.g., the acceptability of the detection delay to the users) and to understand users' concerns for the adoption of Chaperone, we conducted an IRB approved user study.

3.6.1 Objectives and Methodology

We divide our objectives for the user study into three main themes: investigating device loss experiences and users' reactions, acceptability of Chaperone, and effective alert mechanisms for device loss. For device loss experiences, we collected data about the occurrence, location, reaction, and the final outcome of the event. For the acceptability of Chaperone, we collected data on detection ability, detection accuracy, power consumption, and overall effectiveness. We also collected data on what participants liked or disliked about Chaperone and whether they would use Chaperone on their devices. Finally, we asked participants regarding their preferred alert mechanisms for different environments based on perceived effectiveness and annoyance.

To achieve these objectives we conducted a three-part study: a semi-structured interview on smartphone loss experiences, a hands-on experience of Chaperone, and a semi-structured interview for their feedback on Chaperone. While a longer field study may have provided better insights, the nature of smartphone loss events cannot be controlled in a field study.

We recruited participants from the campus (excluding our research lab) and local community through word-of-mouth. We did not require participants to have experienced smartphone loss. For a realistic evaluation of Chaperone, the user study was held in a busy campus cafeteria during weekdays. At the cafeteria, participants responded to a brief demographic survey and the smartphone loss experiences interview.

For the hands-on experience, participants were asked to test Chaperone with real-time distance-tracking display on both the Pixel and the Pixel 3. They could test Chaperone freely and/or under the guidance of the investigator. At this stage, Chaperone alerted the user only through a pop-up message when it detected a potential smartphone loss. Then, we enabled a ringtone-and-vibration based alert without telling the participants about it, asked the participants to simulate a smartphone loss scenario, and observed their reaction to the alert. We chose the Pixel's "Nudge" as the alarm sound with alarm volume at 100%. We then demonstrated participants different alert methods including a strong ringtone (i.e., Pixel's "Classic Bell"), screen flashing, and notification sound to get their feedback on their preferences for each method for different locations. Finally we conducted the semi-structured interview to get their feedback on the acceptability of Chaperone and their preference for alert methods. We provide detailed interview questions in our project link.

3.6.2 Findings from the User Study

We have 17 participants (7 females, 10 males) in the study. 13 participants are 18–25 years old, and the rest are 26–30 years old; 15 are with Computer Science or IT background.

Smartphone loss and unattended experiences. In the first semi-structured interview, 11/17 participants reported having experienced smartphone loss. Four participants reported more than one loss. Among the 15 reported loss cases, two were due to pickpocketing (beyond Chaperone's threat model), ten were due to participants forgetting their phones, and three were due to phones slipping out of participants' pockets. Besides, 7/17 participants reported that they had unintentionally left their phones unattended to do something quick (e.g., go to a washroom) in public places. None of these unattended phone cases resulted in device theft or unauthorized access. Thus, we focus on the 15 smartphone loss cases.

The reported locations of the loss incidents include: library (four cases), street (three), washroom (two), in-vehicle (two) and one each for bus stop, meeting room, semi-open dormitory area, and gym. The participants realized the absence of the phone within 30 minutes for five cases (including two pickpockets), and more than one hour for eight cases. In two cases, the participants realized only when someone found the phone and returned it. Except two pickpockets and a forgotten phone in the semi-open dormitory area (later stolen by someone), the participants eventually recovered their phones.

In nine cases, participants went back to all possible places to look for their phone, which reportedly took them another one hour (four cases), or more than two hours (three cases) to recover their phone. Three participants used "Find My Device" services; one of them managed to recover the lost phone, while the other two failed.

This short survey indicates the importance of a preventive approach: finding lost phones is time-consuming, and in some cases, such phones may never be found.

Feedback on Chaperone. For the device loss simulation, the researcher noted that 11/17 participants reacted (e.g., stopped leaving, turned/moved back) to the ringtone-and-vibration based alarm and another five participants mentioned that they had heard the alarm but they thought it was from somebody else's phone. (Recall that tests were done in a busy cafeteria.) In practice, Chaperone users would be aware of their alert sound so this confusion would unlikely happen; we did not inform participants about the type of alert to prevent them from explicitly waiting for it and biasing the results. 15 participants heard only the ringtone, but not the vibration; only one participant reported hearing the vibration. As for the inaudible acoustic sensing signal, all participants reported not noticing it during the whole demonstration.

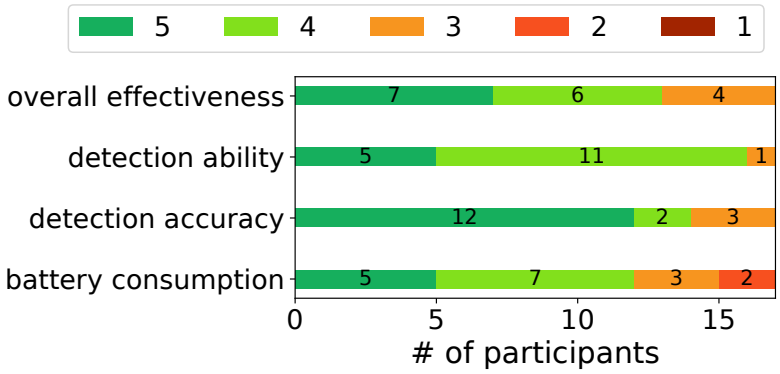


Figure 3.12: Participants' rating of Chaperone on a 5-point Likert scale (5: Very satisfied, 1: Not satisfied at all)

During the interview, when asked what participants liked about Chaperone, all reported liking the idea of alerting a smartphone user when leaving the phone behind to prevent smartphone loss. In terms of dislikes, nine participants suggested the ringtone used in the hands-on experience should be more noticeable. Five thought the real-time distance tracking was not very accurate because they noticed small fluctuations in the real-time trace display although they did not lead to any false positive or false negative.

To measure Chaperone's acceptability, participants rated Chaperone on a 5-point Likert scale, as shown in Figure 3.12 (a higher score means a higher satisfaction), for its overall effectiveness (*Assuming that you want to use a device loss prevention solution, do you think Chaperone is an effective system?*), detection ability (*How do you rate the Chaperone's ability to capture a smartphone loss?*), and detection accuracy (*How do you rate the Chaperone's detection accuracy? (a counterexample is that Chaperone sends an unwanted alert when the owner is not actually leaving*). The average effectiveness score is 4.2, the average detection ability score is 4.2, and the average detection accuracy score is 4.5. The results show that the participants were satisfied with the performance of Chaperone.

For the power consumption, we first shared with participants the battery consumption rate of Chaperone conducting detection reported in §3.4.3, and then explained that it is only triggered when all conditions (see §3.2.1) in the trigger module are satisfied; i.e., the real power consumption will depend on smartphone usage habits. Therefore, we asked the participants to rate the impact of Chaperone's power consumption based on their habits from 1 (i.e., significant) to 5 (i.e., negligible). The average score is 3.88, implying that the power consumption is acceptable for most participants. Participants mentioned that they

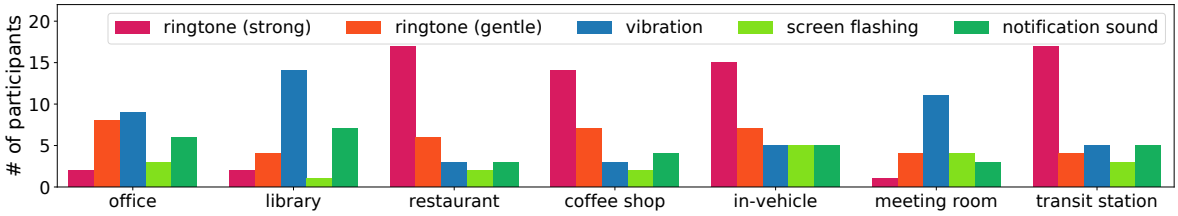


Figure 3.13: Participants' preferences of alert methods for different locations.

usually do not spend a long time in untrusted or public places, and therefore, the extra power consumption by Chaperone is still acceptable considering the potential benefits. Two participants rated the power consumption impact as 2. Their reported reason was that they are heavy smartphone users and their smartphones can hardly accommodate any additional battery consumption.

Alert. We also asked participants to comment on the alert they received during the hands-on experience. Among 16 participants who perceived it, twelve thought the timing of the alert was good to attract their attention, three thought the alert was a little late and they might miss it if the alarm sound was not loud enough in a noisy environment, and one participant thought Chaperone sent the alarm a little early and suggested to allow adjustable sensitiveness for the alert.

For participants' rating of different alerts, 13 participants rated the effectiveness of a strong ringtone as "Very effective" or "Effective," while eleven participants thought vibration was "Not effective at all" since the vibration was too weak to alert the user in a noisy environment. As for screen flashing, ten participants rated it as "Not effective at all" since the phone is usually behind a leaving user. Participants were also asked to choose their preferred alert methods for seven location types based on their perceived effectiveness and annoyance. They were allowed to choose none or multiple alert methods for each location. Most participants chose noticeable alert methods like strong ringtones for noisy places, while for the quiet places gentle ringtones and vibrations were preferred; see Figure 3.13. Five participants chose screen flashing for the in-vehicle scenario as a complementary alert method since it can make the phone noticeable in a dark environment. The results suggest that the trigger module can help Chaperone to determine an appropriate alert method based on the current context. Ten participants mentioned they needed a customized ringtone for device loss and nine participants expected the volume of the ringtone to be automatically adjusted based on the ambient noise level. Three participants requested further actions like e-mail notifications if the user failed to respond to the alert within a pre-specified time. These comments and suggestions are useful in designing

a context-aware alert mechanism for Chaperone.

Adoption. We asked the participants: *Would you like to install Chaperone as a device loss prevention app on your phone? (Yes/No/Maybe).* 8/17 participants answered "yes" because they thought Chaperone helped reduce the risk of smartphone loss. Eight participants answered "maybe"; four of them believed they had a good habit of always keeping their phones with them but they still wanted to try it to record how often they leave their phones behind, two had privacy concerns due to Chaperone requesting the microphone permission, one was worried about the effectiveness of the alarm in very noisy environments, and the other one expected that Chaperone could learn from a user's habits to trigger the sensing smartly and save battery. Only one participant answered "no" as they did not need a device loss prevention application due to the perceived low probability of losing the device.

Threats to validity. Our user study has some reasonable limitations similar to other studies involving human subjects including the limitation of scope to people willing to participate, self reported and subjective views, and participants might be inclined to provide favorable responses to the researchers. More specific limitations follow. Most of our participants are current undergraduate or graduate students in Computer Science, lacking diversity in participants' background. Although we did not require participants to have smartphone loss experiences, the advertised content for the user study mentions the study is to "test the context-aware techniques on smartphones to prevent smartphone loss", which may have attracted users with such experiences. Another threat to validity is that the first interview about smartphone loss experiences may have primed participants for adoption of Chaperone. During the user study, we used a Pixel and a Pixel 3 as the demonstration phones. The participants reported their perceptions of different alert methods based on their experience of using these two devices. These results may not be fully applicable to other devices due to hardware differences (e.g., max volume difference, vibrator difference). In addition, since our user study focuses on collecting smartphone users' perception about Chaperone and its alert mechanism based on one demo session, it may not cover potential issues regarding long-term usage of a product-ready Chaperone.

3.7 Discussion

We discuss a few issues relevant to the deployment and usage of Chaperone, including limitations of our current prototype.

Very close attackers. In §3.1.1, we assume that the attacker is initially farther away

from the phone than the owner. If an attacker who is initially closer to the device than the owner, the system may track the wrong person since it assumes that the initially closest person is the owner. To defend against such an attacker, Li et al. [108] adopt a dedicated approach that requires two microphones and inertial measurement sensors to distinguish the attacker from the owner. However, it provides acceptable accuracy only when the owner and the attacker are facing each other, i.e., not side-by-side. Their approach also requires the owner following a straight path away from the phone with a consistent relative user-phone orientation (unlike Chaperone). Finally, both microphones may not always be available at the same time, since the top or rear microphone could be covered when the phone is lying on a surface. For Chaperone, a potential defense against such an attacker is to trigger sensing right after the user puts down the phone on a surface to track the user's hand movement immediately, assuming the user's hand is the closest moving object at that moment.

Active attackers. As mentioned in the threat model (§3.1.1), Chaperone targets nearby opportunistic attackers, not Chaperone-aware active attackers. An active attacker may attempt to disarm Chaperone so that the auto-lock and alert mechanisms are not triggered. We briefly discuss two types of active attacks here. 1) Jamming attack: If an attacker continuously generates loud noise over the inaudible high-frequency band used by Chaperone, the echo of Chaperone's own acoustic signal will be lost. However, it is possible to measure the ambient noise to detect such an attack and alert the user before conducting acoustic sensing. 2) Misleading attack: A nearby attacker makes significant movements to produce strong reflected signals so that Chaperone tracks the attacker instead of the user. When the owner is leaving, the signal reflected by the owner becomes weaker and the nearby attacker's movements overlap the owner's departure trace. Note that, the attacker must be very close to the target smartphone (i.e., within one meter) and make significant movements before the owner moves too far away. For a better defence against the misleading attack, a potential avenue is to improve the motion tracking algorithm (e.g., include motion history), or use additional detection methods (e.g., RF sensing [175]) to distinguish different people.

False positives and negatives. We noticed some common false positives caused by (relatively) longer range user movements (e.g., reclining on the couch) in the lounge scenarios, and false negatives caused by moving objects (e.g., the car door) in the in-vehicle cases. Since our models were trained with data from the lab-based experiments, it was challenging to handle these special cases. An apparent solution is to train the models with more real-world data covering these situations. We use only lab data for our models to measure Chaperone's robustness in newly encountered situations—which we assume Chaperone to face frequently in practice.

Smartphone hardware differences. We focus on the environmental factors that affect acoustic sensing. Most experiments were conducted on a Pixel, while a few concurrent sensing experiments were done with a Pixel 3. To support different devices, we explain how to transfer the classification model in § 3.2.5. A systematic study on how hardware differences affect Chaperone's sensing ability and the necessary parameter adjustments due to these differences is future work. Two possible areas are the following: 1) Sensing ability: given differences in microphones and speakers, the recording quality above 19kHz may vary on different smartphones, directly affecting the magnitude of the received signal. 2) Hardware layout: the positions of speakers and microphones may differ for different smartphones. Even for the bottom microphones, some may be placed on the bottom edge (e.g., Pixel) while some may be on the bottom front (e.g., Pixel 3). Such differences may result in different sensing ranges because of the directionality of microphones.

Measurement inaccuracies. During our data collection, a human observer marked the reference points for the moment t_a when the user passes the 1m line. A standalone distance sensor may have provided a more accurate labeling. However, setting up such a sensor in public places, like restaurants and coffee shops, is inconvenient, and therefore, we settled for labeling by a human observer.

3.8 Conclusion

This chapter mainly addressed Objective 1 by presenting Chaperone as a standalone, open-source Android app that uses acoustic sensing to detect smartphone loss in real-time. From the perspective of adaptive authentication, Chaperone provided a context sensing technique for detecting a user's departure and absence, which implies an unattended device. It determines when to lock the device with mandatory authentication. Our real-world experiments showed that Chaperone could operate reliably in diverse real-world scenarios characterized by high ambient noise, crowded locations, and diverse physical layouts, *without* retraining our classifiers for specific scenarios. Our user study provided positive evidence that Chaperone can indeed be made into a practical tool to help prevent device loss, and thereby reduce serious privacy and security threats caused by lost smartphones. Beyond device loss, Chaperone's design and our extensive real-world datasets will help advance acoustic sensing research.

Chapter 4

Device Sharing Awareness

4.1 Introduction

This chapter investigates temporary device sharing from the second illustrative example in § 1.1.3 and addresses Objective 2 by proposing device sharing awareness (DSA). During temporary device sharing, a non-owner user, i.e., sharee, is temporarily allowed to access non-sensitive resources on the device. While lockscreens are employed to secure sensitive data with mandatory EA, they offer no protection when the owner temporarily shares their unlocked device. The single-user design of most mobile devices and apps may expose sensitive resources to sharees [71]. For example, an e-mail app may keep the owner's account logged in and pop up notifications new e-mails during sharing. Thus, authentication and access control for mobile devices should be adjusted to support temporary device sharing. The system should prevent unauthorized access to sensitive resources (e.g., via hiding or requiring EA) but not block a sharee from using the resources to be shared. Besides, changes in authentication and access control should take effect only during sharing and be automatically revoked once the device is returned to the owner.

4.1.1 Limitations of Existing Sharing Control Solutions

In § 2.2, we review the existing device sharing control solutions. Most of them emphasize *how to* impose access restrictions on sensitive apps and data during sharing. They allow the owner to add a guest account [79], pin an app [77, 78], or launch apps with limited features (e.g., a camera app without a view of existing pictures) when the device is locked.

However, most of these solutions require an explicit action from the owner before sharing the device. Vulnerabilities arise when humans are forgetful [152] or lack risk perception of certain situations [7]. Owners may forget to switch to the guest account or to pin an app before sharing. Furthermore, sharing behavior is closely related to trust [155]. An owner explicitly enabling these solutions signals mistrust for device sharing between the owner and sharee [4, 71, 90, 123, 125]. Besides, these solutions are inadequate for some sharing scenarios. A guest account works well to entertain children with a game but not for temporary sharing with spouses. Pinning an app grants access only to the current foreground app. It is insufficient when a sharee needs access to multiple shareable apps.

In summary, we list the following limitations of existing sharing solutions:

1. **Lack of subtlety.** Ahmed et al. [4] have found that the act of locking or pinning an app or data may incur social challenges and raise suspicion, especially when it comes to device sharing with family members. Thus, a device sharing solution should be activated subtly and automatically by the device.

2. **Relying on a user's explicit input.** Many device sharing solutions require a user to manually trigger them. A user's forgetfulness or lack of risk perceptions [7] can cause inaction to device sharing, resulting in the exposure of sensitive data. Besides, relying on a user's input can also result in poor usability since an owner may need to take additional steps (e.g., enter a PIN for app locks) to access certain resources during regular phone use.

3. **Coarse-grained access control.** Many solutions follow a simple access control model to grant all or nothing access to each app. However, it is preferable to give apps the option to adapt their own fine-grained access control strategies during sharing. For example, a browser app provides the essential web browsing function and may store the owner's passwords for auto-filling. In this case, the browser app should allow a sharee to browse the web without having access to the owner's data.

4.1.2 Design Goals

We introduce device sharing awareness (DSA) to address *when to* enable device sharing solutions. A device sharing awareness solution should:

1. *proactively detect device sharing* instead of requiring an owner to remember performing a predefined action to enable a control,

2. *continuously identify the owner* to prevent unauthorized access and ensure that only the owner has full access to the device,

3. *be exception-resistant* to automatically handle possible false detection or exceptions and mitigate the exposure of sensitive resources.

For an outcome of device sharing awareness, the solution should involve the change in the behavior of the authentication system (e.g., an IA mechanism should not block the sharee) and provide *flexible access control* to automatically determine what resources are available to the sharee. We present the following example to illustrate how DSA is expected to handle device sharing automatically:

In a coffee shop, Owen shares with his friend, Shannon, a bunch of travel photos stored on his smartphone. When he hands over the phone to Shannon, DSA automatically detects the sharing activity and notifies the gallery app so that the app can hide all photos labeled as private. At the same time, all notifications from messaging apps are silenced. During sharing, DSA allows Shannon to be redirected to the Map app by the location metadata of photos but not to move to any social networking apps to post photos. After Shannon finishes browsing the photos and returns the device, the system recovers as before the sharing activity.

4.1.3 Contributions

To fulfill the above requirements, we present a DSA solution that automatically deals with all aspects of a device sharing event with little to no input from the device owner. For subtle and fast sharing detection, DSA continuously senses for device handover gestures using motion sensors and verify the owner's identity using behavioral biometrics with high accuracy and low power consumption. Behavioral biometrics alone may make it hard to distinguish a sharing event from unauthorized access and rapidly react to the sharing event. When detecting a sharing event (or upon manual activation by an owner), DSA can enable app-level access control using allowlisting or blocklisting. Besides, it allows the shared app to adopt its sharing-specific access control strategies (if available). Other apps are also notified of the device sharing by DSA and can adopt their own sharing reaction (e.g., de-authenticating a user).

We conducted a user study and collected data from 18 participants to evaluate DSA. Our evaluation over 3,700 motion data clips shows that DSA can detect handover gestures accurately for 95% of the sharing events. On a public dataset containing 81-hour phone

usage data from 100 users [159], DSA only generated 0.9 false positives per hour of continuous device use. For an average daily smartphone use of about three hours [61], DSA will generate about three false positives a day. We also tested the device sharing processing ability of our DSA implementation with a popular touch-based IA mechanism [52], which includes 50 complete device sharing sessions. DSA succeeded in detecting handover gestures in 48 sessions, and automatically handled 42 sessions without exceptions while its exception processing recovered six out of the eight sessions. DSA adopts an adaptive sensing strategy and only consumes 0.11% of battery per half hour at high-frequency sensing when there is significant movement, and consumes only 0.06% of battery per half hour at low-frequency sensing.

The contributions are fourfold:

1. A demonstration of the ability of low-cost proactive sharing detection using smartphone built-in sensors.
2. An open-source solution for Android that automatically enables sharing protection during device sharing, while mitigating human factors of forgetfulness and mistrust.
3. An extensive evaluation of DSA to demonstrate its practicality in terms of accuracy to detect sharing and battery consumption.
4. A public, labeled motion sensor dataset with over 3,700 sharing events for the research community.

4.2 Device Sharing Awareness

4.2.1 Modeling Temporary Device Sharing

The device sharing scenario targeted by our work is: The device is initially with its owner, and the owner temporarily hands it over to a sharee as a signal of granting access. During sharing, a sharee should not have access to sensitive resources, including personal data (e.g., messages, photos) and critical operations (e.g., delete files). We do not study device sharing where the device is initially not with its owner, or where a sharee can access the device without the owner's presence. Traditionally, this kind of sharing is enabled with separate user accounts or PIN sharing [4, 123]. We discuss this case in § 4.6.

We describe a sharing event with the following three-stage device sharing model:

1. **Pre-sharing.** The owner initially holds the phone. The owner unlocks the device and opens the app that contains the resources to be shared. Then, the owner passes the device to the sharee.

2. **Sharing.** The sharee holds the device and starts using the opened app. During sharing, the sharee should be able to access only the specified resources for sharing. For the multi-sharee scenario, sharees may pass around the device, but we still regard it as a single sharing event.

3. **Returning.** The (last) sharee finishes using the device and returns it to the owner. A sharing event ends only when the current user is confirmed to be the owner.

We define the *shared app* as the foreground app at the moment when sharing is initiated. Based on the owner's preferences, a sharee may be allowed to access further apps during sharing. The term *shared app* always refers to the original one.

4.2.2 Sharing Detection

For minimizing the restrictions on an owner, a device sharing solution is supposed to take effect only when there is an ongoing sharing event. Therefore, an important requirement of device sharing awareness is to *proactively* determine the beginning and the end of a device sharing event. According to the device sharing model, we emphasize two factors for sharing detection: *sharing gestures* and *owner detection*. A sharing gesture is an indicator of a sharing event and implies that the owner authorizes the sharee to access the phone. We regard manual activation methods adopted by existing sharing solutions as *explicit* sharing gestures (e.g., buttons, touchscreen swipe gestures, and shortcut keys [77,78,141]). They explicitly indicate the beginning of a sharing event and trigger sharing solutions immediately. However, for subtlety and less reliance on explicit input, we also exploit an *implicit* sharing gesture, *the device handover gesture*, which can be directly sensed from the natural hand movements when the device is handed from one person to another.

While a handover gesture indicates the beginning of a sharing event, we cannot use it to determine the end of a sharing event since there may be multiple sharees passing around the device. Therefore, verifying the user's identity is also essential for sharing detection: While a non-owner user is temporarily allowed to access the device during sharing, the device should ensure that the current user changes back to the owner at the end of a sharing event. A common practice for de-activating the sharing mode is to ask for explicit authentication (e.g., a PIN) to ensure the device has been returned to the owner. For

DSA, device sharing solutions should be able to determine if the current user is the owner proactively and transparently. It can be achieved by IA techniques: A mobile device can distinguish the device owner from other people based on biometrics, including continuous facial recognition [47, 118], voice recognition [189], or behavioral biometrics [88, 92]. In addition to determine the end of device sharing, owner detection can complement a device sharing solution in cases where a handover gesture is detected erroneously and can avoid false activation of the sharing mode.

Detecting sharing events based on owner detection alone, ignoring handover gestures, is insufficient. It is hard to distinguish a sharing event from unauthorized access of a stranger (e.g., a stranger using an unattended, unlocked phone without permission) since a non-owner can be detected in both cases. Besides, continuous facial recognition may cause significant power consumption; voice recognition and behavioral biometrics require sufficient input data for identification, making the device slow to react to a sharing event. Thus, a crucial problem is how to combine handover detection and owner detection for detecting sharing events.

4.2.3 App Types

Most sharing solutions impose access control on sharees to avoid access to sensitive resources. We name this restricted environment for device sharing as the *sharing mode*. Many apps contain both shareable and non-shareable content, while some apps may involve redirection to other apps to process specific requests. Thus, existing solutions that only restrict the sharee to the current foreground app cannot fulfill these requirements. Based on whether resources in an app are shareable and existing taxonomies [71, 113], we classify apps into the following three categories:

- **Shareable apps.** Apps that are completely shareable without any sensitive resources, such as games or weather apps. A sharee has full access to such apps.
- **Semi-shareable apps.** Apps that contain both shareable and non-shareable resources, such as social networking or photo gallery apps. A sharee can access the shareable resources during sharing without access to personal data and sensitive operations in such apps.
- **Non-shareable apps.** Apps that contain no shareable content, such as system settings, banking, or corporate apps. During sharing, a sharee should have no access to such apps. Specifically, corporate apps have higher security requirements and need

to react to the sharing event even when running in the background (e.g., terminate the session, disconnect from a remote service).

Our goal is to design a device sharing access control strategy that meets the requirements of different kinds of apps. Moreover, we need to consider some special apps or components such as the home screen and the notification bar, most of which are provided by the system launcher in Android.

4.2.4 Threat Model

Temporary device sharing involves two kinds of roles: an owner and one or more sharees. We assume that owners are not malicious and focus on attacks from sharees. We classify sharees into two categories: A *benign sharee* only uses the specified resources without any intention of accessing sensitive information or other apps during sharing. However, a benign sharee can do accidental mis-operations that expose private data (e.g., switch to other apps). It is also possible that some apps may push notifications that contain sensitive information to a benign sharee (e.g., an email notification with a preview). A *malicious sharee* targets other apps than the shared app and intends to access private information during sharing. They may try to leave the current app and access unauthorized resources. A malicious sharee may be aware of the existence of the protection mechanisms, such as screen lock and IA, and attempts to bypass them. A malicious sharee may also know of the existence of our proposed solution and launches attacks accordingly.

4.3 Our Approach

We now introduce the design of our framework. We present how DSA works based on different states, its main modules, complete workflow, and exception handling strategies.

4.3.1 State Transition

We define three states of a device: *normal*, *sharing*, and *locked*. In state "normal", the user has full access to the device. In state "sharing", the user has limited access to the device and cannot access sensitive resources. In state "locked", the user has no access to the device and needs to explicitly authenticate. Figure 4.1 shows the state transition among the three states. Existing app pinning solutions fully rely on manual operations to switch

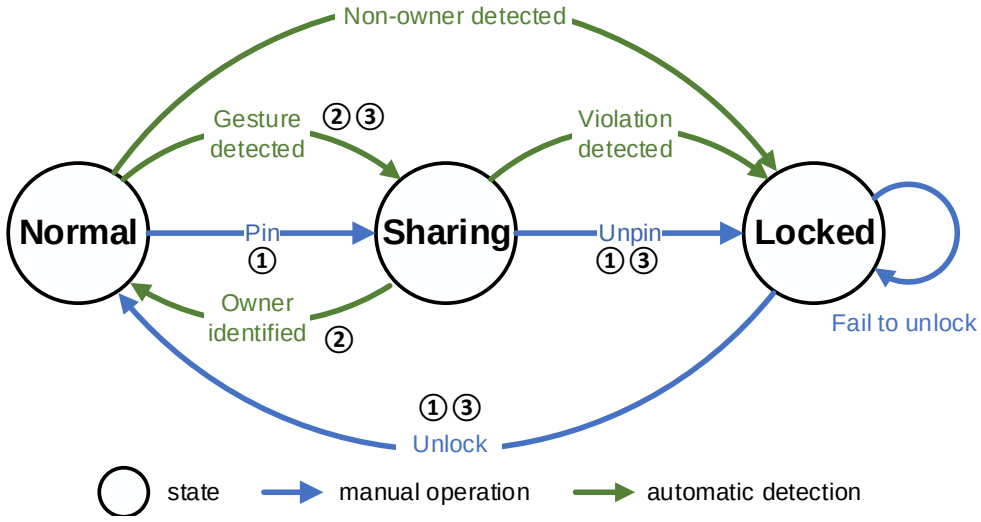


Figure 4.1: State transition of device sharing. Three sharing loops: ① explicit sharing loop (manual option), ② implicit sharing loop (handover gesture + owner detection), ③ hybrid sharing loop (handover gesture + manual unlock).

among the three states (see Figure 4.1 Loop ①): 1) pin an app to start sharing (i.e., limit access only to the current foreground app), 2) unpin an app to end sharing and lock the device, 3) authenticate the user to return to normal state. DSA keeps this loop to allow users to start or end the device sharing manually.

Following § 4.2.2, we introduce an implicit sharing loop (see Figure 4.1 Loop ②) as a new trigger mechanism:

1. Sharing: If DSA detects a handover gesture, the state changes from "normal" to "sharing".

2. Returning: If DSA confirms the owner's identity, the state changes back to "normal".

Note that detecting a handover gesture, which may occur when a sharee returns the device, cannot be used to end a sharing event given possible multi-sharee cases or gesture spoofing attacks (i.e., the sharee fakes a handover gesture). In the implicit sharing loop, DSA can handle device sharing and secure sensitive resources without locking the device or asking for manual actions by the owner. However, state "locked" is still useful for processing violations (see § 4.4.2). DSA allows a hybrid sharing loop (see Figure 4.1 Loop ③) where DSA detects a handover gesture to move into sharing state while the owner or sharee have

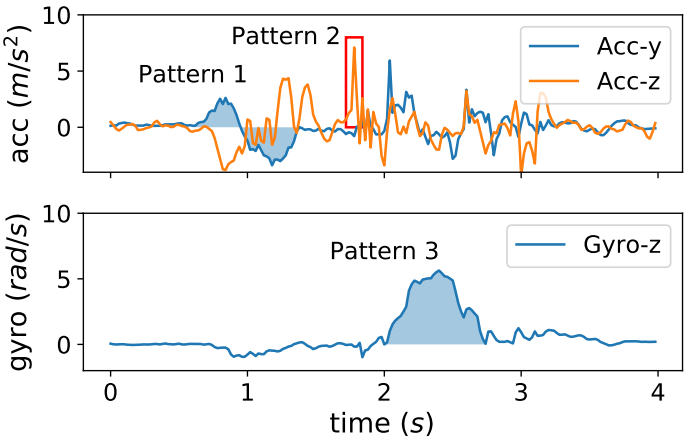


Figure 4.2: Handover patterns. 1. (horizontal movement): the device travels a distance in the xy-plane, where acceleration follows a sine curve like pattern; 2. (spike): When the sharee catches the device, a spike appears on the z-axis of acceleration; 3. (rotation): the device is rotated either by the owner or by the sharee to adjust the orientation.

to manually end sharing, and explicit authentication is required to move back into state "normal".

4.3.2 Handover Detection

We use the device handover gesture as a trigger of an implicit sharing loop. A handover gesture lasts only a few seconds and does not occur frequently. Compared to the typical gesture recognition problem, a handover gesture is performed in a natural manner rather than a specified motion (e.g., drawing a circle). The key to handover gesture detection is to study the common patterns of handover gestures and distinguish them from similar motions (e.g., switch hand).

Pilot experiments. We conducted a pilot experiment to investigate possible handover gesture patterns for feature selection: one experimenter, acting as a device owner, handed over a Google Pixel phone to another person (i.e., sharee) with two different position settings:

1. The owner handed over the phone from their right hand to the sharee sitting in front of them.

2. The owner handed over the phone from their right hand to the sharee sitting next to them.

Each setting was repeated ten times. We collected data from the accelerometer and gyroscope at a sampling rate (denoted as f_s) of 50Hz. We use a software linear acceleration sensor provided by Android, which isolates gravity from raw acceleration data with the help of the gyroscope. The collected data includes linear acceleration and rotation speed on the three axes.

According to the collected sensor data, we observed that the length of a handover gesture is about two to four seconds. We also observed three patterns and exemplify them in Figure 4.2. The observation shows the possibility to detect a handover gesture with motion sensors. It also helps us in determining features and targeting possible misleading activities that share similar patterns with handover gestures. For example, the acceleration readings of a horizontal hand movement follow a sine curve like pattern, which can be described by time-domain waveform features. A spike on the z-axis of acceleration resulting from a slight fluctuation of catching a device can be captured by entropy-based features. A misleading activity with similar patterns is a user's passing the device from their left hand to their right hand (i.e., switching hand).

Feature extraction. To proactively detect handover gestures, the device continuously collects motion data from the accelerometer and the gyroscope. We first divide the collected time series data into fixed-size segments, where the sampling rate is f_s, the segment length is d seconds (equal to $f_s \cdot d$ samples). Since feature extraction and classification are conducted on each data segment, the interval between two consecutive segments can be regarded as the period of gesture detection, which is denoted as t, where $t = d - p$. The choices of d and p affect the detection performance (further investigated in § 4.5). For example, if d is too small, it is hard to capture the handover patterns from a data segment; if d is too large and p is too small, then interval t becomes too large to detect a handover gesture and make the device react to it in time. Therefore, we use the following default settings: $d = 2s$ and $p = 1s$. We investigate the impact of different settings on detection performance in § 4.5.

