Sharing without Scaring: Enabling Smartphones to Become Aware of Temporary Sharing

Jiayi Chen and Urs Hengartner, *University of Waterloo;*
Hassan Khan, *University of Guelph*

https://www.usenix.org/conference/soups2022/presentation/chen

This paper is included in the Proceedings of the Eighteenth Symposium on Usable Privacy and Security (SOUPS 2022).

August 8–9, 2022 • Boston, MA, USA

978-1-939133-30-4

Sharing without Scaring: Enabling Smartphones to Become Aware of Temporary Sharing

Jiayi Chen
University of Waterloo

Urs Hengartner
University of Waterloo

Hassan Khan
University of Guelph

Abstract

Smartphone owners often hand over their device to another person for temporary sharing, such as for showing pictures to a friend or entertaining a child with games. This device sharing can result in privacy concerns since the owner's personal data may become vulnerable to unauthorized access. Existing solutions have usability problems and neglect human factors of sharing practices. For example, since device sharing implies trust between people, explicitly hiding data may signal mistrust. Besides, an owner may fail to enable a sharing-protection mechanism due to forgetfulness or lack of risk perception. Therefore, we propose device sharing awareness (DSA), a new sharing-protection approach for temporarily shared devices, which detects a sharing event proactively and enables sharing protection *subtly*. DSA exploits natural handover gestures and behavioral biometrics for proactive sharing detection to transparently enable and disable a device's sharing mode without requiring explicit input. It also supports various access control strategies to fulfill sharing requirements imposed by an app. Our user study evaluates handover detection over 3,700 data clips (n=18) and comprehensive device sharing processing over 50 sessions (n=10). The evaluation results show that DSA can accurately detect handover gestures and automatically process sharing events to provide a secure sharing environment.

1 Introduction

Prior research shows that it is common for smartphone users to temporarily share their devices with another person for trust and convenience [20, 31]. For example, a smartphone user may show pictures stored on the phone to a friend or hand over the device to a child to play games. This device sharing can result in negative experiences due to all-or-nothing access control [12]. Liu et al. [27] report that 86% of the participants in their user study always kept their phone in sight when sharing, which puts an extra burden on the owner and may make the sharee feel mistrusted. (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.) Hang et al. [12] report that the majority of participants in their user study wanted the ability to share only specific apps and features.

Existing solutions for temporary device sharing emphasize *how to* impose access restrictions on sensitive apps and data during sharing. They allow the owner to add a guest account [17], pin an app [15, 16], or launch apps with limited features (e.g., a camera app without a view of existing pictures) when the device is locked. However, most solutions require an explicit action from the owner before sharing the device. Vulnerabilities arise when humans are forgetful [36] or lack risk perception of certain situations [4]. Owners may forget to switch to the guest account or to pin an app before sharing. Furthermore, sharing behavior is closely related to trust [38]. An owner explicitly enabling these solutions signals mistrust for device sharing between the owner and sharee [1, 13, 20, 30, 32]. 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.

We introduce device sharing awareness (DSA) to address *when to* enable device sharing solutions. DSA should: 1) *proactively detect device sharing* instead of requiring an owner to remember performing a predefined action, 2) *continuously identify the owner* to prevent unauthorized access and ensure that only the owner has full access, 3) *be exception-resistant* to automatically handle possible false detection or exceptions and mitigate the exposure of sensitive resources.

USENIX Symposium on Usable Privacy and Security (SOUPS) 2022.
August 7–9, 2022, Boston, MA, United States.

For an outcome of device sharing awareness, DSA should provide *flexible access control* to choose an appropriate strategy based on the current app type.

To fulfill the above requirements, DSA 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 verifies 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 [40], DSA generated only 0.9 false positives per hour of continuous device use. For an average daily smartphone use of about three hours [11], DSA will generate about three false positives a day. We also tested the device sharing processing ability of DSA with a popular touch-based implicit authentication (IA) solution [10], which includes 50 complete device sharing sessions. DSA succeeded in detecting handover gestures in 48 sessions, and automatically handled 41 sessions without exceptions while its exception processing additionally recovered six sessions. DSA adopts adaptive sensing, and 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.

Our main contributions include: 1) A demonstration of the ability of low-cost proactive sharing detection using smartphone built-in sensors. 2) An open-source solution for Android[1] that secures sensitive data during device sharing, while mitigating human factors of forgetfulness and mistrust. 3) An extensive evaluation to demonstrate its practicality in terms of accuracy to detect sharing and battery consumption. 4) A public, labelled motion sensor dataset with over 3,700 sharing gesture clips for the research community.

