PrivacyShield: A Mobile System for Supporting Subtle Just-in-time Privacy Provisioning through Off-Screen-based Touch Gestures

SAUMAY PUSHP[*], School of Computing, KAIST, South Korea
YUNXIN LIU, Microsoft Research, China
MENGWEI XU, School of Electronics Engineering and Computer Science, Peking University, China
CHANGYOUNG KOH, School of Computing, KAIST, South Korea
JUNEHWA SONG, School of Computing, KAIST, South Korea

Current *in-situ* privacy solution approaches are inadequate in protecting sensitive information. They either require extra configuration effort or lack the ability to configure user desired privacy settings. Based on in-depth discussions during a design workshop, we propose *PrivacyShield*, a mobile system for providing subtle just-in-time privacy provisioning. PrivacyShield leverages the screen I/O device (screen digitizer) of smartphones to recognize gesture commands, even when the phone's screen is turned off. Based on gesture command inputs, various privacy-protection policies can be configured on-the-fly. We develop a novel stroke-based approach to address the challenges in segmenting and recognizing gesture command inputs, which helps the system in achieving good usability and performance. PrivacyShield also provides developers with APIs to enable just-in-time privacy provisioning in their applications. We have implemented an energy efficient PrivacyShield prototype on the Android platform, including smartphones with and without a low-power co-processor. Evaluation results show that our gesture segmentation algorithm is fast enough for real-time performance (introducing less than 200ms processing latency) and accurate (achieving an accuracy of 95% for single-character gestures and 89% for even three-character gestures). We also build a non-touch-screen-based just-in-time privacy provisioning prototype called the *wrist gesture method*. We compare the performance of the two prototypes by doing a 6-week field study with 12 participants and show why a simplistic solution falls short in providing privacy configurations. We also report the participants' perceptions and reactions after the field study.

CCS Concepts: • **Human-centered computing → Ubiquitous and mobile computing systems and tools**; • **Security and privacy → Usability in security and privacy**;

Additional Key Words and Phrases: Mobile Systems, Gesture segmentation, Gesture recognition, Just-in-time privacy provisioning, In-situ privacy, In-situ usability

ACM Reference Format:
Saumay Pushp, Yunxin Liu, Mengwei Xu, Changyoung Koh, and Junehwa Song. 2018. PrivacyShield: A Mobile System for Supporting Subtle Just-in-time Privacy Provisioning through Off-Screen-based Touch Gestures. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 2, Article 76 (June 2018), 38 pages. https://doi.org/10.1145/3214279

[*]Part of this work was done when the author was in Microsoft Research, Beijing, China.

Authors' addresses: Saumay Pushp, School of Computing, KAIST, 291 Daehak-ro, Yuseong-gu, Daejeon, 34141, South Korea, saumay@nclab.kaist.ac.kr; Yunxin Liu, Microsoft Research, 12F, Microsoft Building 2, Danling Street, Haidian District, Beijing, China, yunxin.liu@microsoft.com; Mengwei Xu, School of Electronics Engineering and Computer Science, Peking University, 1434, Science Building No.1, Yiheyuan Road No.5, Haidian District, Beijing, 100871, China, xumengwei@pku.edu.cn; Changyoung Koh, School of Computing, KAIST, 291 Daehak-ro, Yuseong-gu, Daejeon, 34141, South Korea, changyoung@nclab.kaist.ac.kr; Junehwa Song, School of Computing, KAIST, 291 Daehak-ro, Yuseong-gu, Daejeon, 34141, South Korea, junesong@nclab.kaist.ac.kr.

1 INTRODUCTION

Present day smartphones provide a gateway to a wealth of users' private information, e.g., photos, videos, text messages, and emails. To protect such private information, users use simple screen lock solutions like a pass-code or a pattern lock. However, when users share their devices, *in-situ*, with people [29][34], e.g., close friends or family members, they inadvertently put their private information at risk. They either become prey to an information leaks from incoming message notifications which are promptly notified to the users via the smartphone screen [38] or they find themselves in situations where they expose their photos to a person who is gazing at the device screen to see a particular photo(s) [50]. In such cases, being snooped on by socially-close people is a trespassing of one's privacy in its most basic sense [37].