After segmenting the raw data, we extract the following features for each segment: We first calculate the magnitude of linear acceleration:

$$m = \sqrt{a_x^2 + a_y^2 + a_z^2}. \tag{4.1}$$

For the magnitude, m, and each axis of linear acceleration and gyroscope data, we use common statistics widely adopted in gesture detection [6] and activity recognition [101] (see details in § 2.3.2) including: average, standard deviation, maximum, 25^{th} percentile,

median, 75$^{\text{th}}$ percentile, sum, and range of a data segment. Also, we measure root mean square (RMS) [49] of the readings to capture time-domain wavelet patterns:

$$\text{RMS}(\mathbf{v}) = \sqrt{(v_0^2 + v_1^2 + \cdots + v_{n-1}^2)/n}, \tag{4.2}$$

where $\mathbf{v} = \{v_0, v_1, ...v_{n-1}\}$ is a series of n sensor readings. We calculate value entropy and time entropy [171], which measures sudden changes in a signal. We calculate the value entropy by quantizing all magnitude values to a 20-bin histogram for a moderate granularity covering all magnitude ranges. For time entropy, we normalize the magnitudes of sensor readings to form a probability distribution and then calculates:

$$H(|\mathbf{v}|) = -\sum_{i=0}^{n} \frac{|v_i|}{\sum |\mathbf{v}|} \log \frac{|v_i|}{\sum |\mathbf{v}|}. \tag{4.3}$$

Furthermore, to include correlations between different axes, we calculate the correlation coefficient between every two axes as Eq. 2.2.

Classification. Based on the extracted features, DSA uses a pre-trained classifier to determine if the current segment belongs to a handover gesture. We adopt an offline learning strategy and train a generic classifier before deploying the system. For an online strategy, data labeling is challenging — Although a user's feedback can help correct a missed sharing event (see § 4.4.2), it is hard to position a sharing gesture exactly. Besides, our evaluation results in § 4.5 show the feasibility of applying a generic classifier to different users. To reduce false positives, we use a sliding window strategy that makes decisions based on several consecutive segments: if two consecutive segments are classified as positive, the system concludes that a sharing event is happening.

Adaptive sensing. Given that proactive handover detection is always running in the background, its power consumption is a concern for smartphone users. Therefore, we adopt an adaptive sensing strategy to reduce the battery consumption. The accelerometer and gyroscope initially collect raw motion data at a sampling rate at 10 Hz. When significant movement is detected (i.e., the acceleration magnitude exceeds a pre-defined *activation threshold*), it switches to a high sampling rate at 50 Hz and conducts handover detection. When the device is stationary for a period of time, the sampling rate is lowered to 10Hz. This strategy reduces unnecessary computations when the device is stationary.

4.3.3 Owner Detection

Owner detection is provided by IA mechanisms. DSA relies on IA results to determine if the current user is the owner or not. Biometric mismatch results in a negative IA result,

indicating that the current user is not the owner. In state "normal", IA mechanisms are running continuously to prevent unauthorized access from non-owner users. They will lock out the current user upon negative IA results. In state "sharing", IA mechanisms are automatically configured not to block users upon negative results as they indicate an ongoing sharing event. Once the IA results change from negative to positive at this state, DSA regards it as the end of a sharing event.

We incorporate existing IA mechanisms for owner detection and do not design a new IA mechanism. The selection of IA mechanisms should take accuracy, availability, detection latency and battery consumption into consideration. Ideally, IA mechanisms with low false rejection rate and low battery consumption are preferred in state "normal" since a device is not under sharing most of the time. In contrast, IA mechanisms with low false acceptance rate and short detection latency are preferred in state "sharing" to determine if the device has been properly returned to the owner. Owner detection can adopt multiple modalities to ensure accuracy and availability. For example, if touch-based IA produces a positive result, the device can automatically conduct face recognition to determine if the current user has changed back to the owner. It helps to ensure high accuracy with avoiding battery consumption of continuous facial recognition.

Considering the availability of various behavioral biometrics on smartphones, we use the touchscreen input biometric and adopt Touchalytics [52] (see § 2.4.1) whose reported equal error rate is below 4%, as the default scheme in our evaluation. It performs classification for each touch event and authenticates the user based on the results of several consecutive touch events. According to the evaluation done by Khan et al. [92], the battery consumption of touch-based IA is low enough to keep running in the background for continuous owner detection.

4.3.4 Access Control for Device Sharing

For improved usability, DSA adopts different strategies for enforcing app-level access control based on the shared app type. It also notifies apps of sharing status changes so that a shared app can change its behaviors to a shared mode. The common app-level access control strategies involve:

1. **Blocklist.** A device owner can determine a list of non-shareable apps that cannot be accessed by a sharee. In state "sharing", the system rejects all access attempts to the apps on the blocklist. Besides, hiding non-shareable apps is also applicable to block a sharee's access in a subtle way.

2. **Allowlist.** A sharee is only allowed to access a list of shareable or semi-shareable apps. App pinning methods can be regarded as a kind of allowlist that makes only the current foreground app available to a sharee.

3. **Profile switching.** Mobile operating systems (e.g., Android) organize user data in profiles and allow the programmatic switching of an app's profile [9]. In the context of device sharing, DSA switches to a guest profile that does not contain any owner's data. It enables a sharee to use a semi-shareable app without accessing the owner's data. If this feature is not available, an app can switch a profile as part of in-app sharing control (see below).

While an owner can configure access control strategies for different apps, DSA can infer what access control strategy to adopt: In most cases, DSA sets the current foreground app as a shared app and automatically adopts an allowlist-based strategy to restrict a sharee's access to the shared app and any shareable or semi-shareable apps redirected to from the shared app. If there is "no app" running in the foreground (e.g., the current foreground app is a launcher or home screen), DSA enables blocklist-based access control.

In addition to app-level access control, an app may have its own device sharing control strategies. Possible options include switching to guest mode, disabling user-specific content, logging out the owner's account, etc. For example, a camera app can provide only the camera function without revealing any local photos. As suggested by existing studies [4, 5], it is important for apps to incorporate the "shared use" paradigm into their current design to provide more fine-grained in-app sharing control. In this case, DSA can provide important device sharing notifications to these apps to help them decide whether to enable such a shared use design.

4.4 System Design

4.4.1 DSA Workflow

Figure 4.3 shows the architecture of DSA. It consists of four components:

- **DSA service** is responsible for the coordination of all other modules. It also accepts user inputs to switch between normal and sharing states manually.
- **Sharing gesture detector** continuously collects raw motion data, extracts features, and conducts classification for handover gesture detection introduced in § 4.3.2.

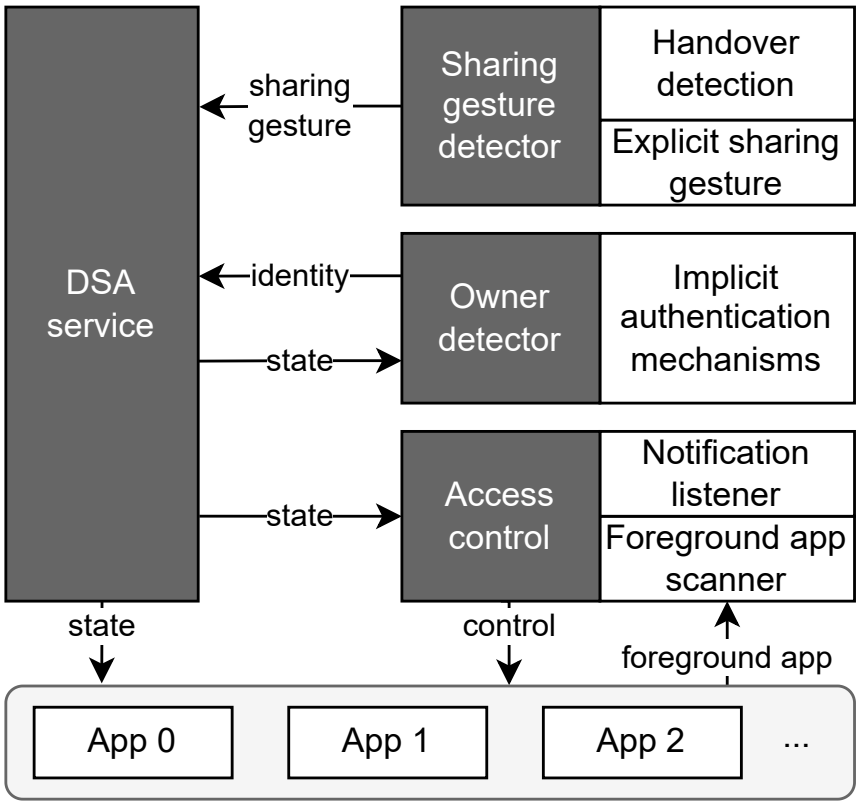


Figure 4.3: DSA architecture.

- **Owner detector** uses behavioral biometrics to determine if the current user is the owner. It also helps apps block non-owner users when there is no sharing event.
- **Access control** manages a sharee's access to apps and notifications during sharing. The foreground app scanner keeps tracking the current foreground app. During sharing, it detects when a sharee attempts to access non-shareable apps and then redirects the sharee to the shared app or locks them out. The notification listener intercepts notifications from all apps and blocks sensitive ones during sharing. The blocked notifications are temporarily stored until the device is returned to the owner.

The high-level workflow of DSA follows the three stages introduced in the device sharing model in § 4.2.1.

Pre-sharing. The sharing gesture detector and the owner detector run continuously. At state "normal", the owner detector is performing continuous authentication to reject non-

owner users. Once a handover gesture is detected or the owner explicitly starts the sharing mode, the DSA service updates the current state to "sharing" and adopts an access control strategy according to the current foreground app. It also broadcasts the device sharing signal to other apps so that they can enable their own sharing mode or other reactions such as requesting re-authentication for the next access.

Sharing. The device is in state "sharing" and the sharing mode is enabled. The foreground app scanner continuously checks if the current foreground app can be accessed by a sharee. It rejects any unauthorized access to sensitive resources based on the access control strategy by redirecting a sharee to the shared app given possible mis-operations. If the mis-operations reach a pre-defined threshold, the DSA service locks the device. The notification listener intercepts incoming notifications to filter out the ones from non-shareable and semi-shareable apps to prevent potential exposure during sharing. The blocked notifications are temporarily stored during sharing. The owner detector keeps verifying if the current user is the owner and stops blocking non-owner users (i.e., negative IA results).

Post-sharing. Once the current user is identified as the owner or the owner manually ends sharing and passes explicit authentication, the DSA service updates to state "normal". The DSA service notifies the foreground app scanner and the notification listener for lifting the access restrictions. The notification listener shows the owner all cached notifications that were missed during sharing. The owner detector resumes to defend against unauthorized access from non-owner users. The DSA service then broadcasts the state change to other apps so that they can revoke the changes made for device sharing.

4.4.2 Exception Processing

As the implicit sharing loop allows DSA to handle device sharing automatically without a user's explicit input, exceptions may occur, resulting from false detection, mis-operations, or attacks, and cause security or usability issues. It is critical to have an exception processing mechanism to recover from exceptions and mitigate their negative impact. Specifically, it needs to minimize the chance of sensitive resources exposed to a sharee. We classify exceptions into four types and provide solutions accordingly. In addition, in our user study (see § 4.5.3), we investigated the exceptions that DSA may encounter and how efficiently it handled these exceptions.

Non-owner detected in state "normal". When the owner detector detects a non-owner, it locks the device and asks for re-authentication, such as a PIN code or password. There are three situations:

1. The current user is an attacker, and the owner detector successfully prevents unauthorized access, which is not an exception of device sharing.

2. The current user is the owner, and the owner detector falsely rejects the owner, which is a failure of the adopted IA mechanisms.

3. The current user is the sharee, and the owner detector makes a correct detection, but the sharing gesture detector failed to capture the sharing event.

Therefore, we need to distinguish the second and third situations. If the user passes the re-authentication, the DSA service prompts a dialog to confirm if a sharing event was initiated. If so, it updates the sharing state and starts the sharing mode.

App exception. An app exception happens when a sharing event is detected but the current foreground app is invalid. It can be one of the following invalid apps: 1) a non-shareable app: DSA blocks the access and re-authenticates the owner. If the non-shareable app is logged in with the owner's account, the current session of the app will be immediately ended. 2) system launcher: it provides entry points to all apps on the smartphone. Since no app is specified for sharing, DSA applies a blocklist-based access control strategy. The sharee is prevented from accessing non-shareable apps, and the notifications of non-shareable apps are hidden.

False positives of the handover detector. If the handover detector falsely detects a handover gesture when there is no sharing event, the DSA service still moves to state "sharing", which causes inconvenience to owners. However, the owner detector can help correct false positives. If the owner continues using this app, the owner detector can identify the owner, and the system moves back to state "normal". Even if the owner detector also happens to make a false detection and mistakenly regards the owner as a sharee, the owner can still explicitly end the sharing mode and do a re-authentication. Another similar exception is that the handover detector makes a true positive while the owner detector falsely accepts the sharee. Since the owner detector keeps verifying the user's identity even after a temporary false acceptance, it may be still able to reject the non-owner afterward. Nevertheless, to avoid possible exposure, a possible solution is to require EA when the owner is detected immediately after the detection of a sharing event. In § 4.5.2, we evaluate how the owner detector addresses false positives of the handover detection.

App redirection. A shared app may involve resources that redirect to other apps, such as a URL to be opened in a browser. DSA allows redirection to shareable and semi-shareable apps. Note that an app can activate its own sharing mode by acquiring the sharing state

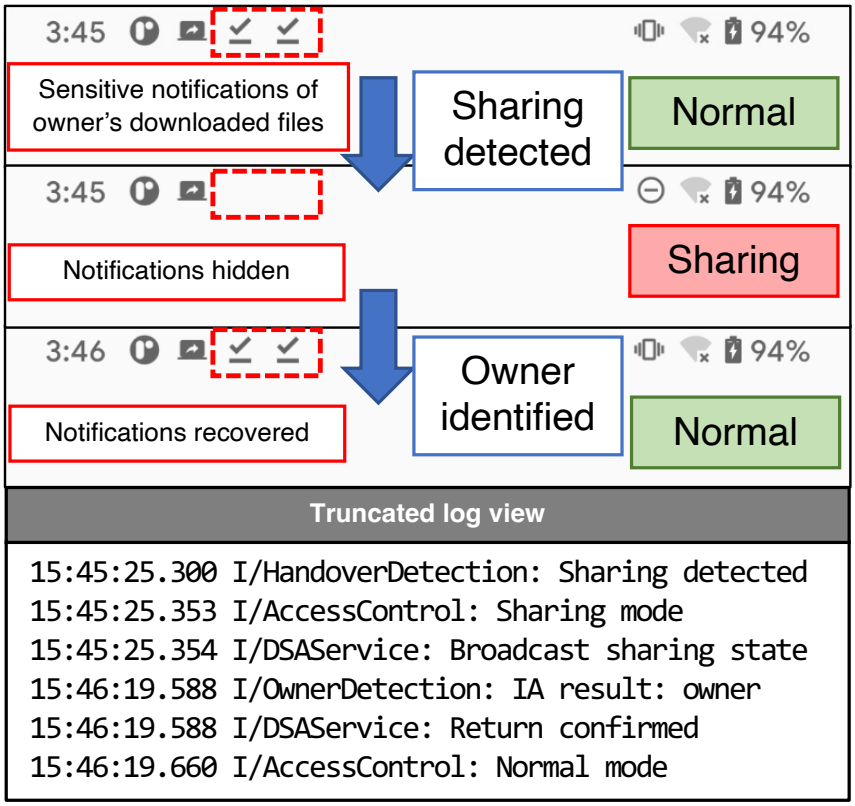


Figure 4.4: DSA Service Example: 1) At 3:45, DSA detected a sharing event and enabled the sharing mode; 2) During sharing, sensitive notifications were hidden, and DSA broadcast the sharing signal; 3) At 3:46, DSA identified the owner and ended the sharing mode with recovering the hidden notifications. Note: the first icon in the notification bar means the device is connected to a computer (for the logging purpose); the second icon indicates that DSA Service is running; the third and fourth icons are the sensitive notifications to hide.

from the DSA service at startup. A special situation is when the sharee switches to the home screen. There are two options to handle this situation: 1) redirect the sharee to the shared app since it may be a mis-operation; 2) allow the sharee to access the shared app and other shareable apps and only redirect to the shared app or lock the screen when a non-shareable app is accessed. We adopt the second solution for our implementation.

4.4.3 Implementation

We implement our demo DSA solution on Android as a service. Developers and researchers can incorporate DSA into their device sharing solutions for automatic (de-)activation. Developers of third-party apps can set up a broadcast receiver to obtain the sharing notifications from the DSA service for enabling their in-app sharing control. Figure 4.4 illustrates the log view and the changes of the notification bar at different states of an implicit sharing loop to reflect how DSA automatically handles device sharing. From the figure, we can see that DSA automatically hid the sensitive notifications after detecting a handover gesture and recovered them once the user changes back to the owner. During this process, the owner did not need to manually enable and disable the sharing mode.

4.5 Evaluation

We first evaluate handover detection as it plays an important role in starting the implicit sharing loop. Then, we test how DSA coordinates handover detection and owner detection to automatically handle sharing events. We also report the performance and battery overhead of DSA. We received approval from our IRB for the user study reported in this work.

4.5.1 Evaluation Setup

Study description. We conducted a user study advertised as "the evaluation of context detection techniques for smartphone sharing", and recruited 18 participants (5 females and 13 males) through word-of-mouth advertising. Eleven participants were between 18–29 years, five were between 30-39 years, and two were above 40 years of age. 13 participants were related to the field of Computer Science and the rest were in non-related fields. The study consists of two parts: handover detection and device sharing. Participants chose to complete the first part only or both parts. 10 of 18 participants completed both parts. Participants received $25 remuneration for completing the whole study ($15 for completing the first part only). Due to the pandemic, most experiments happened remotely and participants were instructed and supervised using a videoconferencing platform. Participants could choose to use a provided experiment smartphone or to install a data collection app on their devices. The phones recorded in the evaluation include Google Pixel, Google Pixel 3, Samsung S8, Xiaomi Redmi 5, and Huawei P9.

Model setup. We used both Support Vector Machines (SVM) and Neural Networks (NN) for model training. Considering the NN model's superior performance and the increasingly mature support for NN on today's smartphones, we adopt NN in our evaluation. The input layer of the NN is the feature vector of each segment. The model includes two hidden fully-connected layers using ReLU as the activation function: one 64-neuron layer and one 48-neuron layer. We apply 10% dropout in between two hidden layers to reduce overfitting. The output layer uses Sigmoid as the activation function since our gesture detection is a binary classification task. We use the cross entropy loss function and Adam optimizer for model training. In our experiments, we set the number of epochs as 120 and the batch size as 128.

As for the training data, we adopt a balanced setting where the positive and negative instances are evenly distributed in the training data, even if handovers are rare in the real world (i.e., extremely skewed distribution). If using the imbalanced training set that reflects the true distribution for training, the model would focus on detecting non-handover gestures. To show that our model can handle an extremely imbalanced testing set, we also evaluate the accuracy of handover detection over an all-negative dataset in § 4.5.2.

Metrics. Handover detection involves segmenting motion data, performing classification on each segment, and making decisions based on a number of consecutive segments. Thus, for segment-level classification, we evaluate the classifier performance based on its ROC curve, and therefore use AUC and EER. For event-level detection, we use precision, recall, and f1-score to evaluate the overall detection performance under different settings. To measure the reaction time of each positive gesture detection, we use its elapsed time after the moment when the participant receives the instruction to hand over the device.

4.5.2 Evaluation of Handover Detection

Handover detection evaluation tasks required participants to work in pairs. Participants were asked to hand over a smartphone from either their left or right hand in two different positions:

1. Face-to-face: the owner was in front of the sharee.
2. Side-by-side: the owner was next to the sharee (including right-to-left and left-to-right directions).

Participants also performed the handover tasks with random directions to provide diverse handover data, where they randomly adjusted their relative positions each time. For each

User#	1	2	3	4	5	6	7	8	9	10	11	12
AUC	0.98	0.98	0.98	0.98	0.99	0.99	0.97	0.97	0.97	0.96	0.97	0.96
EER	0.07	0.05	0.06	0.07	0.03	0.02	0.07	0.08	0.03	0.07	0.04	0.09

Table 4.1: Per-user model experiment.

User#	1	2	3	4	5	6	7	8	9	10	11	12
AUC	0.94	0.96	0.98	0.97	0.99	0.95	0.90	0.90	0.97	0.93	0.94	0.97
EER	0.11	0.10	0.09	0.10	0.04	0.12	0.16	0.15	0.07	0.15	0.11	0.09

Table 4.2: Cross-user experiment.

pair, one participant handed over the device to the other at least 20 times per direction. Then, they swapped roles and repeated. We also recorded motion data for activities that share similar patterns with handover gestures, such as switching hand, putting the device down, rotating the device, and other random movements with combining device rotations and movements at different directions. All participants completed each single-participant task (e.g., switching hand) 20 times. Each data clip of both handover and non-handover lasts 5s to 10s. We label data clips with handover events as positive events and other data clips as negative events. For positive clips, we asked the participants to start the handover activity at the 2$^{\text{nd}}$ second. In total, we collected 2044 positive and 1737 negative clips.

Model Transferability Evaluation

According to our design, we adopt a pre-trained handover gesture model that is supposed to work on a new user or a new device without retraining. Thus, we first evaluate the model transferability to different users and different smartphones and then evaluate the overall accuracy and reaction time. Since handover detection conducts classification over a fixed size of time segment, we divide each data clip into 2s segments and every two consecutive segments have 50% overlap. For positive events, we focus on data segments that have 50% overlap with this time interval and label them as positive segments. We label all the segments from negative events as negative segments.

Cross-user experiments. Ideally, a personalized gesture detection model trained with the owner's data should provide the highest accuracy, i.e., the model should be user independent. We expect a cross-user model, which is pre-trained with hybrid data from multiple users, to work on a new user and have a comparable accuracy as a personalized gesture model. As a reference, we first run a 10-fold cross validation to test the performance

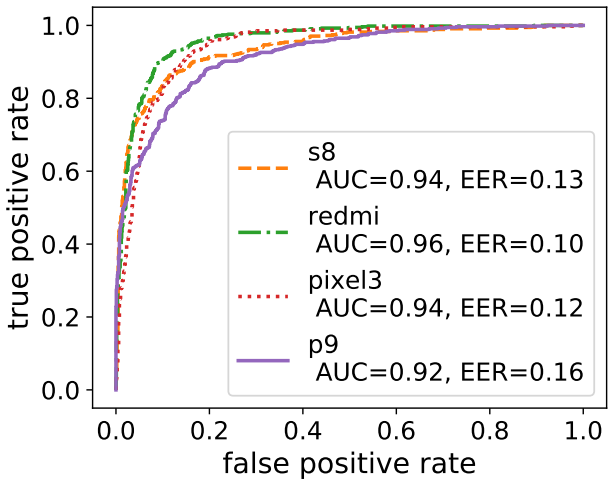


Figure 4.5: One-to-multi test (Pixel)

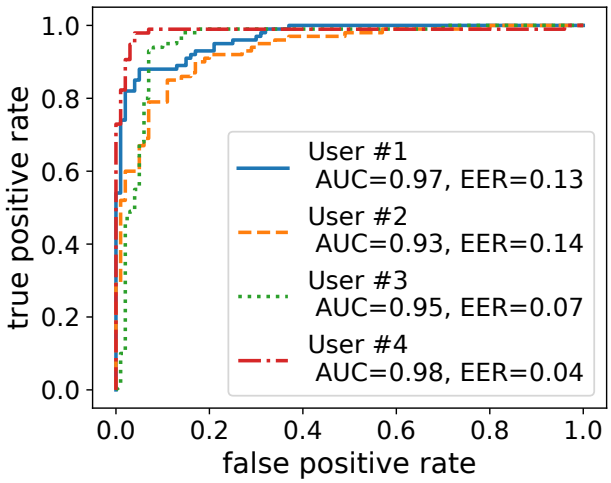


Figure 4.6: Multi-to-one test

of per-user classifiers for 12 participants using the same Google Pixel phone. As shown in Table 4.1, the AUCs of all classifiers are above 0.96 while the EERs are below 10%. Then, for the cross-user model, we use several users' motion data for training. We adopt the following protocol: for each participant, we train a model with 11 other participants' data and then test it on the chosen participant's data. Table 4.2 shows that the cross-user model can still provide a high AUC and a low EER when it is applied to a new user, where the worst AUC is 0.90 and the worst EER is 16%. We can observe an increase in EER by comparing cross-user models to per-user models, which implies a weaker ability to distinguish a new user's handover gesture from their other movements. Nevertheless, we apply a sliding window-based strategy for sharing event detection (see § 4.5.2) and additionally apply IA based owner detection (see § 4.5.3) to further mitigate false detection.

Cross-device experiments. In addition to being transferable across different users, the gesture detection model is also expected to work across different phone models. In the cross-device experiments, we added four other Android phone models: Samsung S8, Redmi 5, Google Pixel 3, and Huawei P9 and collected motion data of two participants for each phone model. We adopt the following two protocols to test cross-device accuracy:

1. We train a model with all 12 participants' training data on the Google Pixel and test it on the other four phones. As shown in Figure 4.5, the model trained with one phone's data shows a consistently good performance on other phones, where the AUCs are always above 0.92 and the EERs are around 10 to 16%.

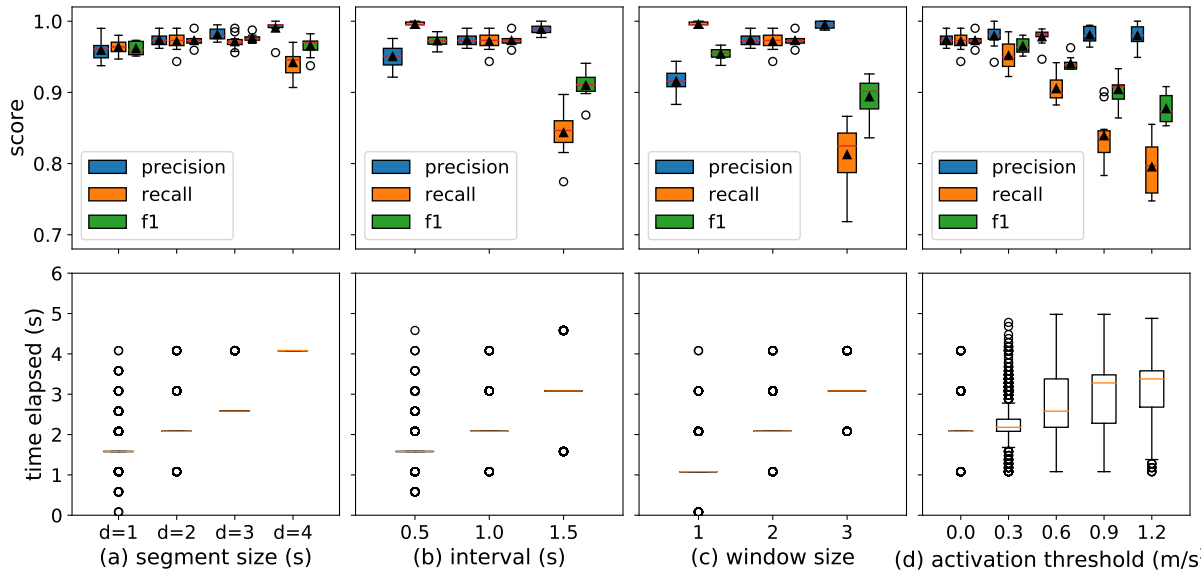


Figure 4.7: Event-level experiments: impact of different parameter on the detection performance and latency.

2. We use mixed training data from four phone models to train the model and test it on the fifth phone. As shown in Figure 4.6, the cross-device gesture detection model provides a consistently good performance across different phone models.

All the above experimental results have shown the transferability of the gesture detection model and also motivated us to use mixed data from different phones and users for model training.

Impact of Different Settings

Since we adopt a sliding window-base strategy, handover detection is performed over more than one segment to ensure the detection accuracy. Besides, DSA adopts adaptive sensing to balance detection accuracy, latency, and battery consumption. Therefore, we conduct controlled experiments to evaluate the impact of four factors: segment size d, interval p, window size w, and activation threshold θ. We divide all events into 10 subsets and

adopt 10-fold cross validation. We enable adaptive sensing only for the adaptive sensing experiments.

Segment size and interval. Intuitively, a larger segment size and a smaller interval provide better ability to cover a sharing gesture. We set the window size to two segments and the overlap between two segments as half of the segment size. We test different segment sizes: 1s, 2s, 3s, and 4s (intervals: 0.5s, 1s, 1.5s, and 2s, respectively). As shown in Figure 4.7(a) the 3s segment provides higher precision compared to the 2s segment setting, but it takes longer to make a detection. To balance the reaction time and accuracy, we set the segment size to 2s. Then, we test three different intervals with the same segment size. Figure 4.7(b) shows that a shorter interval has higher recall, while the precision is low. A larger overlap allows more classifications conducted in the same time period, so it helps improve recall and capture a gesture earlier.

Window size. Considering the length of a sharing gesture, we change the window size from one to three segments: the length of a window varies from 2s to 4s where the segment size is 2s and the interval is 1s. Figure 4.7(c) shows f1-score reaches the highest (median: 98%) when the window size is two segments. When the window size is three, the average recall decreases to 81%, and the average elapsed time is only 2s. When the window size is one, the average precision drops to 92%. This result shows the necessity of a window-based strategy to avoid false positives instead of directly using segment-level results.

Adaptive sensing. We set up the evaluation environment as follows: 1) low frequency mode: $f_s = 10Hz$ without classification task. 2) mode switch: if the acceleration exceeds θ, high frequency mode is activated; if the acceleration is below $0.1m/s^2$, low frequency mode is activated; there is a 90ms latency when mode switch happens, which is the maximum of 50 measurements on Google Pixel. 3) high frequency mode: $f_s = 50Hz$ with feature extraction and classification. We test five different thresholds: 0, $0.3m/s^2$, $0.6m/s^2$, $0.9m/s^2$, $1.2m/s^2$. Figure 4.7(d) shows that the recall score drops with the increment of θ. Due to the delay brought by frequency switching, it is likely to miss data at the beginning of a gesture. Nevertheless, when $\theta = 0.3m/s^2$, the recall score is still acceptable (mean: 95%).

Summary. According to the above experimental results, we use the following default settings, $d = 2s, p = 1s, w = 2, \theta = 0.3m/s^2$, to balance precision (mean: 0.98), recall (mean: 0.95), and reaction time (mean: 2.33s).

False Positive Evaluation

We train the model with the training data and the settings in the event-level evaluation and evaluate its long-term false positive rate using the HMOG dataset [159, 183]. The

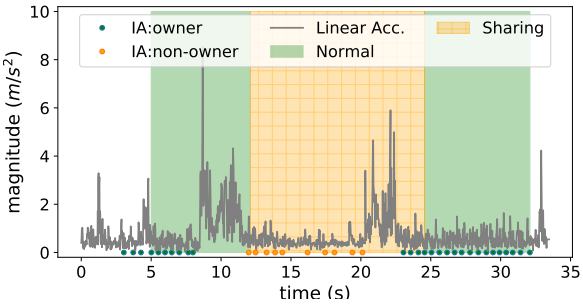
testing data involves 493 sessions (about 81 hours) of 100 smartphone users' reading and writing activities, but no sharing activities, while standing or sitting. For each session, we keep detection running even after a false positive is detected. The result shows that the hourly false positive rate for continuous device use is 0.9 per hour. Although a false positive makes DSA move to state "sharing" falsely, DSA can still switch back to state "normal" once the owner's identity is confirmed.

4.5.3 Device Sharing Processing

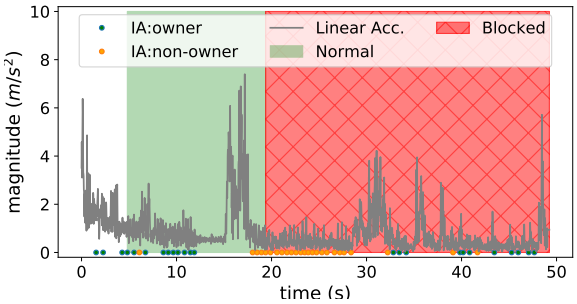
For the second part of our user study, we tested if DSA is able to automatically detect the sharing and the returning of a smartphone with the help of both handover detection and owner detection. Besides, we want to capture potential exceptions during the study.

Task description. We adopted the touch-based IA scheme [52] and used a (m, n)-sliding-window-based strategy (see § 2.4.1). Here, we set $m = 4, n = 7$ for balancing false rejection rate and false acceptance rate of touch-based IA. For IA enrollment, we collected 200 swipes from each participant to train the per-participant IA models. For handover detection, we train a model with the training data from the controlled experiments in § 4.5.2 and use the default settings. In each session, a group of two participants was asked to perform a web page sharing event: the owner shared a web page on the phone and handed the device to the sharee; after reading the page, the sharee returned the phone to the owner. Each participant was required to swipe at least 10 times during reading once the phone is in their possession. Once they completed the reading task, they swapped their roles and repeated the above process for 10 times in total. Note that we did not specify the position of each participant and how they handed over the device so that participants can hand over the device in their natural manner. In total, we collect 50 device sharing sessions from five groups of participants for analysis.