2 Related work

Device sharing surveys. According to recent surveys, mobile device sharing is common in people's daily life [31] and even a systemic practice in some regions (e.g., South Asia [1, 4, 35]). Reasons for device sharing are not limited to economic consideration, help, convenience, or access to specific features. As a social and cultural practice, it is also driven by the need for maintaining social relationships and signaling trust among people [1, 4, 30]. However, as revealed by extensive qualitative studies [1, 20, 31], people still have privacy concerns over sharing their mobile devices given 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.

Access control for device sharing. Researchers have studied smartphone owners' security and privacy concerns with sharing different apps and called for access control mechanisms for device sharing [12, 20, 24, 31, 35]. Studies [12, 13, 20] show that all-or-nothing access control cannot meet the need for device sharing from both security and convenience aspects. Koushki et al. [24] show that app- or task-level access control can significantly reduce unnecessary or missed interventions compared to all-or-nothing access control. xShare [27] enables the owner to specify the resources to share and offers a restricted mode for the sharee but requires modification to the operating system. DiffUser [33] establishes a multi-user security model for Android smartphones, but it requires creating different accounts to apply different access control rules. SnapApp [6] adopts 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 [37] considers 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 comparison, DSA enables smartphones to proactively detect device sharing without being manually activated by an owner and provides flexible access control.

Trust. As trust is an important motivation of device sharing, we also need to take trust into account when designing device sharing solutions. A guest account for socially close sharees is deemed inappropriate since it signals mistrust [20, 30]. Explicitly hiding certain apps may also imply a lack of trust [1, 4]. Recent device sharing proposals have explored how to protect sensitive resources while not compromising trust. Seyed et al. [38] propose a modular smartphone comprising of multiple access-controlled hardware components to address the trust and convenience issues of device sharing. PrivacyShield [34] provides a subtle just-in-time privacy provisioning system, which enables the owner to quickly enable an access control rule by entering pre-defined touch gestures. Ahmed et al. [2] adopt 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, DSA takes control of the entire sharing process *proactively* and *auto-*

[1]The source code and the dataset are available at `https://github.com/cryspuwaterloo/DSA-Framework`

matically so that smartphone users do not need to specify or enable access control rules in front of a sharee. Note that DSA emphasizes the subtlety of *enabling* a device sharing solution. It does not try to hide from a sharee that the device is currently in a restricted environment, which is a design problem [1]. Achieving this goal reliably requires tremendous efforts of app developers to redesign their apps [2].

Activity detection. DSA uses smartphone motion sensors to detect a sharing event based on hand movements. Existing work [3, 28, 39] focuses on using motion sensors to detect specific hand gestures. DSA detects natural device handover gestures continuously so that owners do not need to remember to perform a pre-defined gesture for device sharing. Vaizman et al. [41, 42] propose a multi-modal system that uses various sensors on smartphones and smartwatches to recognize a person's behavioral context in natural environments, which is close to our purpose of detecting a sharing gesture in the wild. However, unlike behavioral contexts, such as walking and running, a sharing gesture lasts only one to two seconds and is not a repetitive or periodical activity. Nevertheless, we follow the feature selection from existing work [25, 41] to train our gesture detection model.

3 Device Sharing Awareness

3.1 Modeling temporary device sharing

Temporary device sharing is a social activity where a device owner shares certain resources on the phone with one or several sharees. The device sharing scenario targeted by our work is: The device is initially with its owner, and the owner directly hands it over to a sharee as a signal of granting temporary access. During sharing, a sharee should not have access to sensitive resources, including personal data (e.g., messages, photos) and critical operations (e.g., deleting files). We do not study device sharing where the device is initially not with its owner, or where a sharee can access the device indirectly (e.g., the owner puts the unlocked device on a table to pass the device) or without the owner's presence. Traditionally, this kind of sharing is enabled with separate user accounts or PIN sharing [1, 30]. We discuss this case in § 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.

3.2 Limitations of existing sharing 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 [7], Norton App Lock [26]) 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. For example, a browser app provides the essential web browsing function and may store the owner's passwords for auto-filling. 3) *App pinning* (e.g., Android Screen Pinning [16], iOS Guided Access [15]) 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., App Vault [18], Private Space [14]) 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 [2]: 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).

There are still several gaps between the current practices and a desired device sharing solution:

1. **Lack of subtlety.** Ahmed et al. [1] 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 [4] 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 the browser app example, a sharee should be allowed to browse the web without accessing the owner's data.