Current state-of-the-art privacy provisioning solution approaches (or methods) for preventing *in-situ* privacy leaks involves on-screen [11][9] and off-screen [13] access to notification settings, or the use of multi-user modes (or guest mode) [34][22][11][9]. However, these solutions/approaches have the following limitations. First, users need to act fast to access a desired privacy setting, e.g., sliding (or browsing) the notification pop-up setting and clicking some buttons to configure notification settings or selecting one of the user profiles (guest mode) and specifying them with permissions to access specific data. Given the need to act *swiftly*, configuring such privacy settings *in-situ* is difficult in front of an observer. Second, the guest mode find its limitation in cases where the device owner wants to show some photographs from a set of photographs. The user uses her own profile to show images which could result in some private contents getting leaked to the borrower or sharer. Such limitations call for additional efforts in providing adequate defense against socially-close adversaries [32], and that too, without compromising device *usability*.

This paper breaks away from the traditional approaches to configure *in-situ* privacy settings, and explores an alternative system, PrivacyShield, which is rooted in providing *subtle* just-in-time privacy while achieving usability. To this end, we first address the notion of *subtle* just-in-time privacy provisioning, which means configuring a privacy setting on a user desired information content by replacing the default approach of content hide with an indirect and swift approach. We do not argue that such an approach itself is always performed in a surreptitious way, although it might be the case in some situations. By leveraging the screen I/O device (screen digitizer) of smartphones, PrivacyShield first recognizes on-screen gestures that users perform even while a phone's display is turned off. If done well, it passes the recognized gestures as gesture queries to instantly change user profiles or the system's privacy modes according to the performed gestures. As the display is off, the phone owner can pro-actively hide, e.g., pictures, notifications or kill apps, before lending or screen sharing her device. The borrower or a nearby person may not *see* (be aware of) what information content the owner is hiding. Users may also use gestures to subtly configure dynamic rules, e.g., hiding message notifications from the WhatsApp app from a particular sender(s). To do so, the system facilitates individual policies associated with a gesture query. PrivacyShield also provides an Application Programming Interface (API) for smartphone apps to achieve selective hide functionality for their data and thus better balance their *privacy* and *usability*.

A key challenge in achieving subtle just-in-time privacy provisioning using PrivacyShield is to segment gesture inputs (or commands) from the screen digitizer in an accurate, fast, and energy-efficient way. Gesture commands in PrivacyShield consist of multiple inputs made with touch gestures, each representing an action or a query. For example, assume that a touch gesture consists of an action followed by a sequence of two characters as parameters **"ab"**. Segmenting such a sequence of touch-gestures with multiple inputs is very challenging because **1)** there is no visual feedback, as the display is turned off, and therefore **2)** multiple gestures can easily get overlapped, as users are not expected to draw them considering per-gesture-input timeout. **3)** Some gestures may consist of multiple strokes such as **"i"** and **"t"**, which, if not segmented properly, make existing gesture-recognition approaches provide the wrong output. **4)** Since PrivacyShield needs to always listen for touch inputs, there will be multiple cases of accidental CPU wake-ups (resulting in energy drain) from accidental touches.

To address the above challenges, we propose a novel stroke-based gesture-segmentation-and-recognition approach. First, we differentiate single-touch and multi-touch as a part of primary segmentation. Second, for segmenting the character gesture(s), we compute their stroke features. We separate a coordinate stream into a set of strokes and examine whether each stroke is for a single character or a part of the character (e.g., **"."** in **"i"**). If a given stroke is for a part of the character, it is merged with the subsequent stroke. We recognize the character by matching each segmented stroke (or two consequent strokes) with the pre-built templates. To provide just-in-time privacy protection, the approach accelerates the process of gesture segmentation. While a user is drawing the gesture on the screen, it instantaneously processes the segmented stroke(s), i.e., character-by-character.