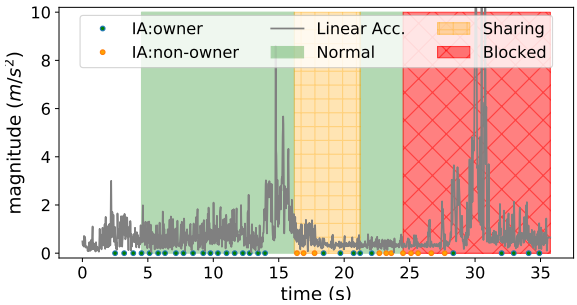
Results. DSA properly detected handover gestures and entered state "sharing" in 48 out of 50 sessions, which shows the effectiveness of handover detection. With the touch-based IA owner detection, DSA automatically detected the end of the sharing event and completed the implicit sharing loop in 42 sessions, which means an owner user did not need to explicitly start and end the sharing mode in these sessions. We note that the results were related to the performance of the selected IA scheme, which can be improved by using IA schemes with higher accuracy. Figure 4.8a shows an example of a successful procession: The owner was using the phone during the first 9 seconds and then handed it over to the sharee; DSA detected a handover gesture and switched to state "sharing" at 12s; after the sharee finished using the phone and returned the phone to the owner, the owner detector



(a) Example 1: error-free session.



(b) Example 2: gesture detection failure.



(c) Example 3: IA false acceptance.

Figure 4.8: A user study session consists of three stages: 1. owner uses the device and then hands it to sharee; 2. sharee uses the device and then returns it; 3. owner receives the device. The grey plot shows the intensity of the movements measured by the accelerometer. Green area: the device is in state "normal". Yellow area: the device is in state "sharing". Red area: the device is locked. Blue and orange points are the per-swipe results of touch-based IA, representing owner and non-owner, respectively.

detected the owner at 24.5s and switched back to the normal state.

Exception procession. We recorded all sessions with exceptions and exhibited how DSA processed them. In two sessions, handover detection failed to detect handover gestures but DSA blocked the sharee according to the negative IA results (see Figure 4.8b). In six sessions, DSA initially falsely identified the sharee as the owner. However, in four of these sessions, it correctly identified the sharee as non-owner within several seconds after obtaining more touch events from the sharee. For example, in Figure 4.8c, DSA falsely identified the sharee as the owner, and consequently, state "sharing" was left at 21.2s. However, after DSA detected several non-owner touch events, it locked the sharee out at

24.4s to prevent potential unauthorized access. For both cases, a sharee was temporarily recognized as the owner and might be able to access sensitive resources before the owner detector blocks the sharee. A possible solution to mitigating potential security threats brought by IA false detection is to set up stricter detection criteria for identifying the owner (e.g., requiring more positive swipes in a window size) in state "sharing". In addition to the above exceptions, DSA failed to recognize the owner after the device had been returned in only one session. In this case, the owner could still manually exit from state "sharing" by passing re-authentication. Note that these errors or exceptions may be specific to touch-based IA owner detection. Using or combining different biometrics may improve accuracy. Furthermore, the training data was collected from only brief reading tasks, which lacks diversity and may result in more false detections.

4.5.4 Performance Evaluation

Since DSA provides real-time detection of sharing activity, we need to consider CPU usage and battery consumption. Handover detection adopts an adaptive sensing strategy. Thus, we evaluated the performance for both high-frequency sensing and low-frequency sensing.

CPU overhead. We instrument the DSA service with Android Profiler and measure the CPU overhead on Google Pixel. We run the CPU profiling for 60 seconds and repeat this process for 10 times. For low-frequency sensing, the average CPU overhead in a 60-second session is 13.5ms (SD: 7.4ms). For high-frequency sensing, its average CPU overhead is 275.7ms (SD: 35.6ms) where the average CPU overhead of the classification process is 57ms (SD: 26.8ms). Other computations, such as using linear acceleration sensor, account for the remaining overhead. The result shows that adaptive sensing can effectively avoid unnecessary computations.

Battery consumption. We run the DSA service on Google Pixel in airplane-mode for 30 minutes without other running apps and repeat 5 times for both high-frequency sensing and low-frequency sensing. We use Battery Historian to estimate the battery consumption of DSA Service. As a reference, we leave the phone with screen always on, and the phone discharges 3% of battery in 30 minutes. The results show that the average estimated battery consumption of DSA Service alone for high-frequency sensing is 0.11% per half hour; the rate for low-frequency sensing is 0.06% per half hour. Therefore, the battery consumption is very small while adaptive sensing can further reduce battery consumption.

4.6 Discussion

Defending against unauthorized access. Given the observed latency of handover detection, we can conclude that DSA can swiftly activate the sharing mode and a sharee can hardly conduct effective attacks during such a short time interval. Even if handover detection fails to detect the handover gesture, owner detection can prevent a sharee from accessing sensitive resources upon negative IA results. As observed in the user study (see § 4.5.3), we found that the IA-based owner tracking was limited by the performance of its IA scheme. False acceptance of the IA scheme may temporarily deactivate the sharing mode so that a sharee can move to sensitive apps at this moment. Given the exception procession of DSA, it is possible to reject a sharee if the IA result is negative again. However, similar situations may result from a malicious sharee launching specific attacks on the adopted IA scheme (e.g., mimicry attacks [95]). A promising countermeasure is to adopt multiple modalities (i.e., multi-modal IA) to improve overall accuracy so that the failure of one modality is not likely to make owner detection fail. Chapter 5 shows how multi-modal IA provides better accuracy in terms of false rejections and false acceptances compared to single modalities. Besides, how to incorporate multi-modal IA into DSA will be our future work.

Limitations of sharing detection. We mainly investigate handover gestures as the trigger of an implicit sharing loop. We acknowledge the following two limitations: 1) The handover detection evaluation tasks involve two fixed positions and a random position. Therefore, they do not cover conditions such as a sitting owner handing a device to a standing sharee, in a vehicle, etc. 2) An owner may adopt other sharing actions, e.g., passing a device via a table, instead of handover gestures. As DSA's sharing detection is extensible, a feasible solution is to add models for these sharing actions. For better security, sharing mode can be enabled for these cases only if a non-owner is detected under certain contexts (e.g., at home).

PIN sharing. For DSA, we assume that a device owner initially holds the device and performs a sharing gesture, indicating a device sharing event. However, PIN sharing, another way of device sharing, breaks the assumption. An owner directly shares their PIN/password with a sharee in advance so that the sharee can unlock the device without any additional involvement of the owner. From the perspective of DSA, owner detection cannot distinguish PIN sharing from unauthorized access since it only captures non-owner access for both cases. However, a device sharing solution can be made aware of PIN sharing through two ways: 1) The shared PIN can reveal a user's identity. A device owner can set up two different PINs [5] for themselves (i.e., private use) and sharees (i.e., shared use),

respectively. If a user is using the PIN for sharees, it implies a sharing event. 2) A sharee can register their biometrics (e.g., fingerprint, face, touch) in the system so that they can be identified. The device sharing solution can activate the sharing mode once the current user's biometrics match any registered sharee's record. Otherwise, it identifies the current user as illegitimate and locks the device.

Evaluation limitations. To collect sufficient device sharing events in a short period, we asked participants to execute tasks in the second part of our user study. Some handovers in the first part of the user study required participants to follow specific position and direction instructions. These may have influenced their device sharing behaviors during the tasks in the second part. Besides, our analysis focused on how DSA handles a sharing device from a system's perspective. A potential avenue is to conduct a field study with our prototype DSA implementation so that we can investigate how DSA handles sharing events in the wild and collect smartphone users' perceptions about DSA.

4.7 Conclusion

This chapter investigates the device sharing context and presents DSA, a device sharing awareness solution for temporary smartphone sharing, which addresses Objective 2 of this thesis. DSA enables smartphones to conduct continuous and proactive device sharing detection with low latency and low power requirements. It provides flexible access control strategies to protect sensitive apps and resources from unauthorized access during sharing. Extensive experiments show that DSA can detect device sharing with high recall and low false positive rates.

For temporary device sharing, a sharee is a guest user who likely cannot be recognized via behavioral or physiological biometrics by the authentication system. However, as indicated by Matthews et al. [124], it is also common for multiple people to mutually use a single device for household sharing. In this case, if a mobile device adopts behavioral biometrics based IA, its IA mechanism should be extended to support multiple legitimate users. Thus, in Chapter 5, we investigate the multi-user IA problem and propose a multimodal solution.

On the other hand, we also notice that an authentication system may need distinct policies to handle various contexts or risks. For example, while an authentication system is devised to lock out a non-owner user, it needs to temporarily allow the sharee to access the device for device sharing despite biometric mismatch. In this chapter, we adopt a transition diagram to determine the workflow of our DSA solution so that it can support

multiple sharing loops and enable exception processing. Chapter 6 extends the idea and proposes a general adaptive authentication and access control framework that uses graph-based models to organize adaptation policies for different risks.

Chapter 5

Design and Evaluation of Multi-user Implicit Authentication Systems

5.1 Introduction

In Chapter 4, we proposed DSA to handle temporary device sharing where a sharee is not a registered user on the device. In this chapter, we investigate another type of sharing scenario where multiple (registered) users share the same device. Although many companies have introduced shared device modes (e.g., Microsoft Authentication Library [12], Cisco Meraki [127]) to smart devices to enable multi-user access, an open issue for shared smart devices is when to de-authenticate/log out a user. For example, when several shift workers alternatingly use the same device, user accounts need to be switched during a shift change. However, user logout and login often rely on manual operations. Failure in logging out from a shared device may expose one user's data to another user and incur impersonation.

Ideally, an automatic logout mechanism can detect user switches and reject illegitimate users in real time. IA is a potential solution for shared smart devices to detect user switches. However, most IA mechanisms were designed for the single-user scenario where the device owner is the only legitimate user. Few studies have investigated the design of IA systems for multi-user shared devices (i.e., multi-user IA). Thus, in this chapter, we propose multi-user IA for shared mobile devices to address Objective 3.

5.1.1 Requirements of Multi-user IA

Multi-user IA requires 1) rejecting a stranger from a pool of legitimate users and 2) distinguishing the legitimate users from each other continuously throughout a session. For instance, at a medical center, a shared device with patients' information should reject opportunistic intruders while accurately identifying different legitimate medical staff. Existing studies [28, 43, 63, 196] have proposed algorithms to solve multi-user IA as a multi-class classification problem. However, designing a multi-user IA system should additionally solve how to combine different IA algorithms for better identification accuracy and how to adapt IA models to new incoming users and data to mitigate performance loss over time [28, 52]. Also, it is necessary to consider the impact of these aspects on accuracy when evaluating a multi-user IA system.

5.1.2 Challenges

The research problem is how to design and evaluate a multi-user IA system that fuses multiple modalities and adapts to new users and incoming data. The main challenges include:

1. Due to the differences in the uniqueness of behavior biometrics and training data amount and quality, IA mechanisms based on different modalities may have different accuracy in identifying each user. Therefore, a challenging problem is incorporating the accuracy differences into the fusion of different modalities.

2. IA models are updated upon adding new users (i.e., user enrollment) and new data. A multi-user IA system should provide different model updating strategies to fit IA mechanisms based on various machine learning techniques.

3. Since new incoming data is collected during device usage, automatic data segmentation and labeling is another challenge given possible mid-session user switches.

4. For evaluation, measuring classification accuracy over individual data points is insufficient to test the performance of a multi-user IA system. It is essential to compose evaluation tasks using real-world traces to simulate common scenarios such as lunchtime attacks [89] and user switches. Also, evaluating model updating and data labeling requires tracking the accuracy of the system over time.

5.1.3 Contributions

We propose SHRIMPS[1], a multi-user, multi-modal IA evaluation framework that consists of an architecture for multi-user, multi-modal implicit identification and authentication and of an evaluation environment. SHRIMPS is targeted at IA researchers and developers and allows them to rapidly compose and evaluate different multi-user, multi-modal IA schemes.

SHRIMPS offers the following features: 1) a general extension of single-user IA mechanisms for multi-user; 2) a variety of score fusion strategies including a Dempster-Shafer (D-S) theory [154] based approach to combine different modalities for continuous authentication; 3) an automatic segmenting and labeling strategy to update IA models with new incoming data, 4) multi-user management supporting adding and removing users. The evaluation framework enables researchers to test and compare different IA schemes with consideration of different metrics, score fusion strategies, sufficiency of training data, and addition of new users. It also helps researchers and developers readily determine answers to key questions such as "Which fusion strategy provides the highest accuracy for the given set of multi-modal biometrics?", "How accurately does the system identify a newly added user?", "How much data is sufficient to train the new user?", or "How much performance gain does model updating bring to the system?". These questions can be answered by creating simple storyboards where researchers and developers can rapidly compose simulation tasks by concatenating data segments from public datasets. SHRIMPS also provides insightful metrics to measure session-level performance to understand the impact of false detections.

In summary, our contributions include:

- We propose a multi-user, multi-modal IA evaluation framework, SHRIMPS. It supports the design and evaluation of multi-user IA schemes that leverage multiple modalities to continuously detect unauthorized access from strangers and identify the user from a group of legitimate users in real-time.
- SHRIMPS provides model updating with new data and user enrollment and removal to ensure high accuracy for multiple users across a session. It can automatically segment and label newly collected behavioral data based on authentication results and user feedback.
- SHRIMPS supports different score fusion methods, including a score fusion method based on D-S theory [154], to combine authentication scores from multiple modalities.

[1] SHaRing-oriented IMPlicit authentication System

- Storyboards enable easy and flexible construction of evaluation tasks using public datasets, which allows IA researchers and developers to evaluate their IA schemes based on simulating complicated practical scenarios.
- We provide two use cases of SHRIMPS to showcase how it supports designing and evaluating different multi-user IA schemes. We test the effectiveness of our multi-modal, multi-user IA solution in these use cases based on four public datasets: HMOG [159], BB-MAS [16], IDNet [54], and Touchalytics [52]. In addition to commonly reported metrics (e.g., false acceptance/rejection rate), we show how to use an additional metric, Gini Coefficients [41], to analyze error distributions in SHRIMPS.

5.2 Multi-user IA Problem

In this section, we formulate the multi-user, multi-modal IA problem and provide the threat model.

5.2.1 Authentication Model

Definitions and Assumptions

In a multi-user IA system, two or more users are allowed to access a device. We define a user who is registered and has full or partial access to the device as a *legitimate user*. We define a *session* as the period of user-device interaction that starts from when the device is unlocked with EA, such as a PIN, to when the device is locked. The IA system continuously identifies the current user and verifies their identity throughout a session. As a consequence of failed IA, the system locks the device and asks for EA. Inspired by recent device/account sharing studies [7, 123, 124], we consider multiple users sharing the *same* smart device, and therefore, participate in the *same* session alternatively, where there may be more than one legitimate user during a session. We assume that there is only one user interacting with the device at any moment. For example, a shared tablet is running a kiosk app for medical staff to look up and process patients' data. A session starts when a medical worker turns on the tablet, and any legitimate medical worker can access the device afterwards. A session ends when the tablet is turned off or detects unauthorized access. It requires continuously and implicitly (re-)identifying the new user from all other legitimate users in real-time during a session.

Problem formulation

The multi-user IA problem is a multi-class classification problem. We denote the legitimate user set as $\mathcal{U}^+ = \{u_0, u_1, \ldots, u_{n-1}\}$, where n is the number of legitimate users, user u_0 is the *primary user* (i.e., *owner*) of the device, and users $u_i, i > 0$ are *secondary users*. We define a *null user* or *attacker* as a user who is not registered and has no access to the device, which is denoted as u_{-1}. The whole user space for a multi-user authentication system is defined as $\mathcal{U} = \mathcal{U}^+ \cup \{u_{-1}\}$.

For accurate identification and authentication, the system adopts multiple IA mechanisms (i.e., *authenticators*). The basic workflow of each authenticator is to extract features from sensor measurements and perform multi-class classification. An authenticator can be described as a function $\boldsymbol{s} = M(\boldsymbol{f})$, where $\boldsymbol{s} = \{s_{-1}, s_0, s_1, \ldots, s_{u-1}\}$ represents the normalized scores of all instances in $\mathcal{U}$, and $\boldsymbol{f}$ is the feature vector. Then, each authenticator obtains a series of feature vectors with timestamps $\{(t_0, \boldsymbol{f}_0), (t_1, \boldsymbol{f}_1), \ldots, (t_k, \boldsymbol{f}_k)\}$ and generates a series of score vectors $\{(t_0, \boldsymbol{s}_0), (t_1, \boldsymbol{s}_1), \ldots, (t_k, \boldsymbol{s}_k)\}$ accordingly, where k is the number of the classification times performed within a given period. The system then identifies the user and decides whether to lock the device. Thus, the multi-user, multi-modal IA problem is about combining different authenticators to obtain who is the most likely user.

5.2.2 Threat Model

For multi-user IA, possible attackers include strangers and legitimate users. A stranger attacker is physically close to the unlocked device and attempts to access sensitive resources, which is a lunchtime attack [89]. A legitimate user attacker may intentionally or accidentally access the *previous* legitimate user's personal resources. For both cases, the authentication system should reject their access and de-authenticate the current user. We assume attackers do not have or know the victim's credentials (e.g., password, PIN) for explicit authentication. We make the assumptions about the device and its operating system as § 2.6. Since our work focuses on a general multi-user IA framework, mimicry attacks [95] that target specific behavioral biometrics are out of our scope. Nevertheless, we test the system under the scenario where the accuracy of one authenticator is significantly lower than other authenticators (see § 5.6.3).

5.3 Our Approach

SHRIMPS first addresses the multi-user IA problem in § 5.2.1 from the following three aspects: 1) a general extension strategy to extend existing IA algorithms into multi-user, 2) a score fusion method to combine multiple modalities, and 3) new incoming data and user enrollment for model updating.

5.3.1 Multi-user Identification

A multi-user IA model is an $n+1$-class classifier for a system with n legitimate users, where negative instances (i.e., imposters) in a binary classification problem are now u_{-1}. Extending an IA mechanism based on binary classification requires an imposter training set to provide negative training data. In SHRIMPS, the imposter training set is sampled from multiple random users' data to represent a "general" user's behavioral biometrics. Besides, the machine learning technique adopted by a multi-user IA mechanism should support multi-class classification. We can adopt the generic "one-vs-the-rest" strategy to extend their models into multi-class classifiers:

1. For each class u_i, we construct a training set with labeling u_i as positive class and all other classes as negative class.
2. We train $n+1$ sub-classifiers for all $n+1$ classes using the training sets constructed in step 1.
3. For authentication, the authenticator calculates the normalized scores of the positive classes from all sub-classifiers and constructs a score vector as the output.

Multi-user scenarios also result in the user data imbalance problem, where we have different amounts of training data for different users. For example, a multi-user system may collect more training data for the owner compared to the other users since the owner usually spends more time doing various activities with the device. Thus, we need to balance the training data by resampling techniques, including downsampling the data for the majority classes and oversampling the minority classes (e.g., SMOTE [29]). But the resampling techniques cannot fully address the accuracy degradation problem [48]. We still need to consider the accuracy imbalance among different users for decision making. We elaborate on this challenge as a part of the score fusion strategy in § 5.3.2.

To achieve multi-user identification, SHRIMPS handles the generation of balanced training data from a user's historical data and the imposter training set, and provides a generic wrapper to extend existing IA mechanisms into multi-class classification (see § 5.4.1).

5.3.2 Multi-modal Score Fusion

We fuse the results of multiple authenticators at score-level to provide accurate identification for multiple users since it allows each modality to work separately. SHRIMPS is designed to support various score fusion methods to aggregate the results from multiple modalities to make decisions. However, the scores produced by different modalities may have different implications such as the likelihood of each user, the similarity to a user's behavioral profile, etc. Also, it is necessary to take the uncertainty of each modality into account. Thus, calculating the average score is not sufficient. In SHRIMPS, we also adopts the Dempster-Shafer theory [154] for score fusion since it is proposed to combine evidence (i.e., scores) from different sources (i.e., modalities) with uncertainty, which is usually applied for sensor fusion problems [180]. Smith et al. [160] adopted a D-S theory based score fusion method for multi-modal IA schemes in the single-user scenario. In our study, we explore the multi-user D-S theory based score fusion method by decomposing the problem into $n+1$ binary cases.

For $u_i \in \mathcal{U}$, there are two mutually exclusive states: positive S_i and negative $\bar{S}_i$. The frame of discernment Ω_i is defined as $\Omega_i = \{S_i, \bar{S}_i\}$. All subsets in the power set $2^{\Omega_i} = \{\emptyset, \{S_i\}, \{\bar{S}_i\}, \Omega_i\}$ are assigned a *basic belief mass* within $[0, 1]$, denoted by m, where $m(\emptyset) = 0$, $\sum_{A \in 2^{\Omega_i}} m(A) = 1$. For an authenticator M that outputs a score vector $\boldsymbol{s} = \{s_{-1}, s_0, s_1, \ldots, s_{n-1}\}$, we define its uncertainty on each class (i.e., user) as $\boldsymbol{v} = \{v_{-1}, v_0, v_1, \ldots, v_{n-1}\}$. For each class, we construct the masses attributed for all hypotheses in 2^{Ω_i} as:

$$m(\emptyset) = 0, m(\{S_i\}) = (1 - v_i)s_i,$$
$$m(\{\bar{S}_i\}) = (1 - v_i)(1 - s_i), m(\Omega_i) = v_i.$$

To combine the masses of hypothesis $A = \{P_i\}$ from two authenticators M_p and M_q (the belief functions are denoted by m_p and m_q, respectively), we use Dempster's rule of combination to calculate its joint mass as

$$m(A) = m_p(A) \oplus m_q(A) = \frac{\sum_{B \cap C = A \neq \emptyset} m_p(B) m_q(C)}{1 - \sum_{B \cap C = \emptyset} m_p(B) m_q(C)}.$$

The combined belief $\text{Bel}(\{S_i\}) = \sum_{A|A\subseteq\{S_i\}} m(A) = m(\{S_i\})$ is the fused score for u_i from multiple authenticators.

We determine the uncertainty $\boldsymbol{v}$ of each authenticator by their model accuracy based on the following observations: 1) An authenticator may have a better accuracy detecting certain classes compared to others. 2) Different authenticators may have different accuracy for the same class. Intuitively, a higher accuracy on a certain user u_i should contribute to a lower uncertainty v_i. In our work, the system leaves 10% of the collected data out of the training data for each authenticator to construct their validation sets. Then, it evaluates all IA models with their corresponding validation sets at each model training or updating. The accuracy metrics include the per-user area under the receiver operating characteristic curve (AUROC) and equal error rate (EER), the threshold value for the equal false acceptance rate and false rejection rate of each user. We adopt two uncertainty functions based on either AUROC or EER. Given the authenticator M and the target user u_i, the uncertainty is calculated as:

$$v_{M,i}^{\text{AUC}} = \min(0, 1 - \text{AUROC}_{M,i}), \tag{5.1}$$

$$v_{M,i}^{\text{EER}} = \max(1, 2 * \text{EER}_{M,i}). \tag{5.2}$$

Assume there are k authenticators $\mathcal{M} = \{M_0, M_1, \ldots, M_{k-1}\}$, and the average score vector of all authenticators is denoted as $\{\bar{\boldsymbol{s}}_0, \bar{\boldsymbol{s}}_1, \ldots, \bar{\boldsymbol{s}}_{k-1}\}$. We use the D-S theory to merge the average score vectors of all authenticators. For each class, we obtain the fused score for each user $\hat{s}_i = m_0(\{S_i\}) \oplus m_1(\{S_i\}) \oplus \ldots \oplus m_{k-1}(\{S_i\}), i \in \{-1, 0, 1, \ldots, u-1\}$. Finally, we choose the most likely user by $\text{res} = \underset{i\in\{-1,0,1,\ldots,u-1\}}{\arg\max} \hat{s}_i$ as the current user.

5.3.3 New Incoming Data and Users

In practice, IA models are not constant: 1) When a new user is added to the system (i.e., *user enrollment*), IA models need to be updated to identify new users. For user enrollment, the new user needs to complete a series of tasks or use the device for a period of time, while the system is collecting and labeling behavioral data for initial model training. In a deployed system, user enrollment would be initiated by an administrator. In SHRIMPS, user enrollment is indicated in the storyboard underlying the evaluated IA scheme. Different from the single-user scenario, adding a new user introduces a new class to the existing IA models. 2) IA mechanisms require model updating with new incoming data to mitigate accuracy degradation over time. During normal device usage, the system is also collecting biometric data while authenticating and identifying the user. Unlike user

enrollment, the system does not always know the ground truth of who is currently using the device. Thus, we need to address the following problems:

Auto labeling. The common labeling strategy of single-user IA systems [34,92] is to label all incoming data as the owner's if no attack is detected. However, for multi-user systems, a piece of behavioral data may involve several users given possible user switches. Thus, the system needs to split the data into segments, where each segment contains *only one* user's usage data. Then, it finds out the corresponding user for each segment. Although external signals (e.g., screen-on/off) may imply user switches and indicate the start and end moments of a segment, they are insufficient to cover all user switches in a shared session. A multi-user IA system can continuously identify the current user and provide coarse-grained segmentation—knowing who is using the device during which segment. However, the time taken to collect sufficient data for decision-making is not negligible (evaluated in § 5.5). If a user switch is detected based on identification results without the help of external signals, the system discards the data collected during a time period (e.g., maximum detection latency) before the detected user switch since its ownership is uncertain. For the remaining data, the system labels the pre-switch part as the former user and the post-switch part as the latter user.

Model updating. User enrollment and new incoming data correspond to class incremental learning and data incremental learning, respectively. There are three types of model updating strategies:

1. *Full retraining* is applicable for all IA mechanisms. Models are retrained with all new and historical data. However, it occupies the most space.

2. *Partial fitting* is applicable for implementing data incremental learning to specific machine learning techniques, such as SGD-based techniques [131] and Naive Bayes classifiers. They can update a trained model with new data without keeping the historical data.

3. *Incremental learning techniques* are applicable for DNN-based IA mechanisms [157, 196]. ContAuth uses EWC [97] and iCaRL [145] to update a model without storing all historical data.

In SHRIMPS, we determine the suitable model updating strategy based on the machine learning techniques adopted by the IA mechanisms: we apply incremental learning based solutions for DNN-based IA mechanisms; for other IA mechanisms, we adopt full retraining for class/data incremental learning or partial fitting for data incremental learning.

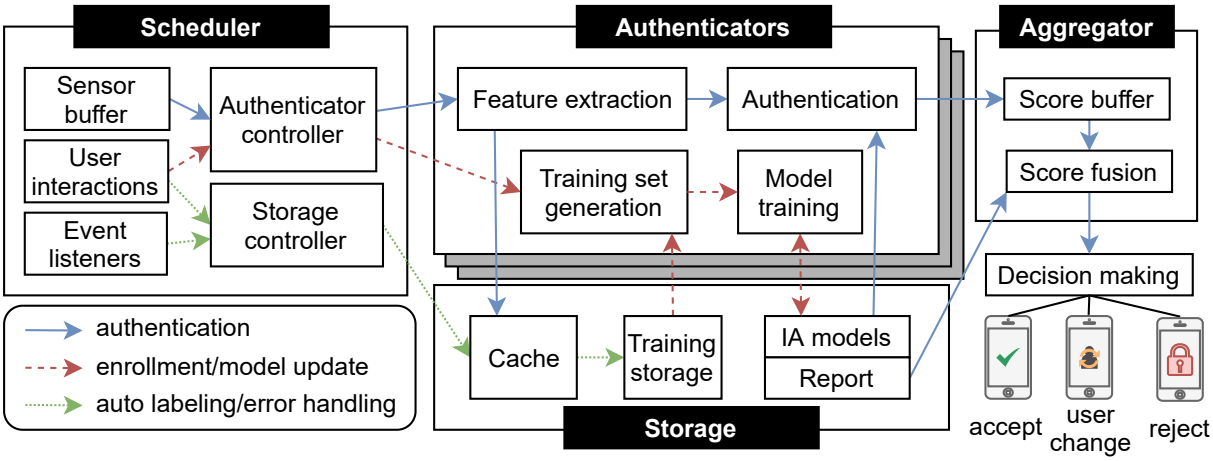


Figure 5.1: Architecture of the SHRIMPS framework

SHRIMPS handles user enrollment and new incoming user in two steps: automatically segmenting and labeling the collected data, and updating IA models for all authenticators with appropriate strategies. Besides, it listens to the user's feedback to correct falsely labeled data. We describe the detailed workflow in § 5.4.1.

5.4 The SHRIMPS Framework

In this section, we propose a multi-user, multi-modal IA evaluation framework, SHRIMPS, which consists of a multi-user IA system and an evaluation environment.

5.4.1 Multi-user IA System

We abstract the main components and behaviors of a multi-user IA system for shared smart devices, including user management, model training and updating, sensor data processing, and authentication.

Architecture

Figure 5.1 shows the architecture of a multi-user IA system, which comprises of four modules: the scheduler, the authenticators, the aggregator, and the storage.

Scheduler receives sensor events and external signals, and coordinates authenticators and the storage module. The scheduler receives and caches the incoming sensor events in the sensor buffer. The authenticator controller is responsible for activating authenticators and invoking authentication or model training. Whenever there is sufficient sensor data for a specific authenticator, the authenticator controller activates that authenticator and dispatches the required sensor data. The scheduler also maintains a set of event listeners to receive and process external signals for auto labeling, model updating, and error handling (see § 5.4.1). External signals, such as screen-off, imply the end of a session or a possible user switch, resulting in clearing cached data, data segmentation and resetting the authentication status of the system. Besides, user feedback that occurs after an erroneous rejection or user switch decision is an important signal for error handling. In response, the scheduler fixes wrong labels of the cached data and sets the authentication status as authenticated.

Authenticators are responsible for providing the essential functions, including feature extraction, model training, and classification. Researchers can provide their own IA mechanisms by specifying these essential functions. If a provided IA mechanism is based on binary or one-class classification, SHRIMPS applies the multi-user extension introduced in § 5.3.1. For each authenticator, the feature extraction function takes raw sensor data as input and produces feature vectors as output. The authentication function feeds the feature vectors to the trained IA models to calculate the authentication scores. The model training function takes two sets of labeled feature vectors as input for training and testing, respectively. Internally, the model training function can further sample a subset of the training dataset for validation, which is usually used for tuning the hyperparameters of IA models. The testing dataset is used to pre-evaluate the accuracy of an authenticator. An authenticator needs to store the pre-evaluation results for the certainty calculation of multi-modal fusion.

The training set generation function is responsible for generating training and testing data for the authenticator. The function loads the history feature data of each user in the training storage and samples negative training data from the imposter training set. All the fetched data is used to construct a labeled dataset. It is optional to apply resampling techniques to produce a balanced dataset (i.e., all classes have the same data size). The processed data is divided into two parts in a configurable ratio for training and testing, respectively, which is provided for the model training function.

Aggregator collects and fuses authentication scores. Since score vectors from various authenticators arrive at the aggregator asynchronously, our strategy is to let the aggregator cache the recent score vectors within a specified time interval and fuse the scores based on the steps in § 5.3.2 (note: the multi-user also supports other score fusion methods such as

average and weighted average). The cache is cleared at the session end or a user switch through final decisions or external signals. In addition, we adopt a (m, n)-sliding window: If at least m out of n results are the same, the aggregator adopts that result as the final decision; otherwise, it waits for more scores to make decisions. There are three types of decisions: accepting the user as the identified one, rejecting the user, and detecting a user change from one to another. Accordingly, there are three types of false decisions: 1) false acceptance (FA): the system falsely accepts an attacker, 2) false rejection (FR): the system falsely rejects a valid user, and 3) false identification (FI): the system identifies a valid user as another. We explore error handling in the next subsection.

Multi-user IA System Workflow

We present the workflow of a multi-user IA system performing the following operations:

User enrollment & removal. SHRIMPS support user enrollment and removal events as external signals. For user enrollment, the system does not conduct authentication and only collects behavioral data for the new user. A piece of labeled behavioral data is directly added into the training storage. Model training is triggered as follows: authenticators fetch the training data from the storage, generate training datasets, and train their models. The models and their pre-evaluation reports are stored in the storage. User removal requires indicating the target user. SHRIMPS supports the following two options for removing a user: 1) If the system stores users' historic behavioral data, authenticators fully retrain IA models with all data except for the removed user. 2) If an IA model consists of several per-user classifiers, the system can remove the removed user's classifier. The removed user's behavioral data is removed from the training storage and excluded from any future model updating.

Authentication. The authentication system continuously collects sensor data in the scheduler. Once the authenticator controller detects sufficient data for a certain modality, it calls the corresponding authenticator with the cached sensor data. The authenticators extract features, load the saved model from the storage, and then conduct classification to obtain score vectors. Score vectors from different authenticators are sent to the aggregator for score fusing. Finally, the system determines whether to accept or reject the current user based on the fused score.

Model updating. SHRIMPS takes both external signals and identification results to segment and label the data automatically and dynamically. Whenever the scheduler receives an external signal or the aggregator detects a user switch, SHRIMPS labels the collected data as a segment with the previously identified user and stores them in the training stor-

age. It can automatically correct the misclassified data points of individual authenticators based on the overall decisions. For example, if a single data point is u_{-1} (i.e., attacker) and the overall decision is acceptance, the system will fix the label of this single data point. If the device mistakenly locks the user out, it can correct the detection as well as the labels for cached features based on the user's feedback (see error handling below). Once sufficient new data is collected, the authenticators update the models for the existing users based on their data incremental learning strategy (see § 5.3.3).

Error Processing. Error processing of an IA system takes the following measures based on the error types: 1) False acceptance may temporarily expose the device to an attacker. Since the system is continuously authenticating the user, it will stop the attacker whenever a rejection decision is made. 2) False rejection leads to explicit authentication. If SHRIMPS receives a legitimate user's feedback (i.e., the user has passed EA), it can correct the labels of the collected features and update the IA models. 3) False identification is not as obvious as the other two errors. Immediate user feedback is not guaranteed if there is no mandatory EA to verify a user's identity. Nevertheless, we can handle false identification using the following strategy: if the system detects frequent user switches within a short time (e.g., the user has changed more than two times in five consecutive decisions), it will issue a rejection decision and a request for identity confirmation. Once the system receives the user's feedback, it will correct the labels accordingly.