As existing device sharing solutions [2, 6, 34] mainly address *how* to protect sensitive resources from unauthorized access during sharing, we focus on a novel perspective, device sharing awareness (DSA), to address *when* to (de-)activate such solutions. The device should be able to detect device sharing and identify the owner proactively and transparently. 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.

3.3 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 [15, 16, 34]). 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. In this paper, we mainly investigate the detection of the device handover gesture.

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. Here, verifying the user's identity is essential: 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. DSA should be able to determine if the current user is the owner proactively and transparently. It can be achieved by continuous and implicit authentication: A device can distinguish the device owner from other people based on biometrics, including continuous facial recognition [8, 29], voice recognition [44], or implicit authentication (IA) based on behavioral biometrics [19, 22]. 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 user 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.

3.4 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 [13, 27], 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.

3.5 Threat model

Device sharing involves two kinds of roles: an owner and one or more sharees. A malicious owner is out of the scope of DSA since the owner can disable DSA and launch attacks on a sharee (e.g., accessing a sharee's account that is not properly logged out after sharing, or sniffing passwords the sharee enters on a website). Instead, we focus on attacks from sharees. We classify sharees into two categories: A *benign sharee* uses only 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 implicit authentication, and attempts to bypass them. A malicious sharee may also know of the existence of our proposed solution and launches attacks accordingly.

4 System design

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.

4.1 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. Fig. 1 shows the state transition among the three states. Existing app pinning solutions fully rely on manual operations to switch among the three states (see Fig. 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.

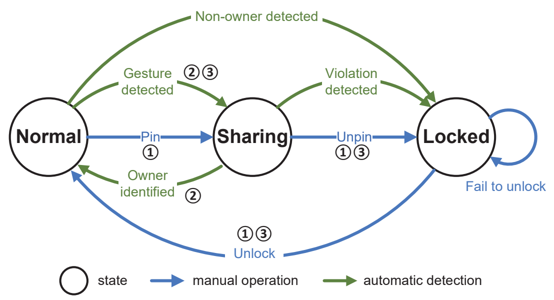


Figure 1: State transition of device sharing. Three sharing loops: ① explicit sharing loop (manual option), ② implicit sharing loop (handover gesture + owner detection), ③ hybrid sharing loop (handover gesture + manual unlock).

Following § 3.3, we introduce an implicit sharing loop (see Fig. 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 § 4.6). DSA allows a hybrid sharing loop (see Fig. 1 Loop ③) where DSA detects a handover gesture to move into sharing state while the owner or sharee have to manually end sharing, and explicit authentication is required to move back into state "normal".

4.2 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

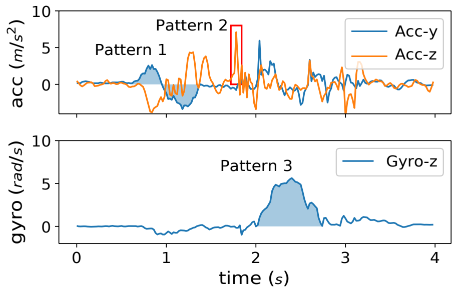


Figure 2: Handover patterns. 1. (horizontal movement): the device travels a distance in the xy-plane, where acceleration follows a sine curve like pattern; 2. (spike): When the sharee catches the device, a spike appears on the z-axis of acceleration; 3. (rotation): the device is rotated either by the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

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 Fig. 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, overlapping segments, where the sampling rate is f_s, the segment length is d seconds (equal to $f_s \cdot d$ samples), and the interval between the start of two consecutive segments is p seconds. (Fig. 7 shows an example of segmentation.) The choices of d and p affect the detection performance. If d is too small, it is hard to capture the handover patterns from a data segment; if d and p are too large, it takes more time to capture handover. We investigate the impact of different settings on detection performance in § 5.

After segmenting the raw data, we extract the following features for each segment: We first calculate the magnitude of linear acceleration, $m = \sqrt{a_x^2 + a_y^2 + a_z^2}$. Together with each axis of linear acceleration and gyroscope data, we use common statistics widely adopted in gesture detection [3] and activity recognition [25] including: average, standard deviation, maximum, 25th percentile, median, 75th percentile, sum, double integration, and range. Also, we measure root mean square (RMS) [9] of the readings to capture time-domain wavelet patterns: $\mathrm{RMS}(\mathbf{v}) = \sqrt{(v_0^2 + v_1^2 + \cdots + v_{n-1}^2)/n}$, where $\mathbf{v} = \{v_0, v_1, \ldots v_{n-1}\}$ is a series of n sensor readings. Value and time entropy [41] measure sudden changes in a signal. We calculate the value entropy by quantizing all magnitude values to a 20-bin histogram for a moderate granularity. For time entropy, we normalize all sensor readings to form a probability distribution and calculate $H(|\mathbf{v}|) = -\sum_{i=0}^{n} \frac{|v_i|}{\sum|\mathbf{v}|} \log \frac{|v_i|}{\sum|\mathbf{v}|}$. Furthermore, to include correlations between different axes, we calculate the correlation coefficient between every two axes. In total, there are 87 features.

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 labelling is challenging since it may need a user's feedback to position a sharing event. Besides, our evaluation results in § 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. We adopt an adaptive sensing strategy to reduce 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 of 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 Owner detection

Owner detection is provided by continuous and implicit authentication (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 in 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 [10], whose reported equal error rate is below 4%, as the default scheme in our evaluation. Touchalytics extracts 29 features from touch events to capture behavior related to acceleration, velocity, duration, orientation, width, pressure, and trajectory length. It performs classification for each touch event and authenticates the user based on the results of several consecutive touch events. According to Khan et al. [22], the battery consumption of touch-based IA is low enough for continuous owner detection.

4.4 Access control for device sharing

For improved usabililty, 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 [5]. 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.

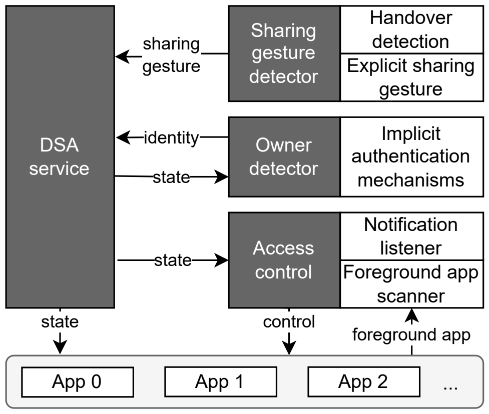


Figure 3: DSA architecture.

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 [1, 2], 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.

4.5 DSA workflow

Fig. 3 shows the architecture of DSA. The high-level workflow of DSA follows the three stages introduced in the device sharing model in § 3.1.

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 sharee is authorized to access the current foreground app. 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. 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.

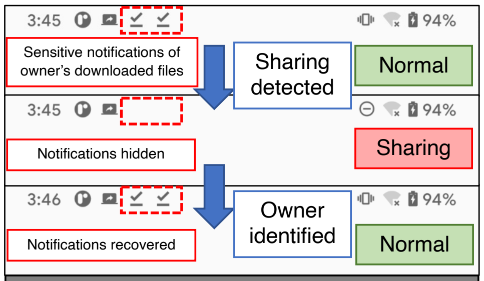
4.6 Exception processing

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 § 5.3), we investigated the exceptions that DSA may encounter and how efficiently it handled these exceptions.

Non-owner user detected in state "normal". When the owner detector detects a non-owner user, 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 mechanism; 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 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



```
15:45:25.300 I/HandoverDetection: Sharing detected
15:45:25.353 I/AccessControl: Sharing mode
15:45:25.354 I/DSAService: Broadcast sharing state
15:46:19.588 I/OwnerDetection: IA result: owner
15:46:19.588 I/DSAService: Return confirmed
15:46:19.660 I/AccessControl: Normal mode
```

Figure 4: DSA Service Example: 1) At 3:45pm, 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 3:46pm, 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 logging purposes); the second icon indicates that DSA Service is running; the third and fourth icons are the sensitive notifications.

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 re-authenticate. In § 5.2, we evaluate how the owner detector addresses false positives.

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 sharing mode by acquiring the sharing state from the DSA service at startup.

4.7 Implementation

We implement our demo DSA solution on Android as a service and release the source code[1]. 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. Fig. 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. We can see that DSA automatically hid the sensitive notifications after detecting a handover gesture and recovered them once the user changed back to the owner. During this process, the owner did not need to manually enable and disable the sharing mode.

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 received approval from our IRB for the user study reported in this work.

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 consisted 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. For the detection of handover gestures, we used Support Vector Machines (SVM) and Neural Networks (NN) to train the gesture detection model and use it for classification. Considering the NN model's superior performance and the increasingly mature support for NN on today's smartphones, we adopted NN in our evaluation. The input layer of the NN is the feature vector (size=87) 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. We set the number of epochs as 120 and the batch size as 128. For the training set, the positive instances were from handover gestures, and the negative instances were from movements sharing similar patterns with handovers. Given the low frequency of sharing events in practice, an imbalanced training set reflecting the actual distribution may make the model focus on detecting non-handover gestures [21]. Thus, we adopt a balanced training set where positive and negative instances are evenly distributed, and use 10% of the data for validation.