5.4.2 Evaluation Framework

Motivation

Evaluating a multi-user IA system requires testing under various conditions. It involves measuring accuracy with different user numbers and different training data sizes, and detection latency for identifying a user after a user switch. As an IA system updates its models with new incoming data and users, it is also necessary to track the overtime accuracy change considering the impact of auto labeling. Moreover, the impact of a false decision may have different implications for continuous authentication: For example, a user is more sensitive to false rejections since they interrupt device use, while an individual false acceptance is tolerable as long as the system rejects an attacker within a reasonable time. A real-world user study is usually adopted to test the system usability under practical settings. However, it is hard to capture unauthorized access and device theft in the real world [74]. Specifically, it is challenging to conduct user studies for multi-user scenarios (e.g., controlling the conditions for valid users and attackers). To trade off, trace-based evaluation is a good option for conducting tasks using real-world public datasets.

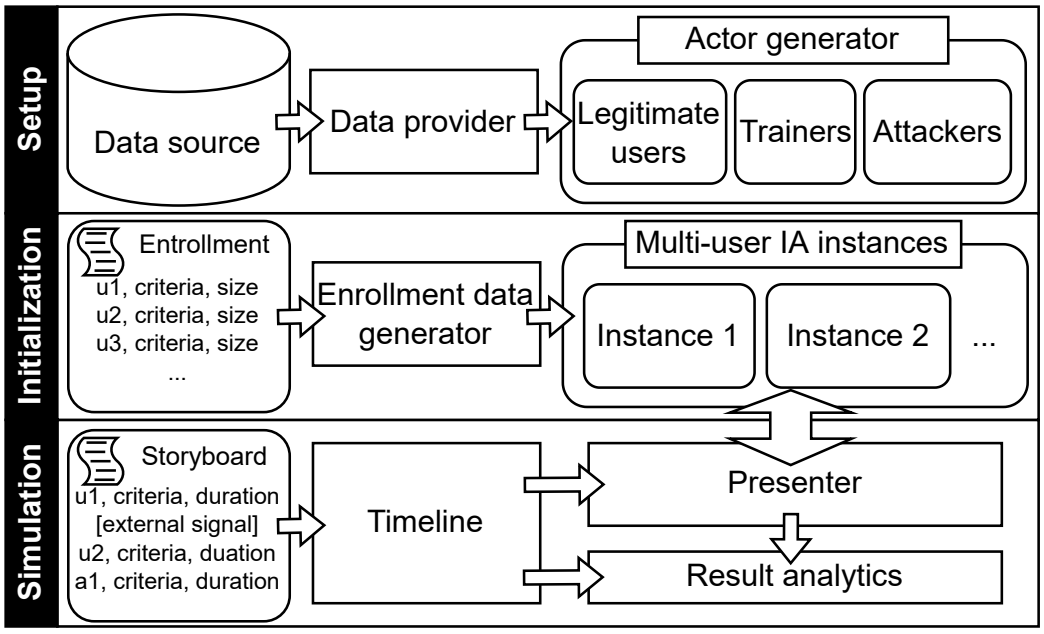


Figure 5.2: Evaluation framework

SHRIMPS enables researchers to stitch together data from public datasets and easily compose evaluation tasks based on specific requirements without falling into two common evaluation pitfalls [55]: 1) Non-contiguous training data selection, and 2) attacker data in training. It supports external signals, user enrollment and user feedback (i.e., reactions to decisions). Besides, we introduce session-level metrics in addition to decision-level metrics to compare different strategies and understand the practical performance of the system.

Evaluation Process

As shown in Figure 5.2, the evaluation process is divided into three stages. We introduce the components of the evaluation framework and their functions at each stage:

Setup. Researchers determine the data source. A data provider manages the connection to a public dataset, parses raw sensor data, and provides an interface for data retrieval. Internally, a user's data is stored in *blocks*, where each block contains sensor data of a user collected over a continuous period of time. It ensures contiguous data selection in chronological order, and no data in the training data blocks will appear in the evaluation tasks. The actor generator fetches a complete list of users via a data provider and randomly

selects a specified number of actors from the list. There are three actor types: legitimate users, trainers, and attackers. As defined in § 5.2.1, legitimate users should be enrolled in and identified by the IA system. Trainers provide negative training data to construct an imposter training set. Attackers attempt to access the device and should be blocked by the system. The user splitting of SHRIMPS ensures the attackers' data will not be used in the model training of legitimate users.

Initialization. The initialization stage determines the initial status of the authentication system. An enrollment script is required to determine which legitimate users have enrolled and how much training data has been collected for each user. The enrollment data generator parses the enrollment script and fetches training data via the data provider. Then, SHRIMPS instantiates the multi-user IA system, and adds the specified legitimated users and their training data for the initial model training. Multiple instances that adopt different schemes can co-exist in the same environment so that we can compare different schemes with the same conditions and inputs.

Evaluation. We introduce a *storyboard* to help researchers quickly design evaluation tasks. A storyboard lists one or a series of data blocks with specifying the actor, the selection criteria (e.g., activity, location), and the duration of each block. It provides the ground truth of data segmentation. To describe a session with the participation of multiple users, one can concatenate multiple data blocks of different actors. External signals, such as "screen off" and "screen on", can be added in between two data blocks to mark the start and end of a session. Besides, SHRIMPS supports adding user enrollment events during an evaluation task. We show simplified storyboards in § 5.6 and § 5.7.

According to the storyboard, SHRIMPS can fetch the matched sensor data and automatically generate a *timeline* comprised of a series of events in chronological order. The timeline automatically adjusts the sensor event timestamps of each block to ensure that the new timestamps of every two consecutive blocks are coherent. Assume that a new block with m events, $\text{sess} = \{(t_0, data_0), (t_1, data_1), \ldots, (t_{m-1}, data_{m-1})\}$, is appended to a timeline, where the last event timestamp of the timeline is T. The new timestamps are adjusted as follows: $t'_i = t_i - t_0 + T + \Delta t, i = 0, 1, \ldots, m-1$, where Δt is the customized interval between two segments. The presenter is responsible for processing the timeline and communicating with the instances: While passing each event to the instances, it also receives and answers their decisions. If a false decision is made, the presenter records it and produces a user's feedback for correction. After traversing the entire timeline, SHRIMPS saves all scores and decisions. The result analytics module generates the metrics by comparing each decision with the ground truth provided by the timeline.

Measures & Metrics. Multi-user IA systems are evaluated at two levels: decision-level

and session-level. For decision-level, all accuracy metrics are defined in terms of decisions. Based on the three error types, we use three basic metrics based on the three types: FAR, FRR, and FIR, respectively; see § 2.8. SHRIMPS uses Gini coefficient (GC) [41] to supplement decision-level FAR, FRR, and FIR for analyzing error distribution. A high Gini coefficient means that errors are concentrated in a small group of users. Session-level metrics aim to help understand the practical impact of false decisions on the whole session. We define session-level errors based on the following criteria:

- False acceptance: the system fails to reject an attacker within a specific time period (i.e., valid attack window).
- False rejection: the system makes *at least one* decision to reject a valid user during the whole session.
- False identification: the system makes *at least one* false identification during the whole session. For user switches where the user changes from one to another without any external signals, we allow the system to take a specific delay (i.e., uninformed switch window) before making the correct decision. During this period, any false identification is ignored since it does not block the user.

Accordingly, we define session-level FAR, FRR, and FIR by dividing the corresponding error number by the total session number. In addition, we record the moment t_d of the first correct decision to measure the detection latency, which is calculated by subtracting the starting timestamp of the session t_0 from t_d.

5.4.3 Evaluation Workflow

IA developers and researchers can use SHRIMPS to design and evaluate multi-user IA schemes according to the following steps: The first step is to build and configure the multi-user IA system, including adding authenticators, specifying the score fusion strategy, and adjusting the auto labeling and model updating behaviors. Researchers can choose to add their own IA mechanisms/score fusion strategies or use the built-in ones provided by SHRIMPS. The second step is to connect to a data source and generate actors. Researchers need to provide the source dataset and its data provider. SHRIMPS includes example data providers for the HMOG dataset [159], the BB-MAS dataset [16], the IDNet dataset [54], and the Touchalytics dataset [52]. Actor generation requires a random seed and the numbers of each actor type. The third step is to design an enrollment script and a

storyboard. Then, SHRIMPS can run the evaluation task and output the raw results accordingly. Based on the external signals in the storyboard, the result analyzer can segment the evaluation results into sessions and produce the per-session results automatically.

5.5 Evaluation Setup

We present two sample use cases that use SHRIMPS to design trace-based tasks and evaluate multi-user IA schemes. This section introduces the common evaluation setup for sample use cases.

IA mechanisms. For demonstration, we choose touch-based and gait-based IA mechanisms (see § 2.4) and use SHRIMPS to adapt state-of-the-art algorithms to multi-user identification: 1) touch-based IA is based on the feature extraction algorithm of Touchalytics [52], and SHRIMPS enables the multi-class classification following the extension strategy in § 5.3.1. 2) gait-based IA adopts the DeepGait algorithm [196], which already supports multi-user identification. For the gait authenticator, the sampling rate of motion sensors is set to 50Hz. The authenticator extracts gaits from a 1024-sample segment and is set to perform authentication every 512 samples (=10.24s). Thus, every two consecutive segments have 50% overlap. The training data generation for each authenticator is delegated to SHRIMPS. We adopt the same settings for data balancing: using SMOTE to oversample minority classes and ensuring that all classes (including the negative class) have the same training size. Note that SHRIMPS also supports researchers to compare different balancing methods and parameters to find the best settings.

Data source. In the evaluation, we use the four public datasets introduced in § 2.7: HMOG, BB-MAS, IDNet, and Touchalytics. For all datasets, we set the trainer size to ten users so that SHRIMPS randomly select ten users to provide behavioral data for the negative class. As introduced in § 5.4.2, SHRIMPS excludes these users from the legitimate user and attacker selections, ensuring no overlap between trainers and attackers to avoid the attacker-data-in-training pitfall. The datasets for evaluation should include multiple users, sufficient cross-session sensor data for each user, and multiple modalities. Since only the HMOG dataset meets all requirements, our sample use cases use only it for most of the evaluations.

BB-MAS does not provide sufficient gait data (i.e., only one 10-minute task for a user), while Touchalytics only provides touch data and IDNet only provides gait data. Due to the lack of qualified public datasets, a compromise solution adopted by existing studies [63, 64, 74, 114] is to fuse multiple datasets for different modalities that are independent of each

other and rely on different sensors (e.g., touch and gait). Therefore, we fuse the IDNet dataset and the BB-MAS (or Touchalytics) datasets as follows:

1. We map the 14 users who provided three or more tasks from IDNet to 14 users randomly selected from BB-MAS (or Touchalytics).
2. We randomly select ten other users from both datasets to provide data for the negative class.
3. For data fusion, we use each IDNet motion data block as a basis and extract the BB-MAS (or Touchalytics) touch events in the same duration.
4. We adjust the timestamps of the touch events to align them to those of the motion data.

We acknowledge the limitation that merged user behavioral data may not be realistic. However, the fused dataset is only used to test the accuracy gain of different fusing methods for the multi-modal scenario where an authenticator is failing. It also demonstrates that SHRIMPS support various public datasets.

5.6 Use Case 1: Fusion Method Comparison

A multi-user IA system is expected to identify each legitimate user and reject imposters under different settings. SHRIMPS enables IA researchers to compare different multi-user IA schemes and choose the best one in terms of accuracy and detection latency. Specifically, a multi-user system should detect user switches, which are common in household sharing activities [7, 124]. It is also possible that an attacker grabs the device from the owner, causing a sudden user change. In this use case, we address the following questions:

1. How does adopting multiple modalities benefit multi-user IA compared to single modality solutions?
2. What score fusion method provides the highest overall accuracy considering false acceptance rate and false rejection rate?
3. Is it necessary to set the maximum user size for a multi-user IA system?
4. How fast and accurately can a multi-user IA system capture an uninformed user switch during a shared session?

We first explain what fusion methods we add to SHRIMPS. Then we describe the evaluation tasks that we execute in the framework.

5.6.1 Fusion Methods

We test different score fusion methods and compared them to single modalities to examine how they balance FAR, FRR, and FIR. The two baseline methods include single-modal gait-based IA and single-modal touch-based IA. The most widely used score fusion method is average-based fusion. Moreover, weighted average methods also take the authenticator's performance into consideration. To compare D-S theory based methods to the average-based methods, we apply the per-user AUCs and EERs as the weights for the average-based methods.

CORMORANT [74] proposed a Kalman filter based score fusion method that is resistant to the noise of detection. We extend it into a multi-user fusion method by applying Kalman filter to multi-user scores for each user with the following settings: 1) Measurement uncertainty R is determined by the per-user EER, 2) Process uncertainty $Q = 0.25$: a large Q makes the estimated score emphasize on new scores [74] (the selection of Q is explained in Appendix A.1). In summary, we compared the following methods:

- **Touch.** Applying the touch authenticator only.
- **Gait.** Applying the gait authenticator only.
- **Mean.** Calculating the average score of all authenticators.
- **Mean-AUC.** Calculating the weighted average score using AUC as the factor.
- **Mean-EER.** Calculating the weighted average score using 1-EER as the factor.
- **Kalman.** Applying Kalman Filter based score fusion.
- **DS-AUC.** Applying multi-user D-S theory based score fusion with the AUC-based uncertainty function (Eq. 5.1)
- **DS-EER.** Applying multi-user D-S theory based score fusion with the EER-based uncertainty function (Eq. 5.2)

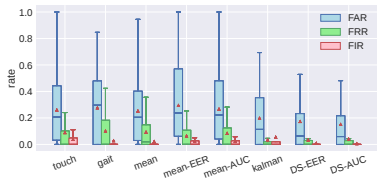
5.6.2 Evaluation Tasks

We design two groups of evaluation tasks to address the above questions. Given the limited data amount we use two fused datasets, IDNet+BB-MAS and IDNet+Touchalytics, only for the first group and adopt different settings for this dataset than for the HMOG dataset.

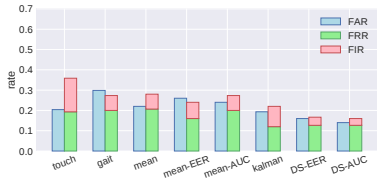
Group 1 (Accuracy evaluation) tests if the system can reject an attacker in a lunchtime attack and verify the identity of a legitimate user. The accuracy evaluation adopts a balanced static setting: there are equal numbers of legitimate users n_v and attackers n_a in each setup; each legitimate user has a fixed number of data blocks for initial enrollment and contributes to one fixed-length block for testing; therefore, there are $n_v + n_a$ blocks for each setup; we set an external signal between blocks to reset the authentication status. For HMOG, we used six enrollment data blocks for each user and set the testing block length as three minutes; we tried four different actor sizes, $n_v = n_a = 3, 5, 7, 10$ and tested 50 different actor combinations for each actor size. For IDNet+BB-MAS and IDNet+Touchalytics, we used two enrollment data blocks for each user and set the testing block length as two minutes; we tested $n_v = n_a = 3$ for 25 different actor combinations. In addition, given the length of the IDNet motion data blocks is much shorter than HMOG, we also reduce the segment size and the detection interval of the gait authenticator to 512 and 256 samples, respectively.

Group 2 (User switch evaluation) tests how each method detects user switches from a legitimate user to another legitimate user or an attacker in real-time. There is no external signal that informs the system of user switches. We assume that there are three legitimate users and three attackers in the task, where each legitimate user has six data blocks for initial enrollment. We composed three device sharing events and three attack events in the following storyboard:

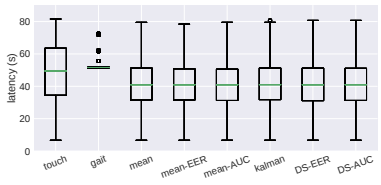
1. u_0's block, u_1's block, [external signal];
2. u_0's block, u_2's block, [external signal];
3. u_1's block, u_2's block, [external signal];
4. u_0's block, a_0's block, [external signal];
5. u_1's block, a_1's block, [external signal];
6. u_2's block, a_2's block, [external signal],



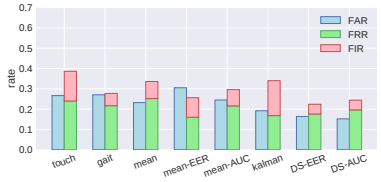
(a) $n_v = n_a = 3$ decision-level accuracy



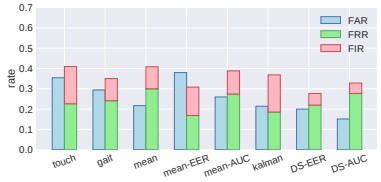
(b) $n_v = n_a = 3$ session-level accuracy



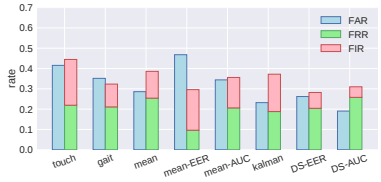
(c) $n_v = n_a = 3$ detection latency



(d) $n_v = n_a = 5$ session-level accuracy



(e) $n_v = n_a = 7$ session-level accuracy



(f) $n_v = n_a = 10$ session-level accuracy

Figure 5.3: Accuracy evaluation on HMOG. For each setting, the number of legitimate users (n_v) and attackers (n_a) are equal

	touch	gait	mean	mean-EER	mean-AUC	kalman	DS-EER	DS-AUC
GC-FAR	0.70	0.70	0.70	0.66	0.69	0.74	0.78	0.81
GC-FRR	0.89	0.86	0.86	0.89	0.87	0.93	0.93	0.92
GC-FIR	0.91	0.94	0.95	0.95	0.95	0.94	0.98	0.98

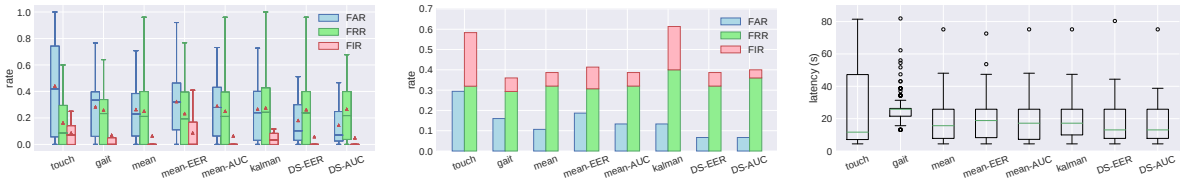
Table 5.1: Gini Coefficient of FAR, FRR, and FRR at $n_v = n_a = 3$.

where $a_{0,1,2}$ are three different attackers (i.e., u_{-1}). As shown in the storyboard, each event (i.e., session) consists of two blocks from two different actors without any external signal in between to describe an uninformed user switch. The external signals in the storyboard only mark the end of each session.

5.6.3 Result Analysis

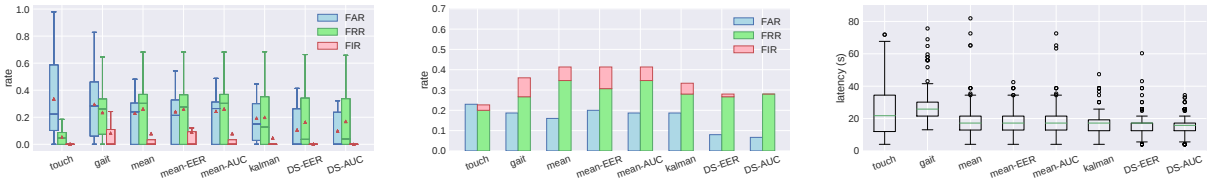
We provide the result analysis as follows:

Group 1. Figure 5.3a shows the results of the first group of evaluation tasks on HMOG, which includes the decision-level accuracy distributions of all eight methods at $n_v = n_a = 3$. The D-S theory based methods have lowest FAR, FRR and FIR, which means they can ef-



(a) decision-level accuracy (b) session-level accuracy (c) detection latency

Figure 5.4: Accuracy evaluation on IDNet+BB-MAS.



(a) decision-level accuracy (b) session-level accuracy (c) detection latency

Figure 5.5: Accuracy evaluation on IDNet+Touchalytics.

fectively reject attackers with less chance to falsely reject a legitimate user. Table 5.1 shows the GC of each error type (the error distribution curves are presented in Appendix A.2). High GCs on D-S theory based solutions imply that most errors were contributed by fewer users after applying the D-S theory based solution. The performance of score fusion methods is bounded by the fused modalities — The error rate concentrated on the users for whom both modalities have low accuracy. Session-level comparisons are in Figure 5.3b. We find that the impact of FRs and FIs is magnified at session-level. Specifically, the touch-based method has a significantly high FIR. Among all methods, D-S theory based methods achieve the lowest overall false detection rate: FRR (0.13) and FIR (0.03), which means about 84% of the legitimate users' blocks are error-free. Although Kalman filter based fusion also achieves a low FRR (0.12), its FIR is significantly higher (0.10). We measure the latency as shown in Figure 5.3c. Detection latency is determined by the adopted IA mechanisms: the touch authenticator relies on a user's interaction with the screen, and the gait authenticator using the default settings [196] performs authentication at a low frequency. Both take much time to collect sufficient data for making decisions. Since all multi-modal methods are implemented in SHRIMPS with the same configuration, there is no significant latency difference. Compared to single modalities, they improve the latency because they receive results from both modalities to make decisions earlier.

Figure 5.4 shows the results on IDNet+BBMAS. This evaluation task aims to compare

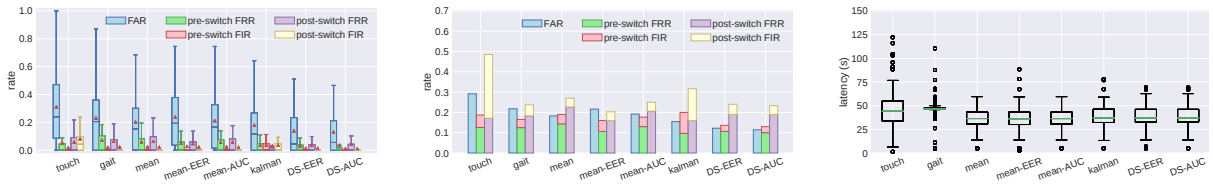
the accuracy gain of different fusion methods when one modality performs significantly worse than the other. Due to the task setup, different swipe types (i.e., vertical swipes on the left and right parts of a screen and horizontal swipes on the bottom) are not evenly distributed in the time series. Consequently, patterns for some swipe types are not well learned by the touch based IA, which leads to poor accuracy. From Figure 5.4a, we can see that the FAR of the touch authenticator was very high. However, the D-S theory based solutions still significantly reduced the FAR (the session-level FARs of DS-AUC and DS-EER are 0.07) compared to the other approaches. Besides, they can also improve the FIR. We can draw the same conclusion from the session-level results in Figure 5.4b. For detection latency, we see the same trend for IDNet+BB-MAS as for HMOG. However, due to the shorter detection interval and earlier touch events, the overall latency for IDNet+BB-MAS is much shorter than for HMOG.

Figure 5.5 shows the results in IDNet+Touchalytics. The decision-level FAR and FRR of the touch-based IA are significantly lower than IDNet+BB-MAS. At the session level, the FAR and FRR+FIR of the touch-based IA are more balanced. After applying the score-level fusion, we can see lower FAR, while D-S theory based methods can achieve the lowest session-level FAR (DS-AUC:0.07). However, due to the gait-based IA, all fusion methods have a higher FRR than the touch-based IA. However, DS-AUC can still achieve the lowest FRR+FIR among all methods.

Based on the above results, we can answer questions 1 and 2: 1) Multi-modal methods achieve significantly better accuracy and balancing FAR, FRR and FIR than single modalities, and 2) Among the tested fusion methods, D-S theory based methods have the lowest false detection rate.

When the legitimate user size is increased to 5, 7, and 10, we observe an increase in false decisions for all methods in Figs. 5.3d, 5.3e, and 5.3f. In particular, the FAR rises significantly, which implies that the ability of detecting attackers is weakened when classifying more classes. Nevertheless, DS-AUC can still well balance FAR and FRR/FIR, (FAR: 0.19, FRR: 0.26, FIR: 0.05, when u_v is 10). From the result, we can answer the third question: it is necessary to control the user size of a system to ensure high overall accuracy. IA researchers need to specify a threshold for the system accuracy and test different user sizes to determine the system capacity.

Group 2. Figure 5.6a shows the results of the second group of tasks: the decision-level results for both pre-switch blocks and post-switch blocks are similar to the accuracy evaluation results. For attack events, DS-EER has the lowest FAR at both decision-level and session-level. However, for sharing activities, we can observe an increase in FIR and FRR for the post-switch blocks for all methods at session-level because of the



(a) decision-level accuracy (b) session-level accuracy (c) detection latency

Figure 5.6: User switch evaluation results

detection latency — although the current user has changed to a different legitimate user, the authentication system still has no sufficient confidence in identifying this user. According to Figure 5.6b, DS-AUC and mean-EER still have better FRRs (0.19, 0.16) and FIRs (0.04, 0.04) compared to the other methods. However, the FAR of mean-EER (0.23) is much higher than that of DS-AUC (0.12). The high FIR (0.16) of Kalman filter based fusion shows that it is not a good option for handling user switches because its noise resistance makes it slow in handle sudden score changes. The detection latency results in Figure 5.6c are similar to the first group of tasks. DS-AUC can provide a low and stable detection latency (mean=37.7s, std=10.8).

The results have shown that D-S Theory based fusion methods can capture user switches during shared sessions with balancing FAR, FRR, and FIR compared to the other methods, which answers the fourth question. In addition, the results also imply the importance of the external signals. If a signal, such as Android's Screen Pinning signal [78], may imply a user switch event, the system can then determine the end of a user's device use and reset the authentication status. Then, a user switch task can be simplified as an accuracy evaluation task, where external signals assist in data segmentation to improve accuracy (see the second use case).

5.7 Use Case 2: Multi-user Model Updating

Compared to the balanced and static settings adopted in the first use case, the second use case considers more factors: First, given that the owner usually spends more time with the device and contributes more training data than a secondary user, we test how the system handles imbalanced user data. Second, as new incoming data is used for model updating and new users are added into the system, the detection accuracy of the system may change over time. Third, user feedback towards false decisions may influence the identification

and auto labeling processes. In addition, a user's lifting and putting down the device and other events may indicate the starting and the end of device usage and can be used to segment the data, which are considered as external signals to the multi-user IA system. The evaluation tasks should address the following questions:

1. What is the accuracy difference of the system identifying the owner and the secondary users?
2. How does new incoming data affect the accuracy of the system identifying different users?
3. How does external signals and user feedback benefit data segmenting and labeling in term of the overall accuracy?

5.7.1 Comparison Strategies

From the first use case, we conclude that the D-S Theory based methods can accurately detect attackers and identify legitimate users. Thus, we adopt DS-EER for score fusion. To address the above questions, we compare three strategies:

1. *baseline*: the system only supports user enrollment and does not learn from historical data (i.e., no model retraining after each part);
2. *uninformed*: the system makes decisions and performs auto labeling based on identification results and user feedback, and ignores external signals;
3. *informed*: the system additionally uses external signals for detecting a user switch and auto labeling.

5.7.2 Evaluation Tasks

For setup, there are three legitimate users: the owner u_0 and the secondary user u_1 have already enrolled in the system, and a new user u_2 will enroll in the system during the task. For initial enrollment, u_0 has three blocks, and u_1 only has one. The length of each segment is randomly sampled, ranging from two to five minutes based on the high variance reported by Harbach et al. [68]. To show the accuracy change over time, we split the task into three parts at each model re-training and design the following storyboard:

	user	Part 1			Part 2			Part 3		
		FRR	FIR	FR	FRR	FIR	FR	FRR	FIR	FR
Baseline	u_0	0.25	0	0.25	0.43	0.05	0.48	0.48	0.03	0.51
	u_1	0.47	0	0.47	0.5	0.05	0.55	0.53	0.02	0.55
	u_2	-	-	-	0.43	0.03	0.46	0.53	0.02	0.55
Uninformed	u_0	0.21	0.01	0.22	0.38	0.11	0.49	0.37	0.10	0.47
	u_1	0.43	0	0.43	0.37	0.03	0.40	0.43	0.07	0.50
	u_2	-	-	-	0.38	0.03	0.41	0.29	0.04	0.33
Informed	u_0	0.23	0.01	0.24	0.43	0.04	0.47	0.33	0.03	0.36
	u_1	0.38	0.01	0.39	0.27	0.02	0.29	0.26	0	0.26
	u_2	-	-	-	0.27	0.02	0.29	0.25	0.01	0.26

Table 5.2: Per-part results for use case 2. Three legitimate users: u_0: primary user; u_1: secondary user; u_2: new legitimate user. False decision rate (FR) is the sum of FRR and FIR.

1. (2 blocks): u_0's block, [external signal], u_1's block, [external signal], model retraining;
2. (3 blocks): u_2's enrollment, [external signal], u_0's block, [external signal], u_1's block, [external signal], u_2's block, [external signal], model retraining;
3. (3 blocks): u_0's block, [external signal], u_1's block, [external signal], u_2's block, [external signal].

External signals are only used in the informed strategy, while user feedback is used in both the informed and the uninformed strategies: External signals indicate device handoff where a legitimate user passes the device to another. User feedback indicates the current user has successfully passed the explicit authentication, which means the IA system made a false rejection. As a result, the evaluation framework will notify the system of the false decision. Model retraining is not applied to the baseline strategy. For each part, we measure the accuracy over all the blocks within. We repeat the task with 150 different actor combinations.

5.7.3 Result Analysis

Table 5.2 shows the per-user results for each part. In Part 1, we can observe that the system had a lower false detection rate at identifying the owner than a secondary user when there

is not much training data. Such difference becomes smaller when more training data is available for secondary users due to the data balancing strategy, which answers the first question.

After u_2's enrollment at the start of Part 2, the multi-user IA system updated all IA models. By the end of Part 2, the system experienced significant accuracy degradation in identifying u_0 compared to Part 1. However, due to score fusion and data balancing, the accuracy of identifying the new user is close to identifying u_1. At the end of Part 2, the system retained all models with new data collected in Part 2. In Part 3, the false decision rate dropped. Compared to the baseline, the FRR and FIR of the informed strategy were lower for all users. The results have addressed the second question: model updating can help improve the cross-session accuracy significantly.

To answer the third question, we compare the uninformed and informed strategies across all parts. The results have shown that external signals can further improve accuracy because 1) they enabled the system to reset the authentication status at a user switch to avoid false identification, and 2) they provided precise data segmentation, which makes the system correctly label more behavioral features. Despite the benefits of improving the accuracy, IA researchers also need to consider the usability of the system. For example, frequently asking for a user's feedback makes the system hard to use. With SHRIMPS, IA researchers can observe the frequency of the external signals and optimize the workflow by modifying the auto labeling and model update mechanisms.

5.8 Discussion

Limitations. We list the following limitations of SHRIMPS or sample use cases: 1) The design of simulation tasks is restricted by the dataset. For example, for HMOG, we limit the length of use case 2 to three parts to satisfy the cross-session requirement, which leads to high error rate for all strategies. 2) Although SHRIMPS supports simulated user feedback, there is still a gap between simulation and user studies in usability evaluation. Nevertheless, SHRIMPS can be used for tuning and evaluating a multi-user IA system before user studies. 3) Since SHRIMPS is a simulation framework, it is not for implementing and developing a deployable multi-user IA system on smart devices. However, it can be easily connected with the real systems for parameter tuning.

Applications. We present two use cases to exemplify how SHRIMPS helps IA researchers and developers design and evaluate multi-user IA schemes. We note that SHRIMPS can be applied in diverse scenarios. For example, it is feasible to use the simulation environment

to generate a long simulation task consisting of random sensor data blocks and random external signals to test the robustness of a multi-user IA scheme. Besides, IA researchers can explore how much training data is required for different user types (e.g., owner and secondary users) to help balance the per-user accuracy by modifying the training set generation module.

Multi-user concurrent usage. We assume only one user operating the device at the same time. Matthews et al. [124] listed broadcasting as a type of device sharing, where multiple people are co-using a single device simultaneously. Recognizing all present legitimate users can be regarded as a multi-label classification problem. However, if the system is always assuming the device is under concurrent usage and performing multi-label classification, its accuracy is very likely to suffer given the problem complexity. Thus, we need a certain external signal indicating the concurrent usage context and then trigger multi-label classification.

Contextual information. According to the user switch simulation tasks, an IA system may falsely reject a legitimate user if it is uninformed. However, if it knows the context of user switch through external signals, the problem can be simplified as a general identification task. Existing studies [74, 129] also use contextual information to adapt IA systems for better accuracy or less battery consumption. A future avenue is to embed contextual information into SHRIMPS and establish connections between context and authenticators.

User removal. The user removal problem involves three steps: 1) excluding the target user from the legitimate user set, 2) purging the training data and models of the target user, 3) eliminating the impact of the target user's data on other IA models. SRHIMPS adopts full retraining to achieve these goals, which is time-consuming. A potential solution is to apply machine unlearning techniques [20, 133] to let IA models forget the removed user's training data with balancing training time, space, and accuracy.

5.9 Conclusion

In this chapter, we proposed a multi-user, multi-modal IA solution to provide continuous identification and authentication for shared mobile devices. We applied the Dempster-Shafer theory based score fusion methods to combine multiple modalities. To support IA researchers, our proposed SHRIMPS provides an architecture of a multi-user IA system and an evaluation framework for designing and evaluating multi-user IA schemes. We presented two use cases that use SHRIMPS to design multi-user IA schemes with touch-based and gait-based IA mechanisms and address practical design questions. For example,

we composed a storyboard using the HMOG dataset to test how a multi-user IA captured sudden user change during a session, which was not covered by the single-user scenario. The evaluation results of the use cases also showed the effectiveness of our proposed multi-user IA solution: the D-S theory based fusion methods can significantly reduce false detections compared to existing score fusion strategies. Besides, we also tested how our solution automatically segmented and labeled new incoming data and updated the IA models with new incoming users. With the involvement of external signals and user feedback, our solution can ensure the cross-session performance of the multi-user IA system.