Metrics. Handover detection involves segmenting motion data, classifying each segment, and making decisions based

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 1: Segment-level experiments: Per-user models.

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 2: Segment-level experiments: Cross-user models.

on a number of consecutive segments. For segment-level classification, we evaluate the classifier performance based on its receiver operating characteristic (ROC) curve and use area under curve (AUC) and equal error rate (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.

5.2 Evaluation of handover detection

The first part of our user study evaluated the accuracy of handover detection. Training a gesture detection model requires both handover (i.e., positive) and non-handover (i.e., negative) data to distinguish handover gestures from other movements. Thus, the user study involved two-participant handover tasks and single-participant non-handover tasks. In the handover tasks, participants were asked to hand over a smartphone from either their left or right hand in two directions: 1) Face-to-face: the owner was in front of the sharee. 2) Side-by-side: the owner was next to the sharee. 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 pair of participants, one participant handed over the device to the other at least 20 times per direction. Then, they swapped roles and repeated. In the non-handover tasks, we recorded motion data for activities having similar patterns with handover, such as switching hand, putting the device down, rotating the device, and random movements with combining device rotations and movements in different directions. All participants completed each single-participant task (e.g., switching hand) 20 times. Each data clip of handover or non-handover events lasts 5s to 10s.

In total, we collected 2044 positive and 1737 negative clips.

5.2.1 Cross-user experiments

A pre-trained gesture detection model should work on a new user (i.e., low user dependence) without retraining. Thus, we evaluate the cross-user performance of gesture detection from the perspectives of AUC and EER. A high AUC and a low EER indicate that a model can distinguish handover gestures from these activities better. We split each data clip into segments ($d = 2s, p = 1s$). For positive events, we focus on data segments that have 50% overlap with this time interval

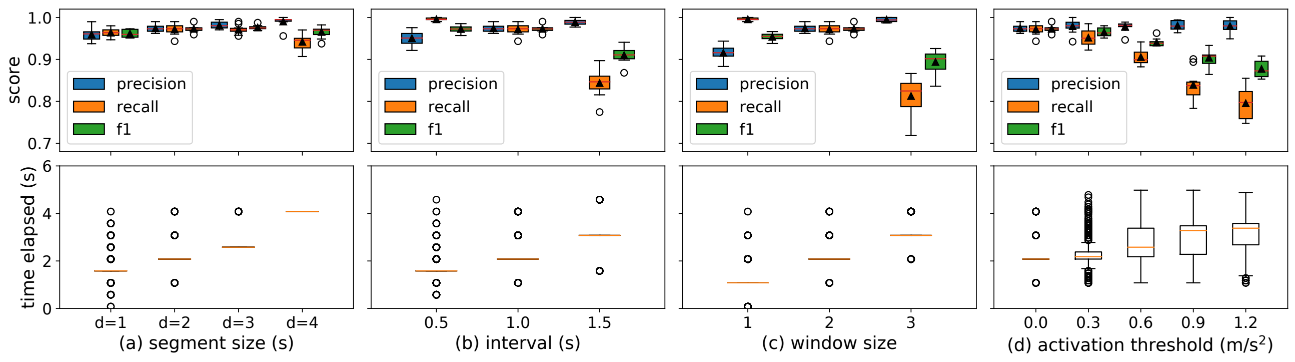


Figure 5: Event-level experiments: impact of different parameters on the detection performance and latency.

and label them as positive segments.We label all the segments from negative events as negative segments.

First, a cross-user model, which is pre-trained with hybrid data from multiple users, should have comparable accuracy as a per-user model. As a reference, we run a 10-fold cross validation to test the performance of per-user models for 12 participants using the same Google Pixel phone. We split each user's data in 10 subsets and use one subset as the testing set and the other nine subsets as the training set for each fold. As shown in Table 1, the AUCs of all models are above 0.96 while the EERs are below 10%. Then, the cross-user model is trained with multiple users' motion data. 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 2 shows that the cross-user model can still provide a high AUC 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 § 5.2.2) and additionally apply IA based owner detection (see § 5.3) to further mitigate false detection. In summary, the results shows that the gesture detection model can recognize handover gestures across different users, which imply its low user dependence.

In the appendix, we present cross-device evaluation results to show model transferability to different devices.

5.2.2 Impact of different settings

As introduced in § 4.2, the choice of segment size d and interval p affects the detection accuracy. Besides, we adopt a sliding window-based strategy, where handover detection is performed over w segments to balance accuracy and detection delay. Moreover, we use adaptive sensing to save battery (see § 4.2), and the choice of its activation threshold θ may affect the detection performance. 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 d and a smaller interval p provide better ability to cover a sharing gesture. We tested four d's ($1s, 2s, 3s, 4s$) and set p to achieve a 50% segment overlap. We set $w = 2$. In Fig. 5(a), $d = 3s$ provides higher precision compared to $d = 2s$, but it takes longer to make a detection. To balance latency and accuracy, we set $d = 2s$. Then, we test three different p's. Fig. 5(b) shows that a shorter interval has higher recall, but lower precision. A larger overlap allows more classifications in the same period to improve recall and capture a gesture earlier.

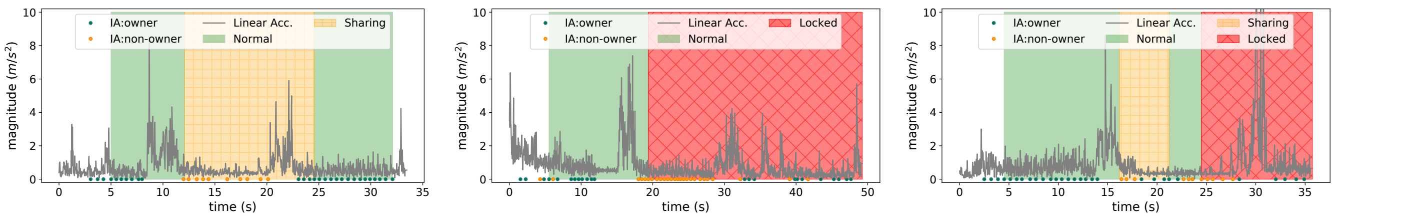
Window size. Considering the length of a sharing gesture, we change the window size from one to three segments at $d = 2s$ and $p = 1s$. Fig. 5(c) shows f1-score reaches the highest (median: 98%) and the average elapsed time is only 2s when $w = 2$. When $w = 3$, the average recall decreases to 81%. When $w = 1$, 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 $m > \theta$, high frequency mode is activated; if $m \leq 0.1m/s^2$, 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, $0.3m/s^2$, $0.6m/s^2$, $0.9m/s^2$, $1.2m/s^2$. Fig. 5(d) shows that recall drops with higher θ. Due to the mode switch delay, it is likely to miss data at the beginning of a gesture. Nevertheless, when $\theta = 0.3m/s^2$, recall is still acceptable (mean: 95%).

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

5.2.3 False positive evaluation

We train the gesture detection model using the training data from all participants in § 5.2.1 and adopt the settings summarized in § 5.2.2. We evaluate the long-term false positive rate of handover gesture detection using the HMOG



(a) Example 1: error-free session. (b) Example 2: gesture detection failure. (c) Example 3: IA false acceptance.

Figure 6: 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.

dataset [40,43]. This 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. Since handover detection runs only when the screen is on, the number of false positives is correlated to the daily screen time of a user. For an average daily screen time of three hours [11], handover detection may produce two to three false positives in a day. Even if a false positive makes DSA move to state "sharing" falsely, DSA switches back to state "normal" once the owner's identity is confirmed, which mitigates the false positive. Thus, the false positive rate of handover detection is acceptable for daily usage. For future work, we will conduct a longitudinal study on the impact of false positives on usability.

5.3 Device sharing processing

The second part of our user study 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 captured potential exceptions.

Task description. We adopted a touchscreen input based IA scheme [10] (i.e., touch-based IA) and used a (m,n)-sliding-window-based strategy: If m out of n swipes are accepted as the owner's, IA will accept the current user as the owner. 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 trained a model with the training data from the controlled experiments in § 5.2 and use the default settings. In each session, a group of two participants was asked to perform a web page sharing task: the owner shared a web page and handed the phone 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 when the phone is in their possession. Once they completed the reading task, they swapped their roles and repeated the above process. Each group contributed to 10 sessions. Given that the amount of time for temporary device sharing is usually limited [31], most sharing events in our study lasted from 30 seconds to one minute. Short sharing events require DSA to detect the starting and end of a sharing event rapidly. We did not specify the position of each participant and how they handed over the device so that participants could hand over the device in their natural manner. In total, we collected 50 device sharing sessions from five groups of participants for analysis.