In this chapter, our multi-user IA solution applies both algorithmic and systematic adaptations to shared mobile devices. In practice, an authentication system many also require multiple adaptations to handle various scenarios. They can be triggered by different contextual factors and apply different levels of changes to authentication systems. Thus, we need to build an adaptation model to organize various adaptations to make them co-exist in the system, which will be covered in Chapter 6.

Chapter 6

Multi-stage Adaptive Authentication and Access Control

6.1 Introduction

This chapter aims to address the fourth research objective of modeling adaptive authentication systems. The key research question is how to organize adaptations of authentication and access control in one adaptation model and enable the deployment of adaptive authentication to handle various risks. Upon addressing this research question, we implement a framework to help developers enable adaptive authentication and access control for their systems and apps.

6.1.1 Limitations of Existing Adaptive Authentication Solutions

Existing adaptive authentication solutions have the following limitations that could lead to severe problems and restrict their applications in various scenarios: **First**, most existing adaptive authentication solutions adopt a simple *single-stage* adaptation structure that can be summarized as multiple "if *condition* then *adaptation*" statements. This structure may introduce conflicts and vulnerabilities. Different contextual factors may hold simultaneously and drive opposite adaptation outcomes. For example, Android Smart Lock has conflicting interoperation between contextual factors [99]: Leaving a trusted location is expected to trigger the automatic lock, whereas a device's connection to a trusted device may make it remain unlocked. **Second,** some adaptive authentication schemes [70,126,129] use

contextual factors to *unlock* a device, which may allow an attacker to bypass EA mechanisms. For example, Smart Lock before Android 10 [80] allowed trusted places or devices to unlock a locked device, which may be exploited by social insiders [126]. As of Android 10, Smart Lock can be used only to extend access to an unlocked device. **Third**, most adaptive authentication systems are difficult to extend or reuse [10]. However, an app may employ multiple authentication mechanisms and apply different adaptations for each. For example, a corporate app for Bring-Your-Own-Devices (BYODs) may adapt IA mechanisms to their data availability (e.g., activate gait-based IA only when a user is on foot) and tune their sensitivity based on the current location (e.g., onsite or offsite). To the best of our knowledge, there is no existing framework that supports various adaptations.

6.1.2 Challenges

The main challenges involve the following aspects:

1. Authentication and access control adaptations are driven by various factors and affect different components of an authentication system. Therefore, organizing them in one model is challenging.

2. An adaptation model should be extensible: it should allow adding adaptation policies for a new scenario with minimal conflicts with the existing policies.

3. An adaptive authentication system should accommodate various context factors and authentication mechanisms. Adding, removing, and modifying a component should not affect other components or adaptation logic.

4. Integration of the adaptation framework in mobile apps and systems should introduce minimal change to the existing code.

6.1.3 Contributions

We present a multi-stage risk-aware adaptive authentication and access control (MRAAC) framework. MRAAC targets three types of stakeholders: Designers of authentication systems for mobile operating systems, developers of mobile device management (MDM) solutions, and developers of mobile apps with high security requirements. MRAAC organizes authentication and access control adaptations in three levels: the top level determines when to explicitly authenticate a user; the second level handles the adaptations driven by changes

in security contexts to choose an appropriate set of authenticators for the current risk type and level; the third level focuses on the adaptation of individual authenticators to ensure usability. The multi-level structure ensures that the system prioritizes security-related adaptations to prevent unauthorized access. MRAAC adopts a graph-based organization where adaptation policies are organized in multiple stages and take effect only at their affiliated stages to avoid possible conflicts with policies in other stages. The multi-stage design also enables the statefulness of MRAAC, which allows the system to progressively adjust its behaviors based on the feedback of a previous adaptation.

The contributions are fourfold:

- An opensource, multi-stage framework for adaptive authentication and access control for Android. Our approach supports automatic generation of multi-stage adaptation models and enables the design of complex and stateful adaptation schemes based on contextual factors, resource sensitivity, and authentication results.
- Two libraries for developers of authentication systems and sensitive apps to enable adaptive authentication and granular in-app access control with low development overhead.
- Three use cases, Smarter Lock, guest-aware continuous authentication and BYOD corporate app, to demonstrate how to design multi-stage adaptation model for different use cases.
- An evaluation of MRAAC based on two implemented MRAAC-enabled apps. The performance evaluation results show that MRAAC introduces low overhead and battery consumption. A use case simulation on the public HMOG dataset [159] shows that the multi-stage design can balance the false rejection rate and detection latency of continuous authentication.

6.2 Motivation

This section presents two motivating examples to show the limitations of existing adaptive authentication systems and then summarizes the requirements of a desired adaptive authentication and access control framework.

6.2.1 Motivating Examples

Smart Lock. To reduce unnecessary EA, Android Smart Lock [82] leverages three contextual factors to determine whether to ask for EA when a user starts a new session: 1) *On-body Detection (BODY):* whether the device is with a user in movements, 2) *Trusted Places (PLACE):* whether the current location is trusted, 3) *Trusted Devices (DEVICE):* whether any trusted devices is connected with the device. Smart Lock (before Android 10) uses the following five adaptation policies:

1. If the device is not on body (`BODY == false`), then activate EA for a new session.
2. If the current location is trusted (`PLACE == true`), then de-activate EA for a new session.
3. If the current location is not trusted (`PLACE == false`), then activate EA for a new session.
4. If any trusted device is connected (`DEVICE == true`), then de-activate EA for a new session.
5. If no trusted device is connected (`DEVICE == false`), then activate EA for a new session.

Note that "activating EA" does not mean an immediate device locking, and the user can still use the device for the current session.

We observe several limitations: (P1.1) *Policy conflicts.* If the device is connected to a trusted device in an untrusted location, Policies 3 and 4 are satisfied simultaneously. But the adaptation outcomes of the two policies are contradictory. The system has to prioritize one outcome over another or set an explicit policy for the situations where both conditions hold. (P1.2) *Misusing contextual factors for unlocking.* The contextual factors in Policies 2 and 4 result in the de-activation of EA. However, contextual factors do not provide as strong evidence as EA mechanisms that the current user is legitimate. As a result, using contextual factors to unlock a locked device makes it easier for an attacker to bypass authentication. Specifically, it introduces a higher risk of social insider attacks [126] where an attacker is in the trusted environment set by the owner and unlocks a locked device without EA. Therefore, Android 10 and later changed the behavior of Smart Lock to only extending unlock, which prevents an attacker from bypassing the EA of a locked device. However, social insiders can still take advantage of the extended unlock to access the device. For example, the owner sets the workplace as trusted to keep the smartphone unlocked

at work. A co-worker can wait for the owner's temporary absence (e.g., coffee break) and access the unlocked device during the extended unlock interval. (P1.3) *Stateless.* The above policies take effect regardless of the device state as long as their conditions are satisfied. It is hard to implement complex adaptations such as the "low watermark" adaptation [129]: If the three factors are all negative at the same time, the device is considered in the insecure state. EA should be required even if either Policy 2 or Policy 4 is satisfied. The insecure state ends only after the user has passed EA.

§ 6.5 presents Smarter Lock to address the above problems and provides an IA-enabled option to incorporate continuous authentication (CA) into Smarter Lock. To further show the extensibility of our solution, we implement the guest-aware CA use case that adapts authentication and access control to guest access, which is a special scenario derived from the social insiders.

BYOD. Itus [92] is a general IA framework for BYOD apps that supports various behavioral biometrics based IA mechanisms (e.g., Touchalytics [52], SilentSense [17]). Assume an enterprise E-mail client app is using Itus to enable IA. It uses a location contextual factor to determine if the device is within the company or not, and extracts behavioral biometrics from body movements and touch events. To coordinate them, the app adopts the following adaptations:

1. If a mismatch in behavioral biometrics (i.e., a negative IA result) is detected, then lock the device and activate EA.
2. If the current location is not within the company (i.e., offsite), then activate IA.
3. If the current location is within the company (i.e., onsite), then de-activate IA.

The adaptations supported by Itus mainly include two types: using IA mechanisms to determine when to activate EA and using context factors to determine when to activate a certain IA mechanism. However, they are insufficient from the following aspects: (P2.1) *Conditions.* An adaptation can be driven by multiple reasons. As location implies the risk level of the current context (i.e., Policies 2 and 3), data availability and resource sensitivity are other possible conditions to determine whether to activate an IA mechanism. For example, if there is no significant movement, de-activate body movement based IA and activate keystrokes or touch events based IA instead. Accessing sensitive resources requires stricter IA mechanisms to prevent unauthorized access. (P2.2) *Adaptation outcomes.* The outcomes of the above policies are all about activation and de-activation. In practice, it is also feasible to tune the parameters of an authentication mechanism and change access control policies. For example, IA mechanisms for the offsite scenario can switch

to a more sensitive configuration than the onsite scenario to defend against higher risk of unauthorized access. The system can also block access to specific resources when a user's identity is uncertain. (P2.3) *False rejection.* According to Policy 1, a negative IA result immediately locks a device and requires EA. False rejection will make an authentication system less usable. It is preferrable to have an intermediate step before locking a device in order to mitigate false rejection while securing sensitive resources.

In § 6.5.3, we present the BYOD use case that provides a comprehensive solution to enable adaptive authentication and access control for a real E-mail client app.

6.2.2 Summary

The motivating examples have shown that adaptations can be driven by various factors and applied to multiple authentication mechanisms. Multiple adaptations can co-exist in the same system to handle a variety of scenarios. Unorganized adaptations can hardly fulfill the security and usability requirements. As reviewed in § 2.5, no existing studies have proposed an adaptation model to organize various adaptations of authentication and access control. To address the above issues and limitations, our adaptation model is expected to:

1. Categorize adaptations based on their conditions and organize them in a multi-level structure to reduce possible conflicts (P1.1);
2. Model risks from the perspectives of user authenticity, resource sensitivity, and threat to describe the system states (P1.3) and cover various conditions that imply risk changes (P2.1);
3. Regulate adaptations so that attackers cannot exploit contextual factors to bypass the authentication system (P1.2);
4. Enable progressive adaptations that adjust authentication and access control based on the feedback of a previous adaptation (P2.2) instead of directly imposing EA to mitigate false rejections (P2.3).

6.2.3 Threat Model

User authentication and access control defend against unauthorized access. We trust the device owner or the primary user (for corporate-owned devices). Attackers are other people

who are physically near the device and can physically access it. Their goal is to access (sensitive) resources on the device.

We follow the general threat model in § 2.6 and focus on nearby strangers (i.e., opportunistic attackers) and social insiders (i.e., informed attackers). Social insiders may share a trusted environment (based on a device owner's trust perception) with the owner (e.g., a home) and take advantage of contextual factors to bypass the authentication system. A special type of social insiders are guest users (i.e., sharees in temporary device sharing), who are allowed to access non-sensitive resources, but not sensitive resources.

Since we provide an Android implementation of the adaptive authentication and access control framework (see § 6.4), we make the same assumptions as mentioned in § 2.6. Moreover, the framework allows communications with apps, it is possible for them to acquire sensitive resources or send false information. We discuss such attacks in § 6.4.2 and § 6.7.

6.3 Modeling

In this section, we give the definition of a multi-stage adaptation model and the model construction process.

6.3.1 Definition

Three levels of adaptive authentication. Adaptive authentication can be divided into three levels based on the adaptation reasons (i.e., what leads to changes in an authentication system). The top level of adaptive authentication is to address *when to authenticate a user*. Both EA and IA assist an adaptive authentication system: EA provides mandatory identity verification when a user logs into the system, and IA continuously verifies a user's identity throughout a user's access and provides weak authentication for low risk levels. Since IA rejection causes EA [52,92], switching between EA and IA is an adaptation process based on the user's identity.

The other two levels address how to authenticate a user based on the *security* context and the *usability* context, respectively. Changes in the security context imply changes to the risk of attacks and determine the requirements of the authentication mechanisms to handle the risk. For example, a higher risk (e.g., accessing sensitive resources) calls for a stricter set of IA mechanisms that are more sensitive to imposters. In comparison, changes

	Default				Capped			
auth lvl	IA+	IA-	EA+	EA-	IA+	IA-	EA+	EA-
a_0	-	-	a_2	a_0	-	-	a_2	a_0
a_1	a_2	a_0	-	-	a_1	a_0	-	-
a_2	a_2	a_1	-	-	a_2	a_1	-	-

Table 6.1: Two examples of authentication transition functions. Default: a negative IA result decreases the authentication level by one, while an EA acceptance raises it to the maximum. Capped: a positive IA result cannot raise the authentication level from a_1 to a_2.

in the usability context are related to the availability and performance of the authentication mechanisms, which help determine the (de-)activation and tuning of these mechanisms. For example, low ambient brightness may result in choosing an authentication mechanism other than facial recognition. Thus, based on the scale of impact from the adaptation outcome, we define the security-driven (or risk-driven) adaptation as high-level adaptation and the usability-driven adaptation as low-level adaptation.

Risk. We describe the risk from three factors: the *risk type* of the scenario, the *authentication level* of the user, and the *sensitivity level* of the current resource. Risk types are characterized by contextual factors, such as locations and activities, and have distinct adaptation requirements. Authentication levels indicate the confidence of identifying a user as the owner. A low authentication level calls for stricter IA mechanisms or even EA. Adopting multiple authentication levels gives flexibility when processing an IA rejection. For example, the first IA rejection may trigger further verification and restrict access to sensitive resources, instead of a direct lockout, to mitigate the impact of false rejections. Sensitivity levels and authentication levels determine whether a user can access a resource. If the authentication level fails to match the sensitivity level of the target resource, the system rejects the access and asks for EA. Besides, an authentication system should impose stricter authentication mechanisms for access to resources with higher sensitivity levels.

6.3.2 Multi-stage Adaptation Model

Risk Model. We define the three factors as: the risk type set $R = \{r_0, r_1, \ldots, r_{n_R-1}\}$, the authentication level set $A = \{a_0, a_1, \ldots, a_{n_A-1}\}$, and the sensitivity level set $C = \{c_0, c_1, \ldots, c_{n_C-1}\}$, where n_R, n_A, and n_C are the sizes of the three sets. The risk type is a categorical variable that characterizes the variety of attack risks by contextual factors. An example are risk types based on location: if the current location is *home* or *workplace*

then apply different adaptation strategies for each of them. We reserve r_0 as the complement of all defined risk types, which represents the risk of general unauthorized attacks. Authentication levels and sensitivity levels can be expressed as two ascending sequences where $a_0 < a_1 < \ldots < a_{n_A-1}$ and $c_0 < c_1 < \ldots < c_{n_C-1}$. We reserve a_0 and c_0 for the Locked state, which represent the lowest authentication level and the lowest sensitivity level, respectively.

For each risk type, we use two functions to describe the authentication and access control behaviors: 1) Authentication results lead to changes in authentication levels, and the mapping between them is a function $\lambda_r : A \times \Gamma \to A$, where Γ is the authentication result set (e.g., IA acceptance/rejection). Table 6.1 shows two examples of authentication transition functions. 2) Mandatory access control is a function $\mu_r : A \times C \to \{\text{True}, \text{False}\}$, which determines if the user's authentication level is insufficient to access resources at a specific sensitivity level. If the function returns False, the user should be rejected and may need further authentication (usually EA) for a higher authentication level. The default access control function is $\mu_r(a_i, c_j) = \mathbb{1}_{i \geq j}$.

To describe transitions among risk types, we use a function $\kappa : R \times \Phi \to R$, where Φ is the set of transition signals, including contextual factors and authentication results. For example, if the location sensors detect that the device is not at a safe place, the system changes the risk type for adaptation to public locations.

Multi-stage model. The design of an adaptation model follows the three adaptation levels introduced in § 6.3.1. On the top adaptation level, an adaptation model can be divided into the unlocked and locked states of a binary access control model [71], where IA is activated in the unlocked state and EA is activated in the locked state. For the high-level adaptation, the system dynamically adjusts its behaviors in response to the input signals that change the current authentication level, sensitivity level, and risk type. We use a finite-state machine (FSM) based multi-stage model to describe the adaptation process. Each stage (i.e., state) hosts a set of adaptation policies to apply the low-level adaptation.

The multi-stage model M is a quadruple (Σ, S, s_0, δ): Σ is the set of all input signals for high-level adaptation, S is the stage set: each stage $s \in S$ is a combination of $(r, a, c) \in R \times A \times C$; s_0 is the initial stage for the unlocked state, which is by default (r_0, a_{n_A-1}, c_1) (i.e., general risk type, highest authentication level, low sensitivity level); $\delta : S \times \Sigma \to S$ is the transition function that determines the next stage based on an input $x \in \Sigma$ and the current stage s. We reserve the locked stage l for EA (e.g., Android Keyguard). The stage transition from l to s_0 is triggered by positive EA results (denoted by e^+): $\delta(l, e^+) = s_0$, while negative EA results (denoted by e^-) trigger a self-transition: $\delta(l, e^-) = l$.

We define that any input signal $x \in \Sigma$ may change *only one* of the three factors (r, a, c)

at a time. An event that causes changes in the security context can be broken down into a series of input signals in chronological order. For example, when the system detects a guest access event, it issues an authentication signal to lower the authentication level and then produces a context signal to change the risk type. If $\delta(s, x)$ is not defined, the system processes it as an exception (i.e., x is an invalid input at stage s) and moves to the locked stage.

To build a multi-stage model, developers first need to prepare a threat model specifying the following information: the risk type set R, the authentication level set A, the sensitivity set C, the authentication result set Γ, the context signal set Φ, the risk transition function κ and the authentication and access control functions under each risk: $\{\lambda_r\}_{r\in R}$ and $\{\mu_r\}_{r\in R}$, respectively. Then, a multi-stage model is automatically generated from a risk model according to the following steps:

1. Construct the input set $\Sigma = \Gamma \bigcup C \bigcup \Phi$ to include all valid inputs that change any of (r, a, c). Note: a sensitivity level $c \in C$, as an input, means "the user attempts to or is currently using a resource of sensitivity level=c".

2. Obtain all valid combinations of three factors $\Omega \subset R \times A \times C$ where the authentication level is sufficient for accessing resources of the sensitivity level under the risk type: $\Omega = \bigcup_{r\in R}\{(r, a, c)|\mu_r(a, c) = \text{True}, \forall a \in A, c \in C\}$. Then, the stage set is$S = \Omega \bigcup \{l\}$.

3. Set the initial stage s_0 to (r_0, a_{n_A-1}, c_1) and all outgoing transitions from the Locked stage l (i.e., l to s_0 and self-transition) in δ.

4. Generate the stage transition function δ by traversing all stages $s \in \Omega$ and inputs $x \in \Sigma$:

$$\delta(s, x) = \begin{cases} (r, \lambda_r(a, x), c) & x \in \Gamma, \\ (r, a, x) & x \in C, \mu_r(c, x) = \text{True}, \\ (\kappa(r, x), a, c) & x \in \Phi, (\kappa(r, x), a, c) \in \Omega, \\ l & \text{otherwise.} \end{cases}$$

The generated multi-stage model defines the high-level adaptation behaviors of the system. The stage transition process enables the system to dynamically capture the risk change and change its adaptation scheme in response to the input signals. By traversing all valid combinations of the three risk factors and all acceptable inputs, the completeness of the model is ensured to cover all possible situations. The deterministic property of stage transition (i.e., given the current stage and the input, the outcome stage is deterministic) avoids possible ambiguity or conflicts. The stateful property provides the flexibility of

adopting different reactions to the same signal. For example, a negative IA result does not always trigger de-authentication if the user is accessing a non-sensitive resource.

In § 6.5, we present three use cases to show in detail how to design risk models and generate the multi-stage adaptation models accordingly.

6.3.3 Adaptation Policies

Adaptation policies determine how to change the authentication and access control mechanisms. A multi-stage model enables designing per-stage adaptation policies to handle the corresponding risk type and level of each stage. The adaptation policies are classified into two categories based on how they are triggered: *general adaptation policies* and *conditional adaptation policies*. General adaptation policies take effect once a stage transition happens. They determine the default authenticators and access control rules of a stage. For authentication adaptation, the system activates the authentication mechanisms listed in the general adaptation policies of the current stage. It also deactivates the ones that are activated in the previous stage but not listed in the current stage. For access control, the multi-stage adaptation model determines resource availability based on sensitivity levels and authentication levels. The access control policies within each stage can further adjust the availability of individual resources. Conditional adaptation policies are a set of "if-then" statements — once the conditions are satisfied, the corresponding adaptation takes effect. They mainly handle the low-level authentication adaptation process to determine when to activate and deactivate authentication mechanisms and how to tune the parameters in response to the usability context (e.g., if *no significant movement is detected*, then *deactivate gait authenticator*.)

The multi-stage adaptation model also supports the design of the adaptation policies. By comparing the risk type, authentication level, and sensitivity level, one can determine if a stage is "riskier" than its neighboring stages (i.e., stages connected to it by a transition edge). A stage at higher risk should have no less secure authentication and no more available resources than a stage at lower risk. Following the security metrics of biometric-based authentication mechanisms [81], it is feasible to compare the authentication mechanisms between two stages and determine if the proposed adaptation policies are reasonable.

6.3.4 Revocation of Adaptation Policies

In practice, some adaptations policies may no longer meet the security or privacy requirements, which calls for revocation. We use different revoking strategies for adaptations at

different levels. For the security level, revocation implies removing certain authentication levels, sensitivity levels, or risk types. It requires the re-generation of the multi-stage adaptation model. It is easy to obtain what stages are revoked by comparing the current and former models. All usability-level adaptation policies in the revoked stages automatically become invalid while other stages remain unchanged. For the usability level, the revocation of an adaptation policy is done by dropping the corresponding item from the adaptation scheme. It does not affect the security-level adaptations.

6.4 System design

6.4.1 Target Audience

We design MRAAC to provide a general adaptation framework for authentication and access control on mobile devices and apps. Our audiences are: 1) developers of mobile authentication systems, 2) security developers of mobile device management (MDM) and enterprise mobility management (EMM) solutions, and 3) developers of sensitive apps.

Authentication system developers can use MRAAC to manage the adaptation process to enhance their systems. As a mobile authentication system provides authentication APIs (e.g., Android's BiometricManager) for apps, MRAAC also enables risk awareness for in-app adaptation. We propose a "client-server" structure for MRAAC: the "server" is a centralized service that performs risk evaluation and provides system-wide adaptive authentication and app-level access control, and the "clients" are apps that receive signals from the "server" to enable risk-awareness and adjust the availability of in-app resources accordingly. Since the server handles most operations, clients can obtain the digested risk information without accessing sensitive permissions or performing expensive computations. Such a structure also works for MDM/EMM solutions. They require an endpoint management agent or are integrated with the operating system (e.g., Android Enterprise) to collect contextual information and enforce security policies, while a list of corporate apps are under its control.

Individual, sensitive apps may require custom adaptive authentication and access control without relying on an external service. For example, corporate apps for BYODs require dedicated adaptive authentication systems to manage a user's access based on their location and network. Thus, MRAAC needs to be integrated into these sensitive apps (called *host apps*) to enable risk awareness and adaptation management. Besides, MRAAC should help developers to implement granular in-app access control. For example, a user can only perform sensitive operations when strongly authenticated.

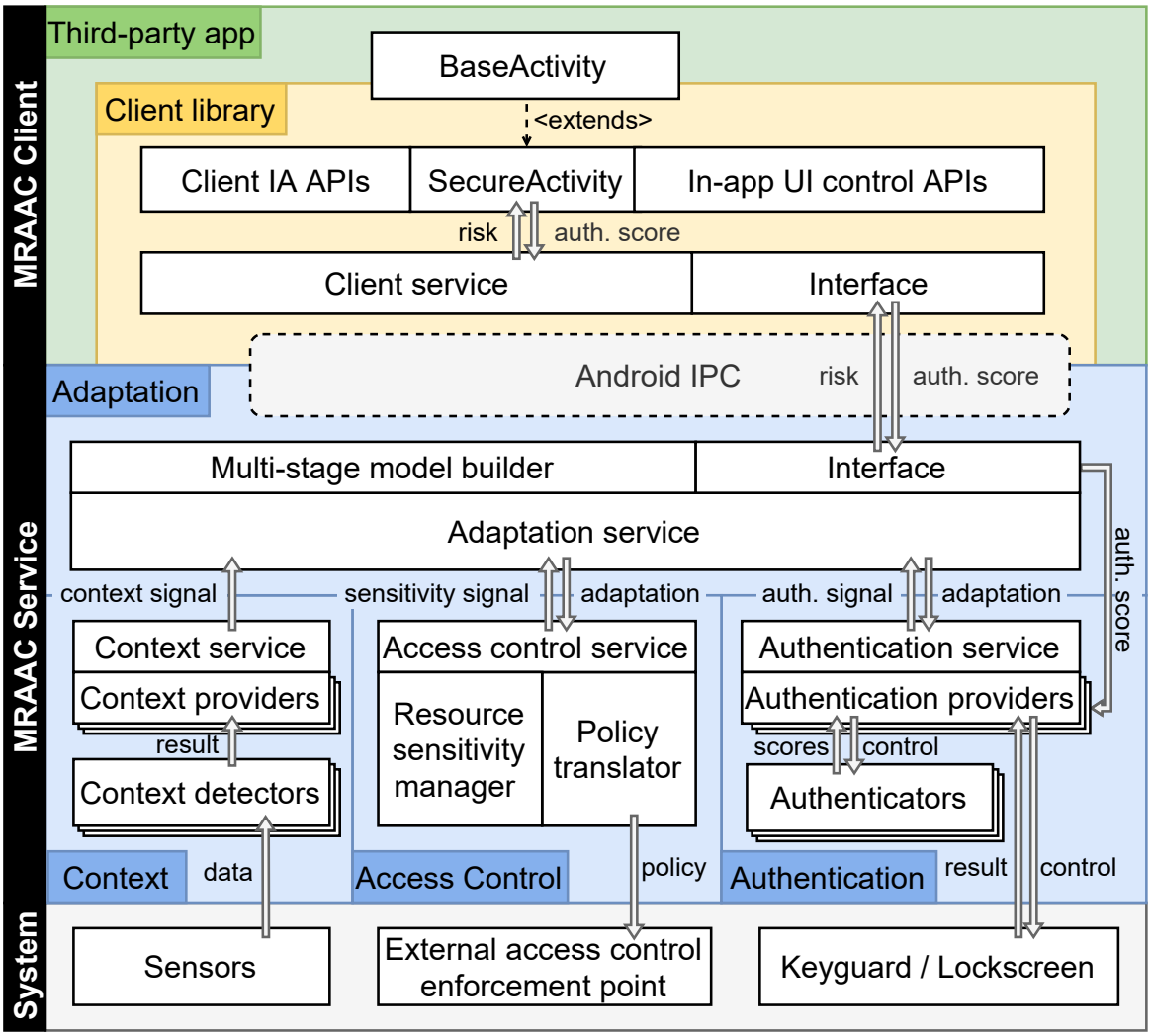


Figure 6.1: System design of the MRAAC framework.

6.4.2 Architecture

Figure 6.1 shows the architecture of MRAAC in the client-server design, which consists of a global service ("MRAAC Service") and a client library ("MRAAC Client"). Note that the "server" is deployed locally on mobile devices as an Android service. For single apps, we propose the MRAAC Integration library to include the components of both MRAAC Service and MRAAC Client.

MRAAC Service

MRAAC Service provides context detection and adapts authentication mechanisms and access control. It consists of four modules, and each module contains a service handling the communication with other modules:

Adaptation module is the core of MRAAC. It provides the multi-stage model builder to help developers generate adaptation models and design adaptation schemes. As introduced in § 6.3.2, a multi-stage adaptation model defines all stages and transitions. With the model, the adaptation service can manage the adaptation process: It listens to authentication results, sensitivity changes, and contextual signals from the other modules. According to the stage transition table, the adaptation service performs stage transitions and enforces adaptation policies based on the current stage and the received signal. It sends the adaptation outcome as a control signal to the other modules for changing their behaviors. In addition, the adaptation module provides an interface to handle communications with MRAAC client apps.

Context module hosts a list of context detectors to collect and process data from various sensors and generate contextual factors used in adaptation conditions. Each context detector works independently as a service, and a context provider is bound to it to convert its results into context signals. The context service manages a list of context providers and forwards context signals to the adaptation service. During the initialization, the context service activates all context detectors via the control interface provided by the corresponding providers.

Authentication module manages and adapts all authentication mechanisms. Similar to the context module, each authenticator runs as a service and binds to an authentication provider. The authentication service aggregates results from authenticators and sends authentication signals to the adaptation module. Besides, it receives adaptation signals from the adaptation service to (de-)activate or adjust the target authenticators or tune its aggregation method. Note that EA mechanisms provided by the operating system (e.g., Android keyguard) also require an authentication provider as an interface for activation and receiving authentication results. In addition, a client app may have its own authenticators (see § 6.4.2). We can also instantiate providers for client authenticators to receive their scores and involve them in the adaptation loop.

Access control module determines the current sensitivity level and manages access control policies. The access control service tracks the current resource and notifies the adaptation service of sensitivity changes. The resource sensitivity manager maps resources to sensitivity levels for each stage. The enforcement of access control policies is performed

by the adaptation service and the external access control enforcement point: 1) the adaptation service determines whether to move to the locked stage after receiving the sensitivity change; 2) the access control service receives the adaptation scheme of the current stage from the adaptation service, and then the policy translator translates each access control rule into the policy format accepted by the external access control enforcement point (e.g., FlaskDroid [21], ASF [13]).

MRAAC Client

Given that an app may contain resources at different sensitivity levels, it is more flexible to control access to in-app resources than to fully block access to the app [71]. Instead of offloading context detection to apps, a better approach is to enable apps to receive contextual cues from the adaptive authentication system through a library. Client apps are the apps that depend on MRAAC Service to obtain the stage information for the adaptation of in-app authentication and access control. MRAAC provides a client library for client apps, which consists of the following components:

Client service manages communications with MRAAC Service and parses the stage information. The service is activated at the startup of the app and automatically sets up the connections to MRAAC Service and pulls the current stage information. While the client app is running, the service keeps listening to broadcasts regarding stage changes from MRAAC Service. Besides, if the app adopts the IA mechanisms provided by the library, the client service can forward authentication results to MRAAC Service via Android IPC. Sending authentication results is a security-sensitive operation since malicious apps may send false results to deceive the system. MRAAC only allows forwarding authentication results when the identity of the client app is verified or the client app is signed with the same key as the service.

SecureActivity is a generic Android application component that client apps are supposed to extend to easily receive the stage change signal. The client library provides optional IA APIs based on the generic IA framework Itus [92] to enable IA within the app scope. It also provides UI control APIs to support in-app access control. App developers can delegate the control of specified UI elements to the `SecureActivity` so that it can automatically adjust the availability and visibility of UI elements based on the current stage.

MRAAC Integration

We provide the MRAAC Integration library to enable standalone adaptive authentication and access control for individual apps. It enables third-party apps to integrate the full

MRAAC Service and `SecureActivity` to provide a complete adaptation workflow within the host app. The locked stage means that the user needs to pass EA to continue using the app. The authentication process of the associated online service of the host app can be involved in the adaptation loop. For example, assuming that the host app is associated with an OAuth2 service [69], the locked stage for the host app can trigger the revocation of the authentication token. As a result, the host app logs the user out to secure the user's personal information. For the access control module, the host app needs to provide the currently accessed resource (e.g., view, file, etc.) to the service. Besides, in-app access control replaces the external access control enforcement point of MRAAC Service.

6.4.3 Development Workflow

This section lists the development workflow of using MRAAC to enable adaptive authentication and access control for MDM/EMM solutions and individual apps.

The first step is to generate the multi-stage adaptation model, which determines the high-level adaptation behaviors of the system. Developers need to specify authentication levels, sensitivity levels, and risk types based on the security requirements of their projects. Then, they define the authentication and access control functions for each risk type in table form. MRAAC provides several common functions to reduce the development efforts. However, developers still need to provide all possible context signals (related to the risk type transitions) and the risk transition functions. Once the above information is prepared, `MultiStageModelBuilder` (see Figure 6.1) constructs the multi-stage model based on a configuration file. Figures 6.3 and 6.5 show two example configuration files.

The second step is to set up the four main modules of MRAAC Service. MRAAC ships with a number of common authenticators (e.g., touch, gait) and context detectors (e.g., locations, activities) so that developers can choose from them and rapidly build their adaptive authentication system. Given that developers may have their own implementation of authenticators and context detectors, MRAAC provides two abstract classes `BaseAuthenticator` and `BaseContextDetector` to define the essential methods required for communication and controlling their customized components. Then, developers need to register all selected authenticators and context detectors in the authentication service and context service via `addAuthenticator()` and `addContextProvider()`, respectively. For the access control service, developers need to override the `acquireCurrentResource()` method to provide the current resource and direct the output of the policy translator to their access control enforcement point. For the adaption module, the developer only needs to load the model generated in the first step without touching other parts.

The third step is to make adaptation schemes and enable MRAAC Service. According to § 6.3.3, each `Scheme` object involves two kinds of adaptation policies: 1) `defaultAdaptation` takes effect at each stage transition, and 2) `conditionalAdaptation` takes effect when the condition is satisfied. Developers need to make adaptation schemes for each stage, and it is possible to set the same scheme for several stages. After scheme making, developers register all the services in the Android manifest file and start them at app/system startup.

Development workflow for client apps. Developers only need to make their base activities extend the `SecureActivity` and override `onStageChanged()` to implement their adaptation solution. Besides, the activity provides the `getCurrentStage()` method for developers to actively acquire the current stage. App developers can change the visibility of view objects and block access to sensitive methods (e.g., methods related to file operations) to protect user data privacy and security. App developers can delegate the UI control to the client library by registering target view objects for automatic control. Since the MRAAC Integration library includes the components of the client library, host app developers can enable in-app control in the same way.