Results. We counted the sessions with exceptions of either handover detection or IA among the 50 sessions. We observed three exception types: 1) failure in detecting handover gestures to enable the sharing mode: 2 sessions, 2) IA falsely accepting a sharee as an owner: 6 sessions, and 3) failure in detecting the end of a sharing event: 1 session. Therefore, DSA completed the implicit sharing loop in 41 sessions without explicit inputs from the owner. 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. Fig. 6(a) shows an example of a session without exceptions: 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 detected the owner at 24.5s and switched back to the normal state.

Exception processing. We recorded all sessions with exceptions and analyzed 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 Fig. 6(b)). 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 Fig. 6(c), 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. 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. 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”. 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.

6 Discussion

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. Our battery consumption evaluation is preliminary. Since DSA performs sharing detection only when the screen is on, we chose a small time frame. In the future, we will evaluate battery consumption with longer time frames and its impact on device usage.

Defending against unauthorized access. Given the observed latency of handover detection, we conclude that DSA can swiftly activate the sharing mode, and a sharee can hardly conduct effective attacks during such a short interval. Even if handover detection fails, owner detection can block a sharee upon negative IA results. As observed in § 5.3, owner detection was limited by the performance of its IA scheme. False acceptance may temporarily deactivate the sharing mode so that a sharee can move to sensitive apps at this moment. The exception processing of DSA will 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 [23]). A promising countermeasure is to adopt multiple modalities (i.e., multimodal IA) so that the failure of one modality is not likely to make owner detection fail. Thus, how to incorporate multimodal IA into DSA will be our future work.

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 shares their PIN/password with a sharee in advance so that the sharee can unlock the device without the owner’s presence. DSA 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 [2] 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. We list the following limitations in the evaluation of DSA: First, the evaluation of handover detection did not cover some conditions (e.g., from standing to sitting, in a vehicle) and edge cases (e.g., non-handover sharing actions via a table). For edge cases, 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). Second, 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 behavior during the tasks in the second part. Third, our analysis focused on how DSA handles sharing a 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.

7 Conclusion

We present DSA, a device sharing awareness solution for temporary smartphone sharing. 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.

Acknowledgements

This research has been supported by the Waterloo-Huawei Joint Innovation Laboratory. The authors would like to thank the anonymous reviewers and shepherd for providing insightful comments and feedback to improve the paper.

References

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

[2] 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 *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2019.

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

[4] 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.

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

[6] 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 *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 3736–3747, 2016.

[7] Samsung Electronics. S Secure. https://galaxystore.samsung.com/prepost/000004637448.

[8] 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)*, pages 1687–1691. IEEE, 2015.

[9] 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.

[10] 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.

[11] 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.

[12] Alina Hang, Emanuel Von Zezschwitz, Alexander De Luca, and Heinrich Hussmann. Too much information! User attitudes towards smartphone sharing. In *Proceedings of the 7th Nordic Conference on Human-Computer Interaction: Making Sense Through Design*, pages 284–287, 2012.

[13] 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 *Proceedings of the Eighth Symposium on Usable Privacy and Security*, pages 1–11, 2012.

[14] Ltd. Huawei Device Co. Create a privatespace for your private data. https://consumer.huawei.com/en/support/content/en-us15834600/.

[15] Apple Inc. Guided access. https://support.apple.com/en-us/HT202612.

[16] Google Inc. Android pin & unpin screens. https://support.google.com/android/answer/9455138?hl=en.

[17] Google Inc. Manage guests and users. https://support.google.com/nexus/topic/6126546?hl=en&ref_topic=3416294.

[18] Xiaomi Inc. App vault. https://play.google.com/store/apps/details?id=com.mi.android.globalminusscreen.

[19] Markus Jakobsson, Elaine Shi, Philippe Golle, and Richard Chow. Implicit authentication for mobile devices. In *Proceedings of the 4th USENIX conference on Hot topics in security*, pages 9–9, 2009.

[20] Amy K Karlson, AJ Bernheim Brush, and Stuart Schechter. Can I borrow your phone? Understanding concerns when sharing mobile phones. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1647–1650, 2009.

[21] Harsurinder Kaur, Husanbir Singh Pannu, and Avleen Kaur Malhi. A systematic review on imbalanced data challenges in machine learning: Applications and solutions. *ACM Computing Surveys (CSUR)*, 52(4):1–36, 2019.

[22] Hassan Khan, Aaron Atwater, and Urs Hengartner. Itus: An implicit authentication framework for Android. In *Proceedings of the 20th annual international conference*

on Mobile computing and networking, pages 507–518, 2014.

[23] Hassan Khan, Urs Hengartner, and Daniel Vogel. Augmented reality-based mimicry attacks on behaviour-based smartphone authentication. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 41–53, 2018.

[24] Masoud Mehrabi Koushki, Yue Huang, Julia Rubin, and Konstantin Beznosov. Neither access nor control: A longitudinal investigation of the efficacy of user Access-Control solutions on smartphones. In *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, August 2022. USENIX Association.

[25] 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.

[26] Norton Labs. Norton app lock. https://play.google.com/store/apps/details?id=com.symantec.applock.

[27] Yunxin Liu, Ahmad Rahmati, Yuanhe Huang, Hyukjae Jang, Lin Zhong, Yongguang Zhang, and Shensheng Zhang. xShare: Supporting impromptu sharing of mobile phones. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, pages 15–28, 2009.

[28] 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.

[29] 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)*, pages 2991–2995. IEEE, 2016.

[30] Diogo Marques, Tiago Guerreiro, Luís Carriço, Ivan Beschastnikh, and Konstantin Beznosov. Vulnerability & blame: Making sense of unauthorized access to smartphones. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2019.

[31] 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 *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 5921–5932, 2016.

[32] 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 *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 645–654, 2010.

[33] 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*, pages 1012–1017. IEEE, 2009.

[34] 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. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(2):1–38, 2018.

[35] 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 *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, pages 127–142, 2018.

[36] Stuart E Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. The emperor's new security indicators. In *IEEE Symposium on Security and Privacy*, pages 51–65. IEEE, 2007.

[37] 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*, pages 130–137. Springer, 2010.

[38] Teddy Seyed, Xing-Dong Yang, and Daniel Vogel. A modular smartphone for lending. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, pages 205–215, 2017.

[39] Sheng Shen, He Wang, and Romit Roy Choudhury. I am a smartwatch and I can track my user's arm. In *Proceedings of the 14th annual international conference on Mobile systems, applications, and services*, pages 85–96, 2016.

[40] 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.

[41] Yonatan Vaizman, Katherine Ellis, and Gert Lanckriet. Recognizing detailed human context in the wild from smartphones and smartwatches. *IEEE Pervasive Computing*, 16(4):62–74, 2017.

[42] 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 *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018.

[43] 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 *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, pages 358–359, 2014.

[44] Linghan Zhang, Sheng Tan, Jie Yang, and Yingying Chen. VoiceLive: A phoneme localization based liveness detection for voice authentication on smartphones. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1080–1091, 2016.

A Appendix

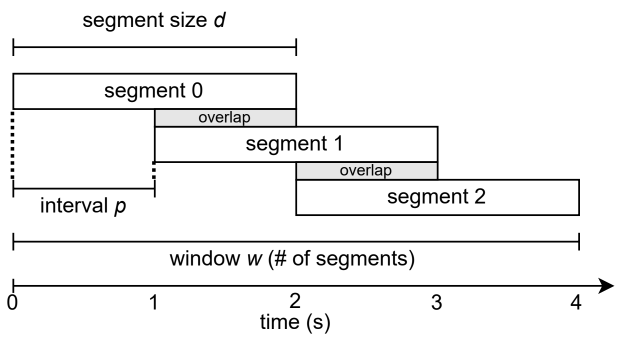
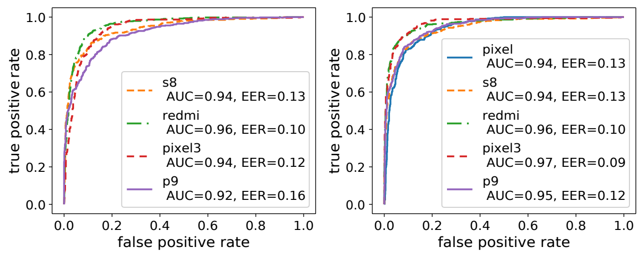


Figure 7: Data segmentation. In this example, segment size $d = 2s$, interval $p = 1s$, and windows size $w = 3$ segments.



(a) One-to-multi test (Pixel) (b) Multi-to-one test

Figure 8: Inter-device experiment.

Cross-device experiments. 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 Fig. 8(a), the model trained with one phone's data shows a consistently good performance on the other phones, where the AUCs are always above 0.92 and the EERs are around 10 to 16%. (2) We use mixed training data from four phone models to train the model and test it on the fifth phone. As shown in Fig. 8(b), the cross-device model provides a consistently good performance across different phone models.