6.5 Use Cases

This section presents three use cases for MRAAC: Smarter Lock, Guest-aware Continuous Authentication, and Corporate App for BYOD. Smarter Lock follows a minimal adaptation model where both the maximum authentication level and the maximum sensitivity level are one. For the latter two use cases, we use the following common settings: resources are classified into non-sensitive c_1 and sensitive c_2, and two authentication levels, a_1 and a_2, represent weakly and strongly authenticated states, respectively. The authentication transition functions follow the default scheme in Table 6.1. We adopt the default access control function (i.e., `geq`): To access a resource of c_i, the authentication level a_j should satisfy $j \geq i$.

6.5.1 Smarter Lock

We first present Smarter Lock to show how to use MRAAC to improve Smart Lock while addressing the issues and limitations mentioned in § 6.2.1. The basic idea of Smarter Lock is to keep a device unlocked if it is in a trusted environment. If any of BODY, PLACE, and DEVICE is positive, the device remains unlocked. However, if all the three factors

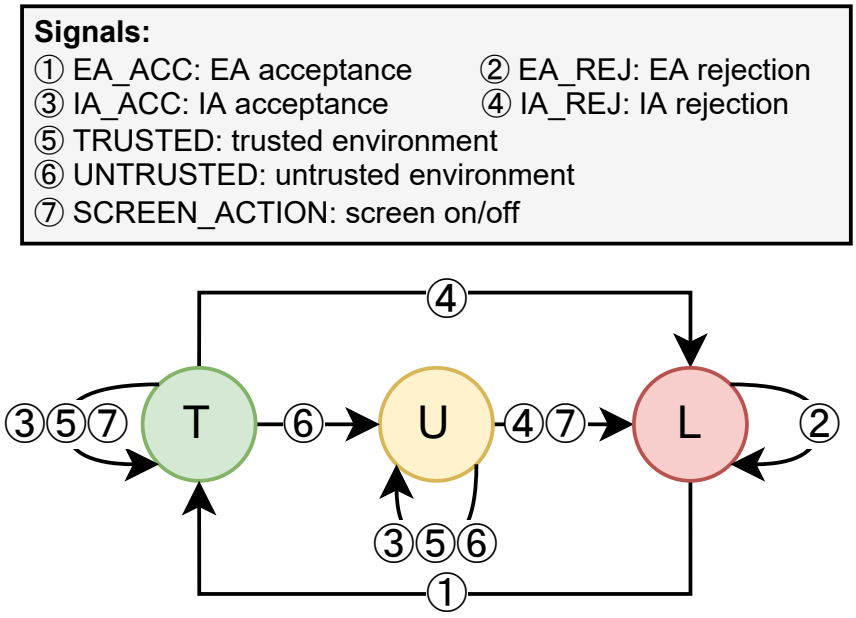


Figure 6.2: Multi-stage model for Smarter Lock. Stages: T: Trusted, U: Untrusted, L: Locked. Note: Signals ③④ are available only when IA is adopted.

are negative at any moment, the device will activate EA for the next session. Similar to Android Smart Lock, negative factors will not lead to an immediate device lock if the device is currently unlocked. We present this use case in two steps. In the first step, the adaptation model only adopts the contextual factors used by Smart Lock, i.e., BODY, PLACE, and DEVICE to address when to end the unlocked state of a device and activate EA.

In the second step, we show the extensibility of MRAAC by handling social insiders that are not covered by Android Smart Lock (including Android 10 and later). Social insiders may take advantage of the unlocked state extended by contextual factors to access the device as introduced in § 6.2.1. As a countermeasure, we enable behavioral biometrics based IA to proactively defend against social insiders. In particular, IA is used to immediately lock out an attacker upon behavioral mismatches (rather than unlock a device).

Risk model. There are two risk types in the unlocked state: Trusted and Untrusted. They indicate whether a device is in a trusted environment. In an untrusted environment, the device is more likely to be targeted by unauthorized access. For simplicity, we set both the maximum authentication level and the maximum sensitivity level to 1 since the use

case only addresses whether to lock. More complicated models are presented in the next two use cases.

We use a high-level context provider `TrustedContextProvider` (TC) to aggregate the results of the three contextual factors. It also adopts a timer (TIMEOUT) to track if the duration of being in a trusted environment exceeds a predefined value. The timeout interval is configurable by the device owner. The resulting aggregation can be expressed as the following logic expression:

$$TC = (\text{BODY} \vee \text{PLACE} \vee \text{DEVICE}) \wedge \neg\text{TIMEOUT}$$

`TrustedContextProvider` outputs two signals: `TRUSTED` and `UNTRUSTED` for transition between the two risk types. When `UNTRUSTED` is issued, the risk type changes from Trusted to Untrusted. However, `TRUSTED` cannot change the current risk type from Untrusted to Trusted since Smarter Lock adopts the low water mark strategy. Another signal is `SCREEN_ACTION`, which indicates if any screen action is performed (i.e., screen on or screen off). This signal implies a user starts or stops using the device, and the system needs to determine whether to activate EA. For Android, it relies on a system event context provider that listens to `ACTION_SCREEN_ON` and `ACTION_SCREEN_OFF`.

Multi-stage model. Based on the risk model, we obtain a three-stage adaptation model as shown in Figure 6.2. According to the model (without IA), the device keeps unlocked if it stays in a trusted environment, i.e., T. When `TrustedContextProvider` issues a `UNTRUSTED` signal (i.e., untrusted environment or the time limit exceeded), the current stage changes to U ($T \rightarrow U$) where the user needs to pass EA for the next session. After that, the device cannot move back to T even if it receives the `TRUSTED` signal (i.e., low watermark). To add or remove a contextual factor for trusted environments, a developer needs to only change `TrustedContextProvider` without changing the topology of the multi-stage model, which helps reduce possible conflicts.

Adaptation. Since only EA is available in the first step, the adaptation is about whether to activate EA. To avoid misusing contextual factors for unlocking, MRAAC makes EA acceptance the only signal that can let the system leave the locked stage (i.e., $L \rightarrow T$). It also shows that the statefulness of MRAAC makes it simple to track the state of the system and make adaptations based on both the current stage and the input signal.

Incorporating IA. Smarter Lock can employ behavioral biometrics based IA mechanisms in Stages T and U to continuously defend against unauthorized access from social insiders. We enable the IA mechanisms and add adaptation policies to activate them at T and U. If there is a negative IA result, the system will immediately lock the device and activate EA (i.e., $T \rightarrow L$ and $U \rightarrow L$). Incorporating IA can help defend against knowledgeable social

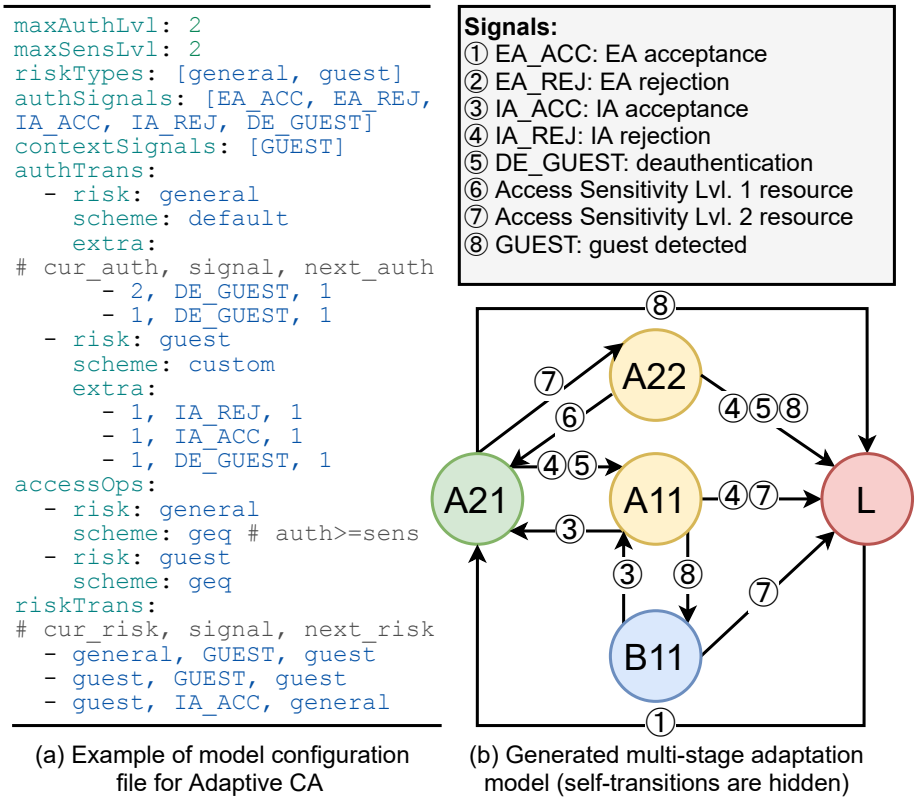
insiders: even if they know the device keeps unlocked in a trusted environment and find a chance to access the unlocked device, IA mechanisms can still block them. However, false IA rejections will immediately block a legitimate user, which may affect the usability. The next use case illustrates how we use MRAAC to mitigate false rejections.

6.5.2 Guest-aware CA

Enabling IA to continuously authenticate a user can proactively defend against unauthorized access. However, the first use case shows the necessity of adapting IA mechanisms to mitigate false IA results. Also, as introduced in Chapter 4, the authentication system should adapt to temporary device sharing where a sharee should not be blocked by IA. Thus, in this use case, we use MRAAC to implement a guest-aware CA system covering the following four aspects: 1) using the current resource sensitivity to dynamically adjust the parameters to balance false rejection and false acceptance rates; 2) incorporating access restrictions as a reaction to biometric mismatch; 3) activating authenticators only when the corresponding biometric traits are available; and 4) not blocking a guest user upon biometric mismatches while restricting a guest's access to sensitive resources.

Risk model. Figure 6.3(a) shows the configuration file of the risk model. Guest-aware CA is supposed to handle the *general* risk type (i.e., general unauthorized access attacks) and the *guest* risk type (i.e., authorized access from a guest user). A `GuestContextProvider` detects specific actions that indicate a user change (e.g., Android's Screen Pinning [78]) and issues the `GUEST` signal to enter the *guest* risk type. We note that guest access also implies a change in authentication level since the user changes to a non-owner. Thus, the guest context provider needs to issue an authentication signal `DE_GUEST` ahead of the context related `GUEST` signal, which de-authenticates the user to a_1 for guest access. In addition to the default scheme in Table 6.1, we add two `DE_GUEST` related transitions to the authentication transition function `authTran` of the general risk type. As for the `authTran` function of the guest risk type, the authentication level is fixed to a_1 in response to all the signals except for `IA_ACC`. Because an IA acceptance implies that the user changes back to the owner, we use it as the context signal for exiting from the guest risk type to the general risk type.

Multi-stage model. In Figure 6.3(b), the `MultiStageModelBuilder` builds a five-stage model based on the risk model: From the model, we can observe several adaptation flows: 1) Sensitive resource access: Only a strongly authenticated user is allowed to access sensitive resources ($A21 \rightarrow A22$), and the system blocks a weakly authenticated user or a guest ($A11 \rightarrow L$, $B11 \rightarrow L$). In addition, IA rejection or switching to a guest will trigger



(a) Example of model configuration file for Adaptive CA

(b) Generated multi-stage adaptation model (self-transitions are hidden)

Figure 6.3: Model generation for Guest-aware CA. Except the locked stage L, a stage id consists of risk type (A: general, B: guest), authentication level, and sensitivity level. E.g., $B11$ represents guest, weakly authenticated, and low sensitivity. Stages with different colors adopt different adaptation schemes.

EA during the access ($A22 \rightarrow L$). 2) IA rejection: to mitigate false IA rejection, $A11$ acts as a buffer for further verifying the user's identity ($A21 \rightarrow A11$) and only restricting access to sensitive resources instead of a direct lockout. If IA in $A11$ accepts the user, the system automatically addresses the previous false reject ($A11 \rightarrow A21$). 3) Guest access: guest access context produces two signals `DE_GUEST` and `GUEST`, and results in two-hop adaptation flows: $A21 \rightarrow A11 \rightarrow B11$ and $A11 \rightarrow A11 \rightarrow B11$. The system downgrades the authentication level in the first hop to prevent a guest from accessing sensitive resources (i.e., $A22 \rightarrow L$). In the second hop, the system adapts to the guest risk type. When the guest finishes using the device, the current user changes back to the owner, which is captured by a positive IA result ($B11 \rightarrow A11$).

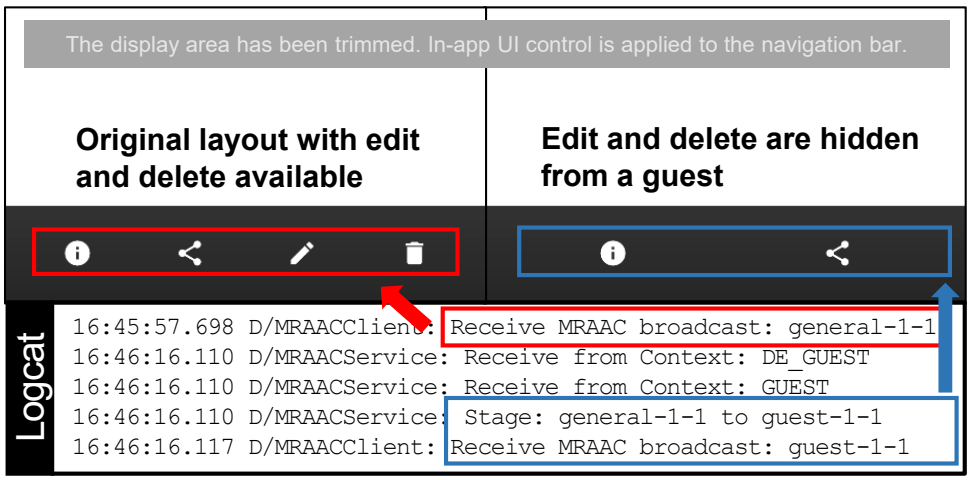


Figure 6.4: Demo: MRAAC Client Camera Roll

Adaptation. Assume that gait and touch authenticators are available. Given the low power consumption of touch-based IA [92], we activate it for all stages except for L. The default adaptation policy is expressed as a tuple `(TOUCH_IA, start, default)`, which means activating the touch authenticator with the default parameters. For the guest stage, we still need the touch authenticator to track the user's identity to determine when to exit from the guest stage. For higher-risk stages (i.e., $A11$ and $A22$), we additionally adopt the gait authenticator, which is activated when the user is on foot. We set up an `OnFootContextProvider` (based on the Google Activity Recognition API) to provide the context signal `ON_FOOT`. The conditional adaptation policy is `ON_FOOT` $\rightarrow$ `(GAIT_IA, start, default)`. We adopt a sliding-window strategy (see § 2.4.1) for result aggregation: If m out of n instances are accepted as the owner's, the authenticator will accept the current user as the owner. An example policy is `(AUTH_AGG, tune, "--reset --m=3 --n=5")`, which means resetting the sliding window and set $m = 3, n = 5$. As for access control adaptation, the device user can customize an allowlist or a blocklist for the guest stage to determine what can or cannot be accessed by a guest user.

Demonstrative example. MRAAC Service is supposed to be a system-level service that enables Guest-aware CA for the device authentication systems. We add all authenticators and context detectors and load the risk model and the adaptation schemes to initialize MRAAC Service. To show how it works with third-party apps with MRAAC Client, we modified an open-source photo gallery app, Camera Roll [98] (see Figure 6.4). Our goal is to hide the file operation buttons, *delete* and *edit* at the guest risk type. Most changes were

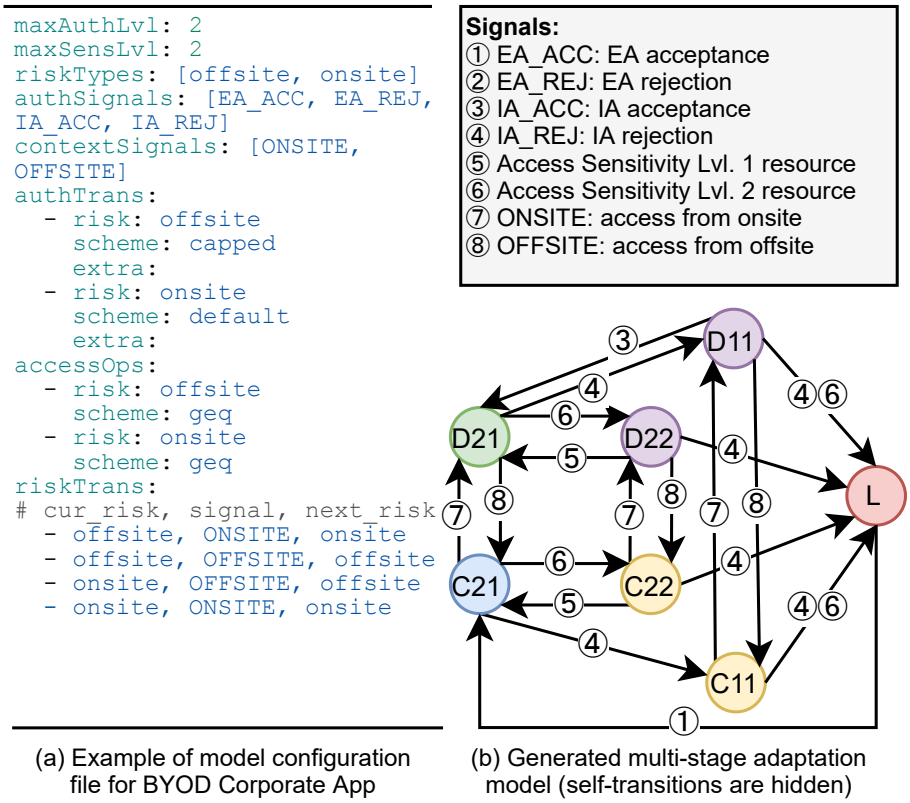
made to the app's activity classes. `ItemActivity` provides a single photo view with file operation buttons. Client app developers only need to import the MRAAC Client library and make activities inherit from `SecureActivity`. Internally, it automatically starts Client Service to connect to MRAAC Service, sends IA results and receives the stage updates. In `ItemActivity`, we overrode `onStageChanged()` and set the visibility of target buttons to `View.GONE` at the guest risk type. In the `onCreate` method of the main activity, we added `acquireRiskType()` to pull the current risk type for initialization.

6.5.3 Corporate App for BYOD

Companies adopt BYOD policies to authorize employees to use their own devices for work purposes. Employees need to use a corporate app to access corporate resources. It is essential to deploy an app-wide authentication solution to secure sensitive data. In addition to EA for login, corporate apps and services also need continuous authentication to determine when to de-authenticate a user upon suspect unauthorized access during a session. Besides, as discussed in § 6.2.1, existing IA frameworks are insufficient when it comes to a complicated adaptation model: the authentication solution should adopt different strategies according to whether a user is accessing the app onsite or offsite: if the user is accessing the app from the company, which is considered secure, it is unnecessary to adopt heavy more stringent IA mechanisms; otherwise, if the user is offsite, stricter IA is required, and more restrictions are imposed on access to corporate resources. For this use case, we use the MRAAC Integration library to enable self-contained MRAAC services and demonstrate it with FairEmail, a popular open-source e-mail client [18].

Risk model. Figure 6.5(a) shows the risk model for the BYOD use case. We adopt the capped authentication transition function (see Table 6.1) for the offsite risk type — a positive IA result cannot raise a low authentication level to the maximum. It ensures that a weakly authenticated user must be explicitly authenticated to access sensitive resources. At the same time, the user can still access non-sensitive resources in the app. We adopt a location-based context provider `OnsiteContextProvider` to determine if the device is in the company or not. It generates two context signals, `ONSITE` and `OFFSITE`, to switch between the two risk types.

Multi-stage model. Figure 6.5(a) shows the multi-stage model generated from the above risk model. We can see that most stage transitions are identical between the two risk types except that it is impossible to move from $C11$ back to $C21$ because of the capped authentication function. The other parts are similar to the general risk type of the Guest-aware CA use case.



(a) Example of model configuration file for BYOD Corporate App

(b) Generated multi-stage adaptation model (self-transitions are hidden)

Figure 6.5: Model generation for BYOD corporate app. C: offsite, D: onsite. Stages with different colors adopt different adaptation schemes.

Adaptation. Authentication adaptation involves two aspects: 1) Activation. Given that the onsite context is secure, we only need to activate the touch-based authenticator. As the offsite context implies a higher risk of unauthorized access and a user's mobility, we additionally activate the gait-based authenticator using the same conditional adaptation policy in the Guest-aware CA use case. 2) Aggregation. Since the sliding-window strategy helps balance false rejection rate and false acceptance rate, we adopt a larger window for the onsite risk type reduce false rejections and a smaller window for the offsite risk type for a low false acceptance rate. Moreover, we can further adapt the aggregation strategy to each stage within a risk type. The access control adaptation scheme for the offsite risk type is mainly about adjusting the sensitivity level of specified resources based on the current stage. For example, it is considered non-sensitive to use a corporate e-mail address to send e-mails in the company. However, it becomes sensitive for the offsite scenario because of

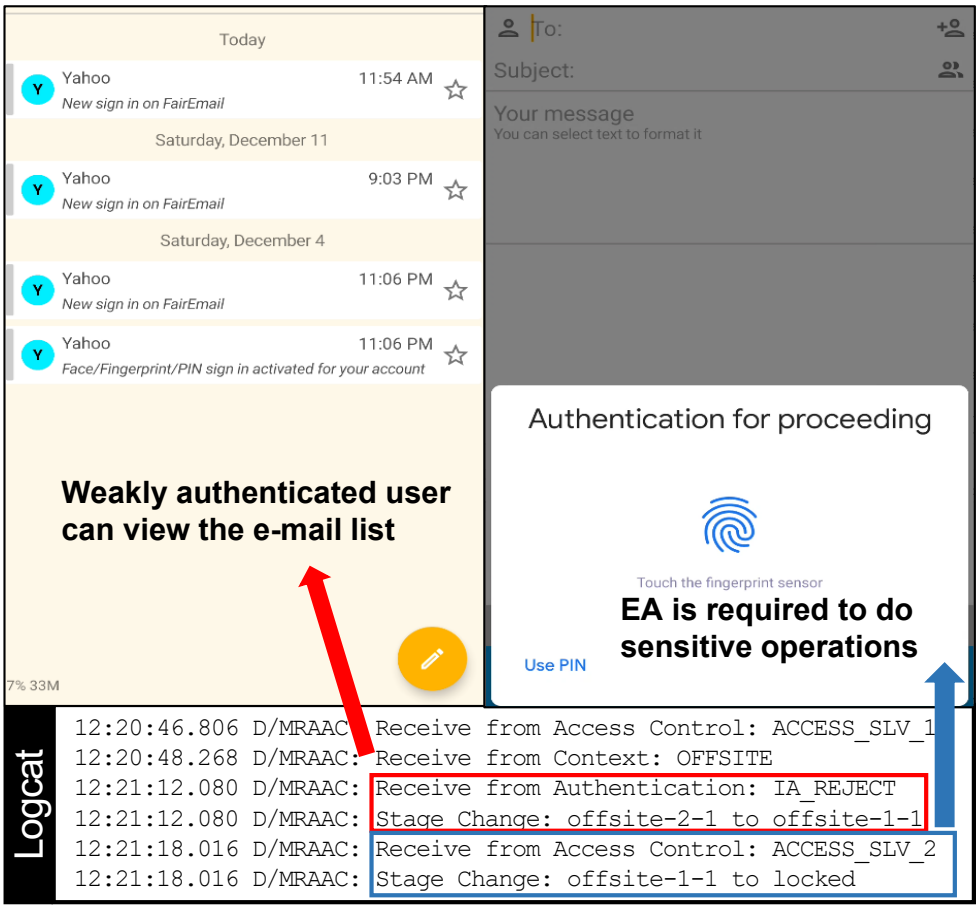


Figure 6.6: Demo: MRAAC Integrated FairEmail

possible imposter attacks. Thus, we can raise its sensitivity level to c_2 to require a high authentication level.

MRAAC Integrated FairEmail [18]. We implemented the BYOD use case on FairEmail using the MRAAC Integration library. As introduced in § 6.4.3, we created the four main services to inherit from the base services. In the adaptation service, we loaded the configuration file of the risk model and set up adaptation schemes for all stages. All the context providers and the authenticators (together with the aggregation method) should be registered in the context and authentication services, respectively. The access control service requires a resource sensitivity map for each stage so that the service can update the sensitivity accordingly. Similar to the MRAAC Client library, we extended the base activity from `SecureActivity` to enable MRAAC. The difference is that we added

`reportResourceName()` to the callbacks related to resource changes so that the access control service can receive the current resource. Developers can also report the current file, document, or data and assign a sensitivity level. For example, the e-mail app can report the details (e.g., tag, sender) of the current message. Then, e-mails from specific senders can be assigned with a higher sensitivity level. With the above process, we integrate MRAAC into the FairEmail app. Figure 6.6 shows how we enabled in-app access control of FairEmail. As described in the access control adaptation scheme, we adjusted the sensitivity level of the compose activity as high for the offsite scenario. As shown in the figure, a weakly authenticated user can view the inbox list. However, if the user wants to compose a new e-mail, they need to provide their fingerprint to proceed.

6.6 Evaluation

6.6.1 Evaluation Setup

Devices. To show the compatibility of MRAAC, we used three different Android smartphones for performance evaluation to cover both old and recent hardware and software: 1) Google Pixel (2016, CPU: 2*2.15GHz + 2 *1.6GHz, RAM: 4GB, Android 9.0), 2) Samsung S8 (2017, CPU: 4*2.3GHz + 4*1.7GHz, RAM: 4GB, Android 7.0), 3) Google Pixel 3 (2018, CPU: 4*2.5GHz + 4*1.6GHz, RAM: 4GB, Android 12).

Performance metrics. For general performance evaluation, we measured the CPU time of critical operations and the memory overhead in terms of heap size. We also measured the inter-service communication latency to evaluate how fast the adaptation service can react to a signal. In addition, we evaluated the battery consumption of MRAAC.

Configurations. Our evaluation covered the guest-aware CA and corporate app use cases in § 6.5. We evaluated the performance of the MRAAC Service integrated in the FairEmail app and evaluated the MRAAC Client using the demo Camera Roll app. *Authenticators:* We used touch- and gait-based IA mechanisms since they rely on common sensors (i.e., touchscreen, accelerometer, gyroscope) available on most smartphones and are related to common activities (i.e., swiping, walking). The touch-based IA adopts the Touchalytics [52] algorithm and conducts classification over each touch event. The gait-based IA adopts a deep neural network based model [196] and conducts authentication every five seconds (the sampling rate of the motion sensors is 50Hz). For result aggregation, we aligned the results from two authenticators in chronological order and applied a sliding window for decision making. *Context detectors:* We adopted a threshold-based

	inter-service			inter-process		
devices	auth. (ms)	context (ms)	adaptation (ms)	pull (ms)	broadcast (ms)	IA update (ms)
Pixel	1.15 (0.51)	1.23 (0.53)	10.20 (8.74)	2.17 (1.02)	9.74 (5.09)	2.18 (0.88)
S8	0.71 (0.62)	0.40 (0.12)	5.04 (1.45)	1.61 (0.64)	7.32 (3.65)	1.69 (0.75)
Pixel 3	0.50 (0.15)	0.56 (0.14)	2.70 (0.66)	0.58 (0.09)	4.30 (0.73)	1.53 (0.25)

Table 6.2: Mean latency (standard dev. in parentheses) of inter-service and inter-process communication.

`OnFootService` to tell if the user is walking. It issues a context signal every 15 seconds. The `ON_FOOT_ENTER` signal activates the gait-based IA while `ON_FOOT_EXIT` triggers the deactivation. We deployed a location context provider for the BYOD use case to detect if the user is onsite. The location update frequency was once per 15 seconds. For the Guest-aware CA use case, we manually triggered the `GUEST` signal to measure the client-server communication latency.

6.6.2 Development Overhead

To show that MRAAC provides a rapid development of adaptive authentication, we measured the development overhead of MRAAC Service and Client in term of lines of code (LOC) for the modified FairEmail and Camera Roll apps. All code changes are in Java. For FairEmail, we count the LOC of four main services of MRAAC Service: 1) Adaptation service: 25, 2) Authentication service: 51, 3) Access control service: 33, 4) Context service: 32. We add 22 lines in the base activity of FairEmail to control the four services and enable in-app access control. To bind an existing authenticator or context detector to MRAAC, developers need to implement an authentication provider or context provider. The LOC of the provider for the gait authenticator is 24, and the LOC of the provider for the onsite context detector is 30. For Camera Roll, which only enables the MRAAC Client, there are only 15 lines added to receive the risk changes and adapt the UI components.

6.6.3 Performance Evaluation Results

CPU overhead. The critical operations of MRAAC are related to the adaptation process. Thus, we instrumented the adaptation service to measure the CPU time of the following aspects and repeat ten times to calculate the average:

1. **Model construction overhead:** The adaptation service first needs to build the multi-stage model from the configuration file for initialization, which is a one-time operation. The results show that the average CPU time was about 35.0ms (std: 0.3ms) for Google Pixel 3, while Google Pixel needed 111.9ms (std: 6.4ms).

2. **Low-level adaptation overhead:** We measured the CPU time of the adaptation service processing a low-level adaptation, which was using the `ON_FOOT_ENTER` signal to activate the gait authenticator. Most cases were below 1ms for Google Pixel 3 and Samsung S8. Even for Google Pixel, the adaptation overhead was only 1.6ms (std: 0.1 ms), which is negligible.

3. **Stage transition overhead.** We tested with the stage transition $C21 \rightarrow D21$ triggered by the `ONSITE` context signal in the BYOD use case, which involves enforcing three default adaptation policies (i.e., low-level adaptation): de-activating the gait authenticator, activating the touch authenticator, and adjusting the sliding window for aggregation. The results show that the CPU time of a stage transition was about 3.3ms (std: 0.1ms) for Google Pixel 3 and about 8.6ms (std: 0.8ms) for Google Pixel. Given that all stage transitions were pre-computed during model construction, it cost low CPU overhead to perform stage transitions. Based on the low-level adaptation overhead, we can see that the enforcement of default adaptation policies took the most overhead in the stage transition.

Memory overhead. We used the Android Profiler to measure the heap size of these services. Note that we excluded the extra memory overhead from each authenticator, given that developers can choose different sets of authenticators and may produce very different results. Specifically, machine learning models and buffered sensor data may take a significant amount of memory space. The results show that MRAAC Service introduced 22kB memory overhead and MRAAC Client introduced only 2kB memory overhead. For comparison, the motion data buffer (250 samples with six double variables for each sample) used by the gait authenticator was 12kB.

Latency analysis. Authenticators and context detectors, as services, need to communicate with the authentication service or the context service to reach the adaptation service. MRAAC adopts an event bus for all signal exchanges among internal services. We measured the following inter-service latency types and repeated each experiment 100 times:

1. **Authentication latency:** the elapsed time from when an authenticator sends an authentication score to when the adaptation service receives the signal.

2. **Context latency:** the elapsed time from when a context detector sends a context signal to when the adaptation service receives the signal.

3. **Adaptation latency:** the total elapsed time from when a context detector generates a context signal to when the target authenticator receives the adaptation signal (i.e., context detector → context service → adaptation service → authentication service → authenticator).

Table 6.2 shows that both the authentication latency and the context latency were only around 1ms for all three phone models. For newer phones like Google Pixel 3, they were always below 1ms. The adaptation latency was longer since it involves more hops and adaptation policy processing. Nevertheless, even for Google Pixel, the average adaptation latency was around 10ms, which means the authenticator can adapt to the context change within a negligible time interval.

In the client-server structure, an MRAAC Client app needs to communicate with MRAAC Service via the Android IPC mechanism for obtaining the risk information and sending the client IA results. We also measured the following inter-process latency types:

1. **Pull:** MRAAC Client obtains the current risk information from MRAAC Service via Android Binder.

2. **Broadcast:** MRAAC Service broadcasts the stage updates to all MRAAC Client apps.

3. **IA update:** MRAAC Client sends its client IA results to MRAAC Service via Android Binder.

Table 6.2 shows that the pull operation and the IA update had very short latency given that they both rely on Android Binder. The broadcast latency was acceptable given that in 95% of the experiments, the client app received the stage update from MRAAC Service within 20ms on the Google Pixel phone.

Battery consumption. We used the Battery Historian [58] to measure the battery consumption of the adaptation process of MRAAC Service on Google Pixel 3. We made the context detectors and the gait authenticator run at a fixed rate. We tested the following two settings and measured one-hour battery consumption five times for each setting: 1) Enabling adaptation. MRAAC was forced to perform one high-level adaptation (i.e., processing the `ON_SITE` and `OFF_SITE` signals) and one low-level adaptation (i.e., processing the `ON_FOOT_ENTER` and the `ON_FOOT_EXIT` signals) every 15 seconds (i.e., the maximum

adaptation frequency). However, the gait authenticator was set to ignore the adaptation outcome and keep running. 2) Disabling adaptation. MRAAC did not process any adaptations. From the result, the average hourly consumption of the first setting was 2.95% and the second setting was 2.92%. It shows that the adaptation process introduced very low battery consumption to the device.

6.6.4 Use Case Evaluation

We conducted a trace-based evaluation based on the BYOD use case to show how MRAAC helps defend against unauthorized access with reducing false rejections. We used real-world sensor data to generate simulation traces and fed them into the MRAAC-integrated FairEmail app running on Google Pixel 3 in real-time so that we can simulate unauthorized access and daily device usage events and log the adaptations, stage transitions, and authentication results of MRAAC. As introduced in § 6.5.2, the multi-stage design enables us to implement an adaptive sliding window that adopts different (m, n) pairs based on the current stage. For simulation settings, we chose the majority vote as the final decision ($m = \lceil n/2 \rceil$) and selected three different n's: $5, 9, 11$. A larger n targets a lower false rejection rate at the cost of longer reaction time and higher false acceptance rate, which is suitable for lower-risk stages. Thus, for onsite stages, we set $n = 11$ for $D21$ and $n = 9$ for $D11, D22$; for offsite stages, we set $n = 9$ for $C21$ and $n = 5$ for $C11, C22$ (note: given the high risk of $C11$ and $C22$, we chose a small n to ensure low reaction time.) In addition to the adaptive sliding window, we adopted the touch authenticator for all stages except the locked stage and adopted the gait authenticator only at $C11$, $C21$, $C22$.

For the simulation task, we randomly selected 10 users from the HMOG dataset [159], which includes motion sensor data and touch data of reading activities on a smartphone while walking or sitting. For each user, we used six sessions of data to train the authenticator models and used another two sessions (one "walking + reading" and one "sitting + reading") for simulation. We replaced raw sensor data with the HMOG data and matched the timestamp to real-world time. Since HMOG did not provide location data and app access data, we randomly generated location switch events (switching between onsite and offsite) and app switch events (switching between sensitive and non-sensitive resources). The interval between two switch events followed exponential distributions. We set the average intervals as one and two minutes for location switch events and app switch events, respectively. We chose short intervals to trigger more adaptations to test the robustness of MRAAC.

Attack detection experiments. In the first set of experiments, we chose one user as

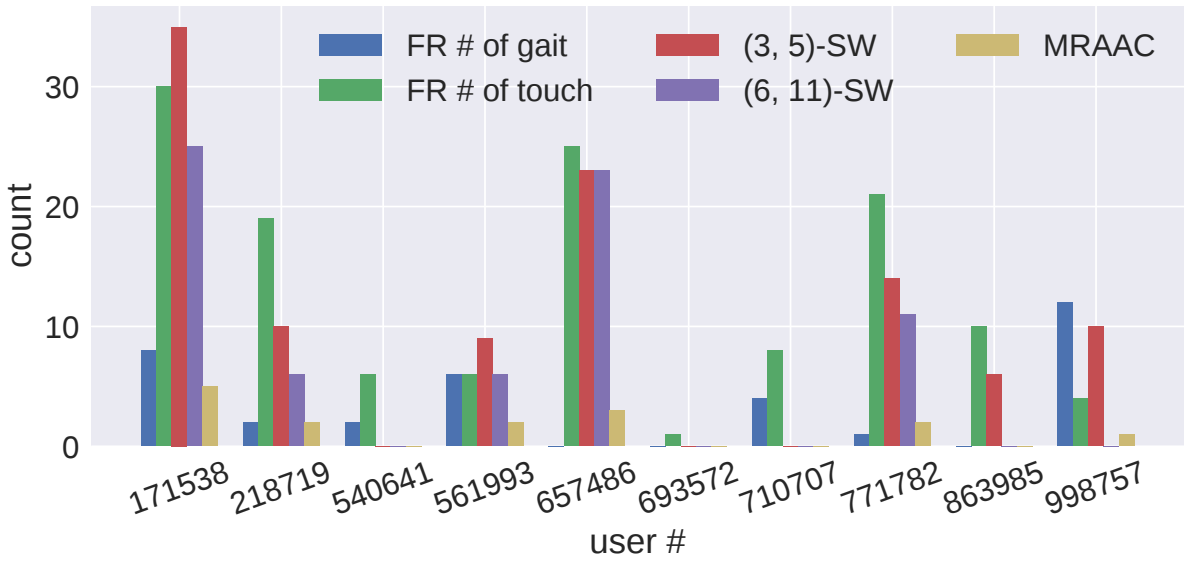


Figure 6.7: The per-user raw false rejection numbers of authenticators and the EA numbers of baseline methods and MRAAC.

the device owner and all the other users as attackers and repeated it for all users. We ensured the attackers' data was not used in the negative training data of the owner user. An attacker is supposed to access the sensitive resources under the offsite context. We measured how long the adaptive authentication system took to lock out the attacker. We study two locking durations: 1) t_h is the duration between the attacker picking up the device and the device locking out the attacker. 2) t_t is the duration between the attacker performing the first touch event and the device locking out the attacker. We study t_t separately since the attacker might not operate on the device immediately after picking up the device. Among 90 attack events, MRAAC failed to detect only one attack when both the gait-based and touch-based authenticators falsely accepted the attacker. The average t_h was 33.2s (95th percentile = 60.6s). The average t_t was 11.0s (95th percentile = 36.0s), which implies that attackers did not have much time to launch an active attack before being blocked. Other biometrics based IA mechanisms may be able to detect an attack even faster. Importantly, MRAAC is oblivious to the type of mechanism being used. Of course, without the use of an IA mechanism, the attack would not get detected at all. In summary, with MRAAC managing the authentication adaptation and performing multi-modal authentication, the authentication system can efficiently detect unauthorized

access.

Usability experiments. Another adaptation goal is to reduce unnecessary detection and false IA rejection, which may interrupt the user and activate EA. In the second set of experiments, we ran the complete trace of each user (the length of each trace was 10-15 minutes). The baseline methods for comparison are adopting two constant-size sliding windows, (3, 5) and (6, 11), which are the smallest and the largest window sizes of the adaptive sliding window, respectively. We assume the user immediately passes EA and continues using the device if the system triggers EA. MRAAC performed 361 stage transitions for all ten users without any undefined transitions. EA was triggered 15 times in total: 12 cases occurred at the access to sensitive resources and 11 cases occurred at offsite (eight common cases when both conditions hold). Besides, no EA was triggered for four out of the ten users. Figure 6.7 compares the raw authentication results of the gait and touch authenticators as well as two baseline methods to the aggregated results using the adaptive sliding window strategy enabled by MRAAC. The result shows that individual authenticators were prone to make false rejections due to their low model accuracy. Note that the gait authenticator produced fewer false rejection decisions because the user was not always walking. A sliding window with larger window size ($n = 11$) can help reduce the false rejections of individual authenticators. However, MRAAC made the fewest false rejections among all methods for nine out of ten users. Only for user 998757, MRAAC activated EA once while (6, 11)-SW did not activate any EA. It was triggered when MRAAC was adopting a small window size at a high risk stage. We also note that the performance of MRAAC was bounded by the performance of individual authenticators. Nevertheless, we can see significant performance gain brought by the multi-stage design of MRAAC. Then, we measured the activation times of the gait-based authenticators. Due to the adaptation mechanism, the gait authenticator was only active for 33.7 minutes out of 122 minutes for all experiments. The above simulation results have shown that MRAAC can help schedule authenticators to reduce unnecessary detection and false rejection rate.

6.7 Discussion

Client-server communications. To support MDM/EMM solutions, MRAAC proposes a client-server structure, which involves communications between the client app and the MRAAC service. Potential attacks may happen during the communication. One the one hand, the stage information broadcast to a client app may include or imply sensitive information (e.g., a user's location). Thus, the MRAAC service can only send digested or coarse-grained risk information that is essential to the client app. The client app should

also declare the required permissions of sensitive risk information. On the other hand, the client app can send fake authentication results to MRAAC, which may lead to impersonation attacks (fake IA acceptance) and denial of service attacks (fake IA rejection). Thus, MRAAC only allows verified and trusted client apps to send authentication results (see § 6.4.2) to avoid these attacks.

Accommodation of existing adaptive mechanisms. We propose MRAAC as a general adaptive authentication framework that is compatible with existing adaptive systems. Most existing adaptive authentication mechanisms [70,142,147] can be accommodated and abstracted as conditional policies. For example, CASA and PRISM use machine learning techniques to learn context models from sensor data, and map the model output to a certain adaptation scheme that (de-)activates an authenticator or changes its parameters. Each model is a context provider and provides a signal for stage transition or an adaptation policy in MRAAC.

Modeling and policy making. MRAAC is primarily provided for developers to design a multi-stage adaptive authentication system. Developers are responsible for designing use cases and using MRAAC to build the multi-stage model for their authentication system. However, end users may have their own preferences and requirements of security and usability and may want to configure the adaptive authentication system. Existing studies [31,51] have investigated how to enable users to customize authentication mechanisms. Thus, it should be possible for end users to configure some components, such as authenticators and context providers (e.g., setting up a geofence for the trusted place [31]). End users should also be able to determine what use cases to enable, but cannot modify the multi-stage model to change the adaptation flow of a certain use case. A possible avenue is to implement a usable configurable adaptive authentication system for end users based on the MRAAC framework.

For deployment, all stages except the locked stage are transparent to end users, i.e., stage transitions and adaptations are performed automatically and internally. End users may perceive the existence of MRAAC only when EA is activated. Thus, the complexity of a multi-stage model is not a direct factor affecting end users' normal device usage.

Usability study. Our simulation used traces generated from a real-world dataset to evaluate the usability of MRAAC in terms of the number of EA triggered. Existing studies [93] have covered the usability of IA mechanisms and shown that users are concerned with interrupt-authenticates (i.e., immediate device lock with EA). According to our experiment results, we can observe that MRAAC can significantly reduce the times of EA compared to individual IA mechanisms. However, performing a user study to collect user perceptions and feedback about MRAAC is future work.

6.8 Conclusion

In this chapter, we proposed a multi-stage risk-aware adaptive authentication and access control framework, MRAAC. The multi-stage model hierarchically organizes multiple adaptation flows driven by various contextual factors and authentication results. This design reduces the conflicts between different adaptation policies and avoid potentially insecure adaptations like using weak contextual factors to unlock a locked device. It supports progressive and complex adaptation workflows to handle different scenarios. The MRAAC framework helps developers to rapidly build multi-stage adaptation models based on their security and usability requirements. MRAAC provides two libraries to enable adaptive authentication and make third-party apps implement in-app control with risk awareness, respectively. We presented three use cases that adopt multi-stage adaptation models to address the practical problems. Based on the use cases, we extended a gallery app and an E-mail client app to show the extensibility of MRAAC. Extensive evaluation has shown the effectiveness of MRAAC in balancing security and usability of authentication systems with low performance overhead.

Chapter 7

Conclusion

In this chapter, we conclude the thesis with summarizing the contributions and providing possible future avenues.

7.1 Summary

Our main research objective was to investigate the adaptations of user authentication systems on mobile devices to various context changes, involving *why to adapt*, *what to adapt*, and *how to adapt*.

Chapter 3 investigated smartphone loss prevention solutions and addressed Objective 1, which can be classified as a *why-to-adapt* problem. A potential device loss is a reason for locking the device and activating explicit authentication while a user's absence from the device is the context of a potential device loss. We proposed Chaperone, an active acoustic sensing based context sensing technique to obtain the user's proximity and activity. It exploited the acoustic signals reflected by a user's body to measure the distance between the user and the device and determine the user's motion state. The design of Chaperone considered the impact of various environmental factors, such as high frequency noise, nearby people, and obstacles, on the detection accuracy. Our extensive real-world experiments showed the effectiveness and robustness of Chaperone in detecting a user's departure and absence. In addition to automatic locking, we also investigated another reaction to a potential device loss, which is alerting the user to prevent device loss. A small-scale user study has shown that Chaperone was able to effectively alert a user before leaving the premises.

Chapter 4 investigated temporary device sharing and addressed Objective 2, which involves both *why-to-adapt* and *what-to-adapt*. Thus, the why-to-adapt problem is about determining the contextual factors that imply device sharing, while the what-to-adapt problem is about adjusting the authentication and access control strategies to allow a sharee to access non-sensitive resources without being blocked. To address these problems, we proposed device sharing awareness (DSA) that used handover gesture detection and owner detection based on behavioral biometrics to automatically handle a device sharing event. Considering the device owner's forgetfulness or the trust implications of device sharing, our DSA solution enabled an implicit sharing loop with little to no input from the device owner. As for the outcome of a detected device sharing event, DSA enabled app-level access control to grant a sharee access to non-sensitive resources while hiding sensitive resources. In addition, the design of DSA enabled the processing of possible failures of the sharing detection. The results of handover detection evaluation showed that our handover detection solution could accurately capture device sharing across different devices and users, and distinguish sharing gestures from other types of hand movements. Our user study showed that DSA could automatically process the device sharing events, and its exception procession mechanism could mitigate possible exposure of sensitive resources caused by false handover detection or owner detection.

Chapter 5 investigated multi-user IA for shared mobile devices and addressed Objective 3, which is mainly a *what-to-adapt* problem. The problem involves the changes in the algorithmic level and the systematic level of an implicit authentication system. At the algorithmic level, we addressed the following three problems: 1) extending single-user IA to achieve real-time multi-user identification while rejecting attackers, 2) adopting the Dempster-Shafer theory to fusing multiple modalities with different uncertainties for better identification accuracy, 3) applying automatic data segmentation and labeling to update IA models with new incoming data and users. At the systematic level, we proposed the SHRIMPS framework to provide the architecture of a multi-user, multi-modal IA system for facilitating the above solutions. SHRIMPS also provided an evaluation framework to help IA researchers to rapidly design evaluation tasks and compare different multi-user IA schemes. The results of the evaluation on four public datasets have shown that our proposed multi-user IA solution with Dempster-Shafer theory based score fusion methods effectively detected user switches in the middle of a session and reduced false detection rate compared to single-modality methods and other score fusion methods.

Chapter 6 proposed a general adaptive authentication framework and address Objective 4, which is a *how-to-adapt* problem. We proposed the multi-stage adaptation model to organize adaptations in a hierarchical structure to fulfill the security and usability requirements of various scenarios. The multi-stage model helped 1) reduce conflicts between

policies, 2) avoid insecure adaptations like using contextual factors for unlocking, 3) enable progressive adaptation workflows based on the feedback of the previous adaptation. We designed MRAAC, a multi-stage risk-aware adaptive authentication and access control framework, to help developers automatically generate multi-stage models and develop adaptive authentication systems. We presented three use cases to show the extensibility of MRAAC, and demonstrated how to enable MRAAC on two open-source apps to address the practical adaptation problems at low development and performance overheads.

7.2 Future Work

This section lists the future avenues for the work in each chapter.

Chapter 3 is mainly about context sensing techniques for detecting a user's proximity and movements. Chaperone is based on active acoustic sensing, which can provide accurate distance estimation and reliable motion detection without the help of external hardware. However, according to our user study, people still have concerns with additional power consumption brought by acoustic sensing. The current solution adopted by Chaperone is to use the trigger module to avoid sensing at specific locations or situations where the device is with the owner. Possible ways to improve power consumption include 1) designing an adaptive sensing mechanism to balance the sensing frequency and detection accuracy of Chaperone, and 2) introducing other contextual factors to the trigger module. Besides, acoustic sensing can be restricted by some physical constraints that completely block the signal transmission (e.g., the device in a bag). To address this problem, Chaperone should work with other sensing techniques to cover more scenarios. For example, we can estimate the coarse-grained user-device distance based on the strength (e.g., RSSI [193]) of the signal received by an additional Bluetooth-enabled device (e.g., smartwatch) when Chaperone fails to work. Also, adopting continuous authentication techniques [122] enables smartphones to block non-owner's access to a lost device. However, a follow-up problem is how to coordinate these techniques with Chaperone to balance power consumption, false positives, and false negatives. In addition, we will perform user studies as future work to evaluate the acceptability of Chaperone from the overhead perspective, including power consumption in real-world device usage and users' perception.

Chapter 4 involves the context sensing technique for handover gesture detection and access control for device sharing. We acknowledge that there are other implicit sharing gestures through which a device owner can pass the device to a sharee. Thus, extending sharing detection with other gestures is a possible avenue. Nevertheless, the implicit sharing loop proposed by DSA is designed to accommodate different gesture detection methods.

For the evaluation of DSA, we conducted a user study to ask participants to perform a series device sharing tasks, which may not cover the real-world device sharing practices. In the future, we plan to conduct a field study to capture real-world device sharing events and collect users' feedback about DSA.

Chapter 5 proposed multi-user, multi-modal IA. Our SHRIMPS framework provides a general architecture of a multi-user IA system and an evaluation environment to test different schemes. In the future, we can use SHRIMPS to extend more existing single-user IA schemes for multi-user scenarios and test their performance. As for the fusion of different modalities, we used the Dempster-Shafer theory based methods with AUC and EER for uncertainty measurements. For future work, we can design more complicated uncertainty functions with more possible factors that may affect the uncertainty of a modality (e.g., temporal factors, training data amount) Another promising avenue is implementing the multi-user, multi-modal IA system on real mobile devices and incorporating it into existing multi-user systems to handle automatic logout. Due to the COVID-19 pandemic, we chose to use existing public datasets for evaluation rather than collecting new datasets. However, only the HMOG dataset satisfied our requirements. For the others, we had to fuse multiple datasets to provide sufficient multi-modal data. We plan to collect a multi-user, multi-modal dataset that contains cross-session behavioral data for each user for future work.

Chapter 6 focused on the organization of authentication and access control adaptions. We proposed the multi-stage adaptation model where each stage represents a certain risk type, authentication level, and sensitivity level. Accordingly, we can regulate the high-level adaptations (i.e., stage transitions) by comparing two connected stages such that a higher-risk stage will require stricter authentication. Based on this idea, a potential avenue is to automatically determine the adaptation policies within each stage, that is, which authentication mechanism to activate and what parameters to adopt. It requires quantifying the performance of authentication mechanisms to determine their strictness and usability from the perspectives of accuracy, availability, power consumption, etc. Since we designed MRAAC for various stakeholders to enable adaptive authentication on their apps and solutions, another possible avenue is to collect their feedback about using MRAAC. Also, a user study for testing MRAAC-enabled apps will be our future work since it can help us understand how MRAAC improves the usability of adaptive authentication in practice.

7.3 Last Word

This thesis investigated adaptive authentication on mobile devices. Due to the COVID-19 pandemic, conducting user studies and human participant experiments has been signifi-

cantly restricted. Thus, we had to adapt our research methodology and direction accordingly. We focused on designing general adaptive authentication frameworks and exemplifying them with detailed use cases. For evaluation, we used existing public datasets and performed trace-based experiments to evaluate our proposed frameworks in a reproducible way.

After having addressed our research objectives, we can recommend a general workflow for developing an adaptive authentication solution to a specific scenario: 1) characterizing the risk of the target scenario with contextual factors and developing context sensing techniques to capture these factors, 2) analyzing the security and usability requirements of the target scenario and deciding how to adapt the authentication system accordingly, 3) handling exceptions of possible false detection of context sensing techniques or implicit authentication. This thesis also proposed several novel context sensing techniques and adaptive authentication solutions and frameworks for various stakeholders, including researchers, app developers, MDM/EMM developers, etc. We believe that our studies, systems, and frameworks can benefit these stakeholders to develop their own adaptive authentication solutions.

References

[1] Mohammed Abuhamad, Ahmed Abusnaina, DaeHun Nyang, and David Mohaisen. Sensor-based continuous authentication of smartphones' users using behavioral biometrics: A contemporary survey. *IEEE Internet of Things Journal*, 8(1):65–84, 2020.

[2] Fadel Adib, Zach Kabelac, Dina Katabi, and Robert C Miller. 3D tracking via body radio reflections. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI '14)*, pages 317–329, 2014.

[3] Fadel Adib and Dina Katabi. See through walls with WiFi! In *the ACM SIGCOMM 2013 conference on SIGCOMM (SIGCOMM '13)*. ACM, 2013.

[4] Syed Ishtiaque Ahmed, Md Romael Haque, Jay Chen, and Nicola Dell. Digital privacy challenges with shared mobile phone use in bangladesh. *the ACM on Human-Computer Interaction*, 1(CSCW):1–20, 2017.

[5] Syed Ishtiaque Ahmed, Md Romael Haque, Irtaza Haider, Jay Chen, and Nicola Dell. "Everyone has some personal stuff": Designing to support digital privacy with shared mobile phone use in Bangladesh. In *the 2019 SGICHI Conference on Human Factors in Computing Systems (CHI '19)*, pages 1–13. ACM, 2019.

[6] Ahmad Akl, Chen Feng, and Shahrokh Valaee. A novel accelerometer-based gesture recognition system. *IEEE Transactions on Signal Processing*, 59(12):6197–6205, 2011.

[7] Mahdi Nasrullah Al-Ameen, Huzeyfe Kocabas, Swapnil Nandy, and Tanjina Tamanna. "We, three brothers have always known everything of each other": A cross-cultural study of sharing digital devices and online accounts. *Proceedings on Privacy Enhancing Technologies*, 2021(4):203–224, 2021.

[8] Mojtaba Alizadeh, Saeid Abolfazli, Mazdak Zamani, Sabariah Baharun, and Kouichi Sakurai. Authentication in mobile cloud computing: A survey. *Journal of Network and Computer Applications*, 61:59–80, 2016.

[9] Android. Supporting multiple users. https://source.android.com/devices/tech/admin/multi-user.

[10] Patricia Arias-Cabarcos, Christian Krupitzer, and Christian Becker. A survey on adaptive authentication. *ACM Computing Surveys (CSUR)*, 52(4):1–30, 2019.

[11] Jeff Avery, Daniel Vogel, Edward Lank, Damien Masson, and Hanae Rateau. Holding patterns: detecting handedness with a moving smartphone at pickup. In *the 31st Conference on l'Interaction Homme-Machine*. ACM, 2019.

[12] Microsoft Azure. Overview of shared device mode. https://docs.microsoft.com/en-us/azure/active-directory/develop/msal-shared-devices, accessed Jan, 2022.

[13] Michael Backes, Sven Bugiel, Sebastian Gerling, and Philipp von Styp-Rekowsky. Android security framework: Extensible multi-layered access control on android. In *the 30th Annual Computer Security Applications Conference (ACSAC '14)*, pages 46–55, 2014.

[14] Kiran Balagani, Matteo Cardaioli, Mauro Conti, Paolo Gasti, Martin Georgiev, Tristan Gurtler, Daniele Lain, Charissa Miller, Kendall Molas, Nikita Samarin, et al. Pilot: Password and pin information leakage from obfuscated typing videos. *Journal of Computer Security*, 27(4):405–425, 2019.

[15] Ling Bao and Stephen S Intille. Activity recognition from user-annotated acceleration data. In *International Conference on Pervasive Computing*. Springer, 2004.

[16] AK Belman, L Wang, SS Iyengar, P Sniatala, R Wright, R Dora, J Baldwin, Z Jin, and VV Phoha. SU-AIS BB-MAS (syracuse university and assured information security-behavioral biometrics multi-device and multi-activity data from same users) dataset. *IEEE DataPort*, 2019.

[17] Cheng Bo, Lan Zhang, Taeho Jung, Junze Han, Xiang-Yang Li, and Yu Wang. Continuous user identification via touch and movement behavioral biometrics. In *33rd International Performance Computing and Communications Conference (IPCCC '14)*. IEEE, 2014.

[18] Marcel Bokhorst. Fairemail. https://github.com/M66B/FairEmail, accessed Dec, 2021.

[19] Joseph Bonneau, Cormac Herley, Paul C Van Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *2012 IEEE Symposium on Security and Privacy (S&P '12)*. IEEE, 2012.

[20] Lucas Bourtoule, Varun Chandrasekaran, Christopher A Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 141–159. IEEE, 2021.

[21] Sven Bugiel, Stephen Heuser, and Ahmad-Reza Sadeghi. Flexible and fine-grained mandatory access control on Android for diverse security and privacy policies. In *22nd USENIX Security Symposium (USENIX Security '13)*. USENIX, 2013.

[22] Andreas Bulling, Ulf Blanke, and Bernt Schiele. A tutorial on human activity recognition using body-worn inertial sensors. *ACM Computing Surveys (CSUR)*, 46(3):1–33, 2014.

[23] Attaullah Buriro, Bruno Crispo, Filippo Del Frari, Jeffrey Klardie, and Konrad Wrona. Itsme: Multi-modal and unobtrusive behavioural user authentication for smartphones. In *International conference on passwords*. Springer, 2015.

[24] Daniel Buschek, Fabian Hartmann, Emanuel Von Zezschwitz, Alexander De Luca, and Florian Alt. SnapApp: Reducing authentication overhead with a time-constrained fast unlock option. In *the 2016 SIGCHI Conference on Human Factors in Computing Systems (CHI '16)*, 2016.

[25] Business-news.eu. 4% or €20 million: A stolen mobile may cost you hundreds, but could cost your employer millions. https://www.business-news.eu/2019-4-or-20-million-a-stolen-mobile-may-cost-you-hundreds-but-could-cost-your-employer-millions, 2019.

[26] Chao Cai, Rong Zheng, and Jun Luo. Ubiquitous acoustic sensing on commodity iot devices: A survey. *IEEE Communications Surveys & Tutorials*, 2022.

[27] Jagmohan Chauhan, Yining Hu, Suranga Seneviratne, Archan Misra, Aruna Seneviratne, and Youngki Lee. BreathPrint: Breathing acoustics-based user authentication.

In *the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '17)*, pages 278–291, 2017.

[28] Jagmohan Chauhan, Young D Kwon, Pan Hui, and Cecilia Mascolo. ContAuth: continual learning framework for behavioral-based user authentication. *ACM Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT '20)*, 4(4):1–23, 2020.

[29] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[30] Jiayi Chen, Urs Hengartner, Hassan Khan, and Mohammad Mannan. Chaperone: Real-time locking and loss prevention for smartphones. In *29th USENIX Security Symposium (USENIX Security '20)*. USENIX, 2020.

[31] Geumhwan Cho, Sungsu Kwag, Huh Jun Ho, Bedeuro Kim, Choong-Hoon Lee, and Hyoungshick Kim. Towards usable and secure location-based smartphone authentication. In *Seventeenth Symposium on Usable Privacy and Security (SOUPS '21)*. USENIX, 2021.

[32] Selina Chu, Shrikanth Narayanan, and C-C Jay Kuo. Environmental sound recognition with time–frequency audio features. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(6):1142–1158, 2009.

[33] Heather Crawford and Ebad Ahmadzadeh. Authentication on the go: Assessing the effect of movement on mobile device keystroke dynamics. In *the Thirteenth Symposium on Usable Privacy and Security (SOUPS '17)*. USENIX, 2017.

[34] Heather Crawford, Karen Renaud, and Tim Storer. A framework for continuous, transparent mobile device authentication. *Computers & Security*, 39:127–136, 2013.

[35] Francesco Crenna, Giovanni Battista Rossi, and Marta Berardengo. Filtering biomechanical signals in movement analysis. *Sensors*, 21(13):4580, 2021.

[36] Federico Cruciani, Anastasios Vafeiadis, Chris Nugent, Ian Cleland, Paul McCullagh, Konstantinos Votis, Dimitrios Giakoumis, Dimitrios Tzovaras, Liming Chen, and Raouf Hamzaoui. Feature learning for human activity recognition using convolutional neural networks. *CCF Transactions on Pervasive Computing and Interaction*, 2(1):18–32, 2020.

[37] Alexander De Luca, Alina Hang, Frederik Brudy, Christian Lindner, and Heinrich Hussmann. Touch me once and i know it's you!: implicit authentication based on touch screen patterns. In *the 2012 SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*, pages 987–996. ACM, 2012.

[38] Mohammad Omar Derawi, Claudia Nickel, Patrick Bours, and Christoph Busch. Unobtrusive user-authentication on mobile phones using biometric gait recognition. In *2010 Sixth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. IEEE, 2010.

[39] DigitalTrends. Study reveals americans lost $30 billion worth of mobile phones last year. https://www.digitaltrends.com/mobile/study-reveals-americans-lost-30-billion-of-mobile-phones-last-year/, 2012.

[40] Benjamin Draffin, Jiang Zhu, and Joy Zhang. Keysens: Passive user authentication through micro-behavior modeling of soft keyboard interaction. In *International Conference on Mobile Computing, Applications, and Services*. Springer, 2013.

[41] Simon Eberz, Kasper B Rasmussen, Vincent Lenders, and Ivan Martinovic. Evaluating behavioral biometrics for continuous authentication: Challenges and metrics. In *the 2017 ACM on Asia Conference on Computer and Communications Security (AsiaCCS '17)*. ACM, 2017.

[42] Serge Egelman, Sakshi Jain, Rebecca S Portnoff, Kerwell Liao, Sunny Consolvo, and David Wagner. Are you ready to lock? In *the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*, 2014.

[43] Muhammad Ehatisham-ul Haq, Muhammad Awais Azam, Usman Naeem, Yasar Amin, and Jonathan Loo. Continuous authentication of smartphone users based on activity pattern recognition using passive mobile sensing. *Journal of Network and Computer Applications*, 109:24–35, 2018.

[44] AYMAN El-Sayed. Multi-biometric systems: a state of the art survey and research directions. *International Journal of Advanced Computer Science and Applications*, 6, 2015.

[45] Samsung Electronics. S secure. https://galaxystore.samsung.com/prepost/0000004637448.

[46] Glenn Elert. Frequency range of human hearing. *The Physics Factbook*, 2003.

[47] Mohammed E Fathy, Vishal M Patel, and Rama Chellappa. Face-based active authentication on mobile devices. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '15)*. IEEE, 2015.

[48] Alberto Fernández, Salvador Garcia, Francisco Herrera, and Nitesh V Chawla. SMOTE for learning from imbalanced data: progress and challenges, marking the 15-year anniversary. *Journal of Artificial Intelligence Research*, 61:863–905, 2018.

[49] Davide Figo, Pedro C Diniz, Diogo R Ferreira, and Joao MP Cardoso. Preprocessing techniques for context recognition from accelerometer data. *Personal and Ubiquitous Computing*, 14(7):645–662, 2010.

[50] Finder.com.au. Australians have blown $755 million on replacements. https://www.finder.com.au/press-release-australians-have-blown-755-million-on-replacements, 2017.

[51] Alain Forget, Sonia Chiasson, and Robert Biddle. Choose your own authentication. In *the 2015 New Security Paradigms Workshop*, 2015.

[52] Mario Frank, Ralf Biedert, Eugene Ma, Ivan Martinovic, and Dawn Song. Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication. *IEEE Transactions on Information Forensics and Security*, 8(1):136–148, 2012.

[53] Lex Fridman, Ariel Stolerman, Sayandeep Acharya, Patrick Brennan, Patrick Juola, Rachel Greenstadt, and Moshe Kam. Multi-modal decision fusion for continuous authentication. *Computers & Electrical Engineering*, 41, 2015.

[54] Matteo Gadaleta and Michele Rossi. Idnet: Smartphone-based gait recognition with convolutional neural networks. *Pattern Recognition*, 74:25–37, 2018.

[55] Martin Georgiev, Simon Eberz, Henry Turner, Giulio Lovisotto, and Ivan Martinovic. Common evaluation pitfalls in touch-based authentication systems, 2022.

[56] Mikhail I Gofman, Sinjini Mitra, Tsu-Hsiang Kevin Cheng, and Nicholas T Smith. Multimodal biometrics for enhanced mobile device security. *Communications of the ACM*, 59(4):58–65, 2016.

[57] Liangyi Gong, Wu Yang, Dapeng Man, Guozhong Dong, Miao Yu, and Jiguang Lv. WiFi-based real-time calibration-free passive human motion detection. *Sensors*, 15(12):32213–32229, 2015.

[58] Google. Battery Historian. https://github.com/google/battery-historian, 2017.

[59] Google. Jacquard Jacket. https://atap.google.com/jacquard/, 2018.

[60] Francesco Gringoli, Matthias Schulz, Jakob Link, and Matthias Hollick. Free your csi: A channel state information extraction platform for modern wi-fi chipsets. In *the 13th International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization*, pages 21–28, 2019.

[61] The Guardian. Shock! Horror! Do you know how much time you spend on your phone? https://www.theguardian.com/lifeandstyle/2019/aug/21/cellphone-screen-time-average-habits.

[62] Tiago Guerreiro, Ricardo Gamboa, and Joaquim Jorge. Mnemonical body shortcuts: improving mobile interaction. In *the 15th European Conference on Cognitive Ergonomics: the Ergonomics of Cool Interaction*, 2008.

[63] Sandeep Gupta, Attaullah Buriro, and Bruno Crispo. DriverAuth: A risk-based multi-modal biometric-based driver authentication scheme for ride-sharing platforms. *Computers & Security*, 83:122–139, 2019.

[64] Sandeep Gupta, Mouna Kacimi, and Bruno Crispo. Step & turn-a novel bimodal behavioral biometric-based user verification scheme for physical access control. *Computers & Security*, page 102722, 2022.

[65] Alina Hang, Emanuel Von Zezschwitz, Alexander De Luca, and Heinrich Hussmann. Too much information! User attitudes towards smartphone sharing. In *the 7th Nordic Conference on Human-Computer Interaction: Making Sense Through Design*, 2012.

[66] James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982.

[67] Marian Harbach, Alexander De Luca, and Serge Egelman. The anatomy of smartphone unlocking: A field study of android lock screens. In *the 2016 SIGCHI Conference on Human Factors in Computing Systems (CHI '16)*, 2016.

[68] Marian Harbach, Emanuel Von Zezschwitz, Andreas Fichtner, Alexander De Luca, and Matthew Smith. It's a hard lock life: A field study of smartphone (un) locking behavior and risk perception. In *Symposium On Usable Privacy and Security (SOUPS '14)*. USENIX, 2014.

[69] Dick Hardt et al. The OAuth 2.0 authorization framework, 2012.

[70] Eiji Hayashi, Sauvik Das, Shahriyar Amini, Jason Hong, and Ian Oakley. CASA: context-aware scalable authentication. In *the Ninth Symposium on Usable Privacy and Security (SOUPS '13)*, 2013.

[71] Eiji Hayashi, Oriana Riva, Karin Strauss, AJ Bernheim Brush, and Stuart Schechter. Goldilocks and the two mobile devices: Going beyond all-or-nothing access to a device's applications. In *the Eighth Symposium on Usable Privacy and Security (SOUPS '12)*. USENIX, 2012.

[72] Wenfeng He, Kaishun Wu, Yongpan Zou, and Zhong Ming. Wig: Wifi-based gesture recognition system. In *2015 24th International Conference on Computer Communication and Networks (ICCCN '15)*. IEEE, 2015.

[73] Samuli Hemminki, Petteri Nurmi, and Sasu Tarkoma. Accelerometer-based transportation mode detection on smartphones. In *the 11th ACM Conference on Embedded Networked Sensor Systems*, 2013.

[74] Daniel Hintze, Matthias Füller, Sebastian Scholz, Rainhard D Findling, Muhammad Muaaz, Philipp Kapfer, Eckhard Koch, and René Mayrhofer. Cormorant: Ubiquitous risk-aware multi-modal biometric authentication across mobile devices. *ACM Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT '19)*, 3(3):1–23, 2019.

[75] Martin Hirzel, Scott Schneider, and Kanat Tangwongsan. Sliding-window aggregation algorithms: Tutorial. In *the 11th ACM International Conference on Distributed and Event-based Systems*, 2017.

[76] Yu-Hsuan Huang and Homer H Chen. Face recognition under low illumination via deep feature reconstruction network. In *2020 IEEE International Conference on Image Processing (ICIP '20)*. IEEE, 2020.

[77] Apple Inc. Guided access. https://support.apple.com/en-us/HT202612.

[78] Google Inc. Android pin & unpin screens. https://support.google.com/android/answer/9455138?hl=en.

[79] Google Inc. Manage guests and users. https://support.google.com/nexus/topic/6126546?hl=en&ref_topic=3416294.

[80] Google Inc. Android 10. https://source.android.com/security/enhancements/enhancements10, accessed Apr, 2022.

[81] Google Inc. Measuring biometric unlock security. https://source.android.com/security/biometric/measure, accessed Dec, 2021.

[82] Google Inc. Smart lock. https://support.google.com/pixelphone/answer/6093922?hl=en, accessed Dec, 2021.

[83] Google Inc. Motion sensors. https://developer.android.com/guide/topics/sensors/sensors_motion, accessed Mar, 2022.

[84] Google Inc. Android activity recognition API. https://developers.google.com/location-context/activity-recognition, accessed Oct, 2021.

[85] Xiaomi Inc. App vault. https://play.google.com/store/apps/details?id=com.mi.android.globalminusscreen.

[86] Anil K Jain, Lin Hong, and Yatin Kulkarni. A multimodal biometric system using fingerprint, face and speech. In *2nd International Conference of AVBPA*, volume 10, 1999.

[87] Anil K Jain and Stan Z Li. *Handbook of face recognition*, volume 1. Springer, 2011.

[88] Markus Jakobsson, Elaine Shi, Philippe Golle, and Richard Chow. Implicit authentication for mobile devices. In *the 4th USENIX conference on Hot topics in security (HotSec '09)*. USENIX, 2009.

[89] Tyler Kaczmarek, Ercan Ozturk, and Gene Tsudik. Assentication: user deauthentication and lunchtime attack mitigation with seated posture biometric. In *International Conference on Applied Cryptography and Network Security*, pages 616–633. Springer, 2018.

[90] Amy K Karlson, AJ Bernheim Brush, and Stuart Schechter. Can I borrow your phone? Understanding concerns when sharing mobile phones. In *the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, 2009.

[91] Kaspersky Lab. Around 23,000 devices go missing every month, finds Kaspersky Lab. Press release (Aug. 9, 2018). https://www.kaspersky.com/about/press-releases/2018_missing-devices, 2018.

[92] Hassan Khan, Aaron Atwater, and Urs Hengartner. Itus: An implicit authentication framework for Android. In *the 20th Annual International Conference on Mobile Computing and Networking (MobiSys '14)*, pages 507–518. ACM, 2014.

[93] Hassan Khan, Urs Hengartner, and Daniel Vogel. Usability and security perceptions of implicit authentication: convenient, secure, sometimes annoying. In *Eleventh Symposium on Usable Privacy and Security (SOUPS '15)*. USENIX, 2015.

[94] Hassan Khan, Urs Hengartner, and Daniel Vogel. Targeted mimicry attacks on touch input based implicit authentication schemes. In *the 14th International Conference on Mobile Systems, Applications and Services (MobiSys '16)*. ACM, 2016.

[95] Hassan Khan, Urs Hengartner, and Daniel Vogel. Augmented reality-based mimicry attacks on behaviour-based smartphone authentication. In *the 16th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '18)*. ACM, 2018.

[96] Ji-Hae Kim, Gwang-Soo Hong, Byung-Gyu Kim, and Debi P Dogra. deepgesture: Deep learning-based gesture recognition scheme using motion sensors. *Displays*, 55:38–45, 2018.

[97] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *the National Academy of Sciences*, 114(13), 2017.

[98] Lukas Koller. Camera Roll Android app. https://github.com/kollerlukas/Camera-Roll-android-App.

[99] Masoud Mehrabi Koushki, Borke Obada-Obieh, Jun Ho Huh, and Konstantin Beznosov. On smartphone users' difficulty with understanding implicit authentication. In *the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21)*. ACM, 2021.

[100] Rajesh Kumar, Vir V Phoha, and Abdul Serwadda. Continuous authentication of smartphone users by fusing typing, swiping, and phone movement patterns. In *2016 IEEE 8th International Conference on Biometrics Theory, Applications and Systems (BTAS '16)*. IEEE, 2016.

[101] Jennifer R Kwapisz, Gary M Weiss, and Samuel A Moore. Activity recognition using cell phone accelerometers. *ACM SigKDD Explorations Newsletter*, 12(2):74–82, 2011.

[102] L.A. Times. 68% of smartphone theft victims never recover device, report says. https://www.latimes.com/business/technology/la-fi-tn-70-smartphone-theft-victims-never-recover-device-20140507-story.html, 2014.

[103] Norton Labs. Norton app lock. https://play.google.com/store/apps/details?id=com.symantec.applock.

[104] Imane Lamiche, Guo Bin, Yao Jing, Zhiwen Yu, and Abdenour Hadid. A continuous smartphone authentication method based on gait patterns and keystroke dynamics. *Journal of Ambient Intelligence and Humanized Computing*, 10(11):4417–4430, 2019.

[105] Patrick Lazik, Niranjini Rajagopal, Oliver Shih, Bruno Sinopoli, and Anthony Rowe. ALPS: A Bluetooth and ultrasound platform for mapping and localization. In *the 13th ACM Conference on Embedded Networked Sensor Systems (SenSys '15)*. ACM, 2015.

[106] Seon-Woo Lee and Kenji Mase. Activity and location recognition using wearable sensors. *IEEE Pervasive Computing*, 1(3):24–32, 2002.

[107] Hong Li, Wei Yang, Jianxin Wang, Yang Xu, and Liusheng Huang. Wifinger: Talk to your smart devices with finger-grained gesture. In *the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '16)*, 2016.

[108] Tao Li, Yimin Chen, Jingchao Sun, Xiaocong Jin, and Yanchao Zhang. iLock: immediate and automatic locking of mobile devices against data theft. In *the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*, pages 933–944. ACM, 2016.

[109] Jie Lian, Jiadong Lou, Li Chen, and Xu Yuan. Echospot: Spotting your locations via acoustic sensing. *ACM Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT '21)*, 5(3):1–21, 2021.

[110] Hong Lin, Jiafen Liu, and Qing Li. TDSD: A touch dynamic and sensor data based approach for continuous user authentication. In *PACIS 2018 Proceedings.*, 2018.

[111] Jiayang Liu, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing*, 5(6):657–675, 2009.

[112] Xinyu Liu, David Wagner, and Serge Egelman. Detecting phone theft using machine learning. In *the 2018 International Conference on Information Science and System (ICISS '18)*. ACM, 2018.

[113] Yunxin Liu, Ahmad Rahmati, Yuanhe Huang, Hyukjae Jang, Lin Zhong, Yongguang Zhang, and Shensheng Zhang. xShare: Supporting impromptu sharing of mobile phones. In *the 7th International Conference on Mobile Systems, Applications, and Services (MobiSys '09)*. ACM, 2009.

[114] Pedro Lopes Silva, Eduardo Luz, Gladston Moreira, Lauro Moraes, and David Menotti. Chimerical dataset creation protocol based on doddington zoo: A biometric application with face, eye, and ecg. *Sensors*, 19(13):2968, 2019.

[115] Huawei Device Co. Ltd. Create a privatespace for your private data. https://consumer.huawei.com/en/support/content/en-us15834600/.

[116] Zhiyuan Lu, Xiang Chen, Qiang Li, Xu Zhang, and Ping Zhou. A hand gesture recognition framework and wearable gesture-based interaction prototype for mobile devices. *IEEE Transactions on Human-machine Systems*, 44(2):293–299, 2014.

[117] Yongsen Ma, Gang Zhou, and Shuangquan Wang. WiFi sensing with channel state information: A survey. *ACM Computing Surveys (CSUR)*, 52(3):1–36, 2019.

[118] Upal Mahbub, Vishal M Patel, Deepak Chandra, Brandon Barbello, and Rama Chellappa. Partial face detection for continuous authentication. In *2016 IEEE International Conference on Image Processing (ICIP '16)*. IEEE, 2016.

[119] Stéphane Mallat. *A wavelet tour of signal processing*. Elsevier, 1999.

[120] Wenguang Mao, Jian He, and Lili Qiu. CAT: High-precision acoustic motion tracking. In *the 22nd Annual International Conference on Mobile Computing and Networking (MobiCom '16)*, pages 69–81. ACM, 2016.

[121] Wenguang Mao, Mei Wang, and Lili Qiu. AIM: Acoustic imaging on a mobile. In *the 16th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '18)*. ACM, 2018.

[122] Shrirang Mare, Andrés Molina Markham, Cory Cornelius, Ronald Peterson, and David Kotz. Zebra: Zero-effort bilateral recurring authentication. In *2014 IEEE Symposium on Security and Privacy*, pages 705–720. IEEE, 2014.

[123] Diogo Marques, Tiago Guerreiro, Luís Carriço, Ivan Beschastnikh, and Konstantin Beznosov. Vulnerability & blame: Making sense of unauthorized access to smartphones. In *the 2019 SIGCHI Conference on Human Factors in Computing Systems (CHI '19)*. ACM, 2019.

[124] Tara Matthews, Kerwell Liao, Anna Turner, Marianne Berkovich, Robert Reeder, and Sunny Consolvo. "She'll just grab any device that's close": A study of everyday device & account sharing in households. In *the SIGCHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, 2016.

[125] Michelle L Mazurek, JP Arsenault, Joanna Bresee, Nitin Gupta, Iulia Ion, Christina Johns, Daniel Lee, Yuan Liang, Jenny Olsen, Brandon Salmon, et al. Access control for home data sharing: Attitudes, needs and practices. In *the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, 2010.

[126] Masoud Mehrabi Koushki, Borke Obada-Obieh, Jun Ho Huh, and Konstantin Beznosov. Is implicit authentication on smartphones really popular? on Android users' perception of "smart lock for Android". In *22nd International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '20)*. ACM, 2020.

[127] Cisco Meraki. Configuring apple school manager for shared ipad or 1-to-1 ipad. ConfiguringAppleSchoolManagerforSharediPador1-to-1iPad, accessed Jan, 2022.

[128] L Mezai, F Hachouf, and M Bengherabi. Fusion of face and voice using the dempster-shafer theory for person verification. In *International Workshop on Systems, Signal Processing and their Applications (WOSSPA '11)*. IEEE, 2011.

[129] Markus Miettinen, Stephan Heuser, Wiebke Kronz, Ahmad-Reza Sadeghi, and N Asokan. ConXsense: Automated context classification for context-aware access control. In *the 9th ACM Symposium on Information, Computer and Communications Security (AsiaCCS '14)*. ACM, 2014.

[130] Yisroel Mirsky, Asaf Shabtai, Lior Rokach, Bracha Shapira, and Yuval Elovici. SherLock vs Moriarty: A smartphone dataset for cybersecurity research. In *the 2016 ACM Workshop on Artificial Intelligence and Security (AISec '16)*. ACM, 2016.

[131] Daniela Moctezuma, Eric S Tellez, Sabino Miranda-Jiménez, and Mario Graff. Appearance model update based on online learning and soft-biometrics traits for people re-identification in multi-camera environments. *IET Image Processing*, 13(12):2162–2168, 2019.

[132] Lindasalwa Muda, Mumtaj Begam, and Irraivan Elamvazuthi. Voice recognition algorithms using mel frequency cepstral coefficient (MFCC) and dynamic time warping (DTW) techniques. *arXiv preprint arXiv:1003.4083*, 2010.

[133] Seth Neel, Aaron Roth, and Saeed Sharifi-Malvajerdi. Descent-to-delete: Gradient-based methods for machine unlearning. In *Algorithmic Learning Theory*, pages 931–962. PMLR, 2021.

[134] NetworkWorld.com. More data breaches caused by lost devices than malware or hacking, Trend Micro says. https://www.networkworld.com/article/2988643/device-loss-data-breach-malware-hacking-trend-micro-report.html, 2015.

[135] Xudong Ni, Zhimin Yang, Xiaole Bai, Adam C Champion, and Dong Xuan. DiffUser: Differentiated user access control on smartphones. In *2009 IEEE 6th International Conference on Mobile Adhoc and Sensor Systems (MASS '09)*. IEEE, 2009.

[136] Sameera Palipana, Piyush Agrawal, and Dirk Pesch. Channel state information based human presence detection using non-linear techniques. In *the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*, 2016.

[137] Patec. Mini Bluetooth anti-lost anti-theft alarm kid pet object finder. https://www.amazon.ca/Patec%C2%AE-Bluetooth-Anti-lost-Anti-theft-Object/dp/B00MPGKL1U, 2014.

[138] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[139] Abena Primo, Vir V Phoha, Rajesh Kumar, and Abdul Serwadda. Context-aware active authentication using smartphone accelerometer measurements. In *the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. IEEE, 2014.

[140] Protect Your Bubble. Nearly half a million Brits had phones stolen last year. https://uk.protectyourbubble.com/our-blog/blog/2017/03/23/nearly-half-million-brits-mobile-phone-stolen-last-year, 2017.

[141] Saumay Pushp, Yunxin Liu, Mengwei Xu, Changyoung Koh, and Junehwa Song. PrivacyShield: A mobile system for supporting subtle just-in-time privacy provisioning through off-screen-based touch gestures. *the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT '18)*, 2(2):1–38, 2018.

[142] Arun Ramakrishnan, Jochen Tombal, Davy Preuveneers, and Yolande Berbers. Prism: Policy-driven risk-based implicit locking for improving the security of mobile end-user devices. In *the 13th International Conference on Advances in Mobile Computing and Multimedia (MoMM '15)*. ACM, 2015.

[143] Arun Kishore Ramakrishnan, Davy Preuveneers, and Yolande Berbers. A loosely coupled and distributed bayesian framework for multi-context recognition in dynamic ubiquitous environments. In *2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing*. IEEE, 2013.

[144] Theodore S Rappaport et al. *Wireless communications: principles and practice*, volume 2. prentice hall PTR New Jersey, 1996.

[145] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *the IEEE conference on Computer Vision and Pattern Recognition (CVPR '17)*. IEEE, 2017.

[146] Yanzhi Ren, Chen Wang, Jie Yang, and Yingying Chen. Fine-grained sleep monitoring: Hearing your breathing with smartphones. In *the 2015 IEEE Conference on Computer Communications (INFOCOM '15)*. IEEE, 2015.

[147] Oriana Riva, Chuan Qin, Karin Strauss, and Dimitrios Lymberopoulos. Progressive authentication: deciding when to authenticate on mobile phones. In *21st USENIX Security Symposium (USENIX Security '12)*. USENIX, 2012.

[148] Jong-hyuk Roh, Sung-Hun Lee, and Soohyung Kim. Keystroke dynamics for authentication in smartphone. In *2016 international conference on information and communication technology convergence (ICTC '16)*. IEEE, 2016.

[149] David Sachs. Sensor fusion on Android devices: A revolution in motion processing. *Google Tech Talk*, 2, 2010.

[150] Hataichanok Saevanee, Nathan Clarke, Steven Furnell, and Valerio Biscione. Continuous user authentication using multi-modal biometrics. *Computers & Security*, 53:234–246, 2015.

[151] Nithya Sambasivan, Garen Checkley, Amna Batool, Nova Ahmed, David Nemer, Laura Sanely Gaytán-Lugo, Tara Matthews, Sunny Consolvo, and Elizabeth Churchill. "Privacy is not for me, it's for those rich women": Performative privacy practices on mobile phones by women in south asia. In *the Fourteenth Symposium on Usable Privacy and Security (SOUPS '18)*. USENIX, 2018.

[152] Stuart E Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. The emperor's new security indicators. In *IEEE Symposium on Security and Privacy (S&P '07)*. IEEE, 2007.

[153] Julian Seifert, Alexander De Luca, Bettina Conradi, and Heinrich Hussmann. TreasurePhone: Context-sensitive user data protection on mobile phones. In *International Conference on Pervasive Computing*. Springer, 2010.

[154] Kari Sentz, Scott Ferson, et al. *Combination of evidence in Dempster-Shafer theory*, volume 4015. Sandia National Laboratories Albuquerque, 2002.

[155] Teddy Seyed, Xing-Dong Yang, and Daniel Vogel. A modular smartphone for lending. In *the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. ACM, 2017.

[156] Sheng Shen, He Wang, and Romit Roy Choudhury. I am a smartwatch and I can track my user's arm. In *the 14th annual international conference on Mobile systems, applications, and services (MobiSys '16)*. ACM, 2016.

[157] Siho Shin, Jaehyo Jung, and Youn Tae Kim. A study of an EMG-based authentication algorithm using an artificial neural network. In *SENSORS*. IEEE, 2017.

[158] Babins Shrestha, Manar Mohamed, and Nitesh Saxena. Zemfa: zero-effort multi-factor authentication based on multi-modal gait biometrics. In *2019 17th International Conference on Privacy, Security and Trust (PST '19)*. IEEE, 2019.

[159] Zdeňka Sitová, Jaroslav Šeděnka, Qing Yang, Ge Peng, Gang Zhou, Paolo Gasti, and Kiran S Balagani. HMOG: New behavioral biometric features for continuous authentication of smartphone users. *IEEE Transactions on Information Forensics and Security*, 11(5):877–892, 2015.

[160] Max Smith-Creasey and Muttukrishnan Rajarajan. A novel scheme to address the fusion uncertainty in multi-modal continuous authentication schemes on mobile devices. In *International Conference on Biometrics*. IEEE, 2019.

[161] Mohamed Soltane, Noureddine Doghmane, and Noureddine Guersi. Face and speech based multi-modal biometric authentication. *International Journal of Advanced Science and Technology*, 21(6):41–56, 2010.

[162] Wesllen Sousa Lima, Eduardo Souto, Khalil El-Khatib, Roozbeh Jalali, and Joao Gama. Human activity recognition using inertial sensors in a smartphone: An overview. *Sensors*, 19(14):3213, 2019.

[163] Ke Sun, Ting Zhao, Wei Wang, and Lei Xie. VSkin: Sensing touch gestures on surfaces of mobile devices using acoustic signals. In *the 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18)*. ACM, 2018.

[164] Symantec. The Symantec smartphone honey stick project. https://www.symantec.com/content/en/us/about/presskits/b-symantec-smartphone-honey-stick-project.en-us.pdf, 2012.

[165] Furkan Tari, A Ant Ozok, and Stephen H Holden. A comparison of perceived and real shoulder-surfing risks between alphanumeric and graphical passwords. In *the Second Symposium on Usable Privacy and Security (SOUPS '06)*. USENIX, 2006.

[166] TheRegister.co.uk. A quarter of banks' data breaches are down to lost phones and laptops. https://www.theregister.co.uk/2016/08/25/us_bank_breaches_survey/, 2016.

[167] Yu-Chih Tung, Duc Bui, and Kang G Shin. Cross-platform support for rapid development of mobile acoustic sensing applications. In *the 16th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '18)*, pages 455–467. ACM, 2018.

[168] Yu-Chih Tung and Kang G Shin. EchoTag: Accurate infrastructure-free indoor location tagging with smartphones. In *the 21st Annual International Conference on Mobile Computing and Networking (MobiCom '15)*. ACM, 2015.

[169] Yu-Chih Tung and Kang G Shin. Use of phone sensors to enhance distracted pedestrians' safety. *IEEE Transactions on Mobile Computing*, 17(6):1469–1482, 2017.

[170] Uber. The 2019 Uber lost & found index. https://www.uber.com/newsroom/2019-uber-lost-found-index, 2019.

[171] Yonatan Vaizman, Katherine Ellis, and Gert Lanckriet. Recognizing detailed human context in the wild from smartphones and smartwatches. *IEEE Pervasive Computing (PerCom '17)*, 16(4):62–74, 2017.

[172] Yonatan Vaizman, Katherine Ellis, Gert Lanckriet, and Nadir Weibel. ExtraSensory app: Data collection in-the-wild with rich user interface to self-report behavior. In *the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, 2018.

[173] Sudip Vhaduri and Christian Poellabauer. Multi-modal biometric-based implicit authentication of wearable device users. *IEEE Transactions on Information Forensics and Security*, 14(12):3116–3125, 2019.

[174] Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. Deep learning for sensor-based activity recognition: A survey. *Pattern Recognition Letters*, 119:3–11, 2019.

[175] Wei Wang, Alex X Liu, Muhammad Shahzad, Kang Ling, and Sanglu Lu. Understanding and modeling of WiFi signal based human activity recognition. In *the 21st Annual International Conference on Mobile Computing and Networking (MobiCom '15)*. ACM, 2015.

[176] Wei Wang, Alex X. Liu, and Ke Sun. Device-free gesture tracking using acoustic signals. In *the 22nd Annual International Conference on Mobile Computing and Networking (MobiCom '16)*. ACM, 2016.

[177] Yuxi Wang, Kaishun Wu, and Lionel M Ni. WiFall: Device-free fall detection by wireless networks. *IEEE Transactions on Mobile Computing*, 16(2):581–594, 2017.

[178] Jason Wiese, T. Scott Saponas, and A.J. Bernheim Brush. Phoneprioception: Enabling mobile phones to infer where they are kept. In *the 2013 SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, 2013.

[179] Adam Wójtowicz and Krzysztof Joachimiak. Model for adaptable context-based biometric authentication for mobile devices. *Personal and Ubiquitous Computing*, 20(2):195–207, 2016.

[180] Huadong Wu, Mel Siegel, Rainer Stiefelhagen, and Jie Yang. Sensor fusion using dempster-shafer theory [for context-aware hci]. In *IEEE Instrumentation and Measurement Technology Conference*, volume 1. IEEE, 2002.

[181] Xiaomi. Mi Band. https://www.mi.com/global/miband/, 2014.

[182] Yang Xu, Wei Yang, Jianxin Wang, Xing Zhou, Hong Li, and Liusheng Huang. Wistep: Device-free step counting with WiFi signals. *ACM Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT '18)*, 1(4):1–23, 2018.

[183] Qing Yang, Ge Peng, David T Nguyen, Xin Qi, Gang Zhou, Zdeňka Sitová, Paolo Gasti, and Kiran S Balagani. A multimodal data set for evaluating continuous authentication performance in smartphones. In *the 12th ACM Conference on Embedded Network Sensor Systems (SenSys '14)*. ACM, 2014.

[184] Xing-Dong Yang, Khalad Hasan, Neil Bruce, and Pourang Irani. Surround-See: Enabling peripheral vision on smartphones during active use. In *the 26th Annual*

ACM Symposium on User Interface Software and Technology (UIST '13). ACM, 2013.

[185] Xingjie Yu, Zhan Wang, Kun Sun, Wen Tao Zhu, Neng Gao, and Jiwu Jing. Remotely wiping sensitive data on stolen smartphones. In *the 9th ACM Symposium on Information, Computer and Communications Security (ASIACCS '14)*. ACM, 2014.

[186] Sangki Yun, Yi-Chao Chen, Huihuang Zheng, Lili Qiu, and Wenguang Mao. Strata: Fine-grained acoustic-based device-free tracking. In *the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '17)*. ACM, 2017.

[187] Özgür Yürür, Chi Harold Liu, Zhengguo Sheng, Victor CM Leung, Wilfrido Moreno, and Kin K Leung. Context-awareness for mobile sensing: A survey and future directions. *IEEE Communications Surveys & Tutorials*, 18(1):68–93, 2014.

[188] Huanle Zhang, Wan Du, Pengfei Zhou, Mo Li, and Prasant Mohapatra. DopEnc: Acoustic-based encounter profiling using smartphones. In *the 22nd Annual International Conference on Mobile Computing and Networking (MobiCom '16)*. ACM, 2016.

[189] Linghan Zhang, Sheng Tan, Jie Yang, and Yingying Chen. VoiceLive: A phoneme localization based liveness detection for voice authentication on smartphones. In *the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*, 2016.

[190] Nan Zheng, Kun Bai, Hai Huang, and Haining Wang. You are how you touch: User verification on smartphones via tapping behaviors. In *22nd International Conference on Network Protocols (ICNP '14)*. IEEE, 2014.

[191] Bing Zhou, Mohammed Elbadry, Ruipeng Gao, and Fan Ye. BatMapper: Acoustic sensing based indoor floor plan construction using smartphones. In *the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '17)*. ACM, 2017.

[192] Bing Zhou, Jay Lohokare, Ruipeng Gao, and Fan Ye. EchoPrint: Two-factor authentication using acoustics and vision on smartphones. In *the 24th Annual International Conference on Mobile Computing and Networking (MobiCom' 18)*. ACM, 2018.

[193] Cheng Zhou, Jiazheng Yuan, Hongzhe Liu, and Jing Qiu. Bluetooth indoor positioning based on rssi and kalman filter. *Wireless Personal Communications*, 96(3):4115–4130, 2017.

[194] Man Zhou, Qian Wang, Jingxiao Yang, Qi Li, Feng Xiao, Zhibo Wang, and Xiaofen Chen. PatternListener: Cracking Android pattern lock using acoustic signals. In *the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. ACM, 2018.

[195] Yanzi Zhu, Zhujun Xiao, Yuxin Chen, Zhijing Li, Max Liu, Ben Y Zhao, and Haitao Zheng. Et tu Alexa? When commodity WiFi devices turn into adversarial motion sensors. In *Network and Distributed Systems Security (NDSS '20) Symposium*, 2020.

[196] Qin Zou, Yanling Wang, Qian Wang, Yi Zhao, and Qingquan Li. Deep learning-based gait recognition using smartphones in the wild. *IEEE Transactions on Information Forensics and Security*, 15:3197–3212, 2020.

APPENDICES

Appendix A

Multi-user IA Complementary Results

A.1 Q Selection for Kalman Filter

In the first use case in § 5.6, we adopt the Kalman Filter settings from CORMORANT [74]. The Kalman filter assumes there is Gaussian noise in the state transition, which is modeled with a noise covariance matrix Q. CORMORANT highlighted the significant impact of the value selection of Q: A large Q has a smaller confidence in the system model and a larger confidence in the observations, which is desired by the score fusion purpose. To ensure the Kalman filter based score fusion method is optimized, we tested several Q values and compared their accuracy values using the same settings of the first use case, where $n_v = n_a = 3$. Fig. A.1 shows the decision-level metrics of different Q's. In general, a large Q results in a better FAR. An extremely small $Q = 0.001$ leads to high FAR and FIR. However, FRR and FIR do not decrease with larger Q. To balance the three metrics, we choose $Q = 0.25$ in our experiments, which has the lowest average FRR (0.05) plus FIR (0.06) with a relatively low FAR (0.18).

A.2 Gini Coefficient for Use Case 1

The Gini Coefficient (GC) is calculated between the area between the Lorenz Curve and the Line of Equality. For evaluating IA systems, Lorenz Curve plots percentiles of the users

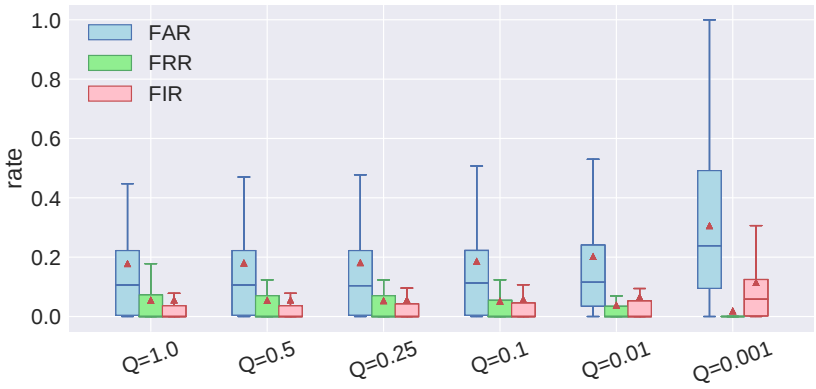
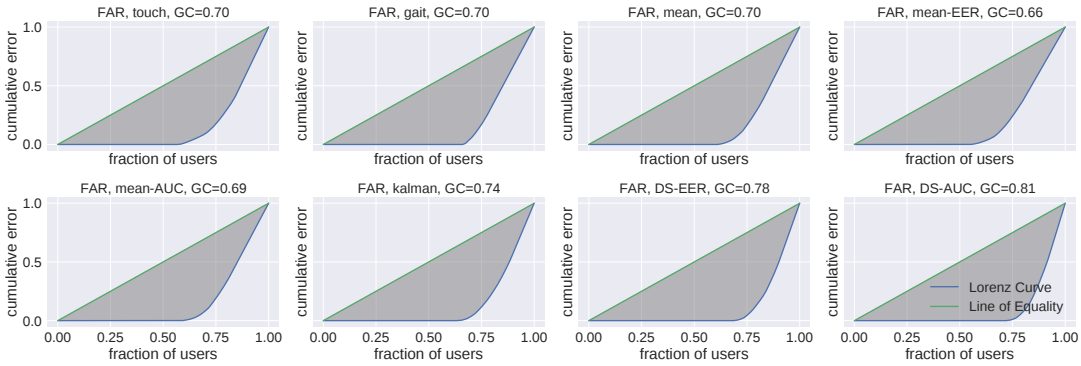
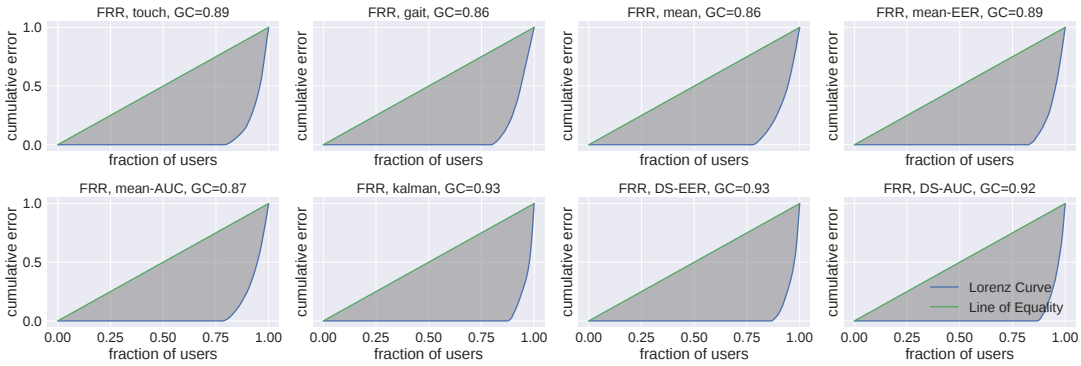


Figure A.1: Decision-level FAR, FRR, and FIR of Kalman filter based fusion methods at different Q's ($n_v = n_a = 3$)

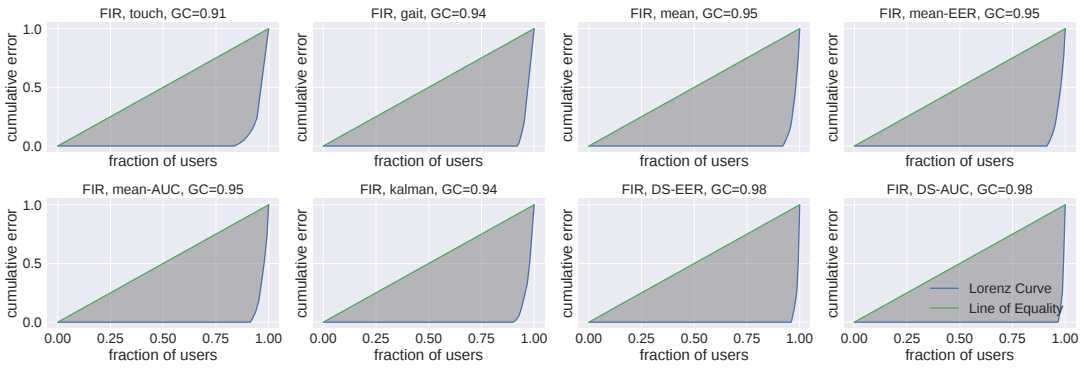
on the x-axis according to error rate and plots cumulative error rate on the y-axis [41]. A point (x, y) on the curve indicates the normalized total error rate y contributed by the bottom x users. The Line of Equality is a straight diagonal line with a slope of 1, which represents that all users contribute to the same error rates. In Fig. A.2, we present the Lorenz Curve and the GC of all eight methods in the first group of evaluation tasks for the first use case. If we compare the D-S theory based fusion methods to other methods, we can observe fewer users contribute to more errors from the figure. For FAR, it means that the system may mis-classify a few "very successful" attackers as legitimate users. However, it is also possible that the system rejects an attacker at a certain time during continuous authentication. Thus, we also measure session-level metrics and detection latency to capture such situations. For FRR and FIR, the Lorenz Curve does not change as significantly as FAR, which means the error distribution remain similar among different methods. We can infer that score fusion strategies are effective in reducing random errors. To further improve the accuracy and reduce systematic error, a possible avenue is to incorporate new modalities.



(a) FAR



(b) FRR



(c) FIR

Figure A.2: Lorenz Curve and Gini Coefficient of use case 1, $n_v = n_a = 3$