We further optimize PrivacyShield to provide an energy-efficient solution against accidental touches which, in turn, causes the CPU to process touch inputs. To this end, we leverage low-power co-processors and a proximity sensor to determine when to process touch events and interrupt the CPU. PrivacyShield first detects a smartphone's position with the proximity sensor and then enables Android's touch events processing only when the smartphone is outside, e.g., device owner's pocket.

The design of PrivacyShield has been conducted through an online questionnaire and a focus group study. We designed a mock-up PrivacyShield system, and further crafted the design of PrivacyShield through a design workshop with fifteen participants who frequently came across device borrowing and screen sharing situations. We have implemented the PrivacyShield system and its segmentation-and-recognition algorithm on commercial smartphones. Experimental results show that the algorithm enables high recognition accuracy. More specifically, the segmentation accuracy is 95% and 89% for 1-character and 3-character gestures, respectively. The recognition accuracy (after segmentation) is 87% and 74% for 1-character and 2-character gestures, respectively. The recognition is performed within 200ms (at most), which enables just-in-time provisioning. We further performed extensive experiments to measure the usability of PrivacyShield by doing a 6-week system deployment study. We also implement the *wrist gesture method*, an alternate energy efficient and simple non-touch-screen-based subtle just-in-time privacy provisioning approach. We compare the alternate approach with PrivacyShield in relation to *in-situ* usability, flexibility, and accessibility to perform gesture inputs, and show why a simplistic solution falls short in providing a privacy configuration. Overall, our findings include the effectiveness of PrivacyShield in configuring subtle just-in-time privacy, the satisfaction of using PrivacyShield, the ability of the system to make people perform required gestures in a subtle way, and more importantly, participants' perceptions on using PrivacyShield.

The following are our main contributions:

- We conducted a design workshop with fifteen participants to establish design requirements for crafting the PrivacyShield system. §2.
- We conceptualize, propose, and design the PrivacyShield system for providing subtle *in-situ* just-in-time privacy option on mobile devices. §4.
- We develop a new algorithm for accurate, fast, and low-power gesture segmentation. §6.
- We implement the PrivacyShield system on commercial smartphones and build and customize example apps using the system API. §8 and §9.
- We conduct comprehensive experimental evaluations and report the results. §10 and §11.

2 DESIGN STUDY

To design a just-in-time privacy provisioning system, we conducted a design study consisting of two phases: an online survey and a design workshop. The goal of the first phase was to explore individual's perception of current privacy solutions. We set a motivating scenario and analyzed online feedback from 80 participants. In the second phase, we recruited 15 participants and performed a design workshop to extract design considerations. All user studies in the paper were conducted under IRB approval.

Table 1. Participants' demographics from the online survey.

Gender	Male (54), Female (26)
Age	16-20 (3), 21-25 (24), 26-30 (37), 31-35 (16)
Occupation	Bachelor's degree (35), Master's degree (23), Doctorate degree (22)
Ethnicity	Chinese (26), Korean (25), Indian (10), Mongolian (7), Others (12)

2.1 Motivating Scenario and Online Survey

Example Motivating Scenario. Sam recently had a vacation trip with his family members where he took several photographs with his smartphone. After coming back from the trip, he is now working on his project assignment using his laptop. Alongside, Sam is also using the desktop version of a smartphone based Instant Messaging (IM) app to talk to his friends. A moment after, Sam's mother asks him for his phone to see some photographs from their vacation trip. Sam is now uncomfortable that his personal photographs and the message-pop-ups from the IM app may reveal his secret information. *Another day*, Sam is attending a meeting with his client at a cafe. While showing some project details on his laptop he gets a message from his school friend with whom he was chatting while walking to the cafe. He realizes that the message was also seen by his client when the notification popped-up on the phone as it was lying on the table and was close enough to the client. In this case, Sam wants to hide further message notifications from his IM app but does not want to miss any messages from his boss.

In order to explore *in-situ* privacy control in above scenarios, we conducted an online user survey. We posted a recruitment flyer to online community sites of students and staff at our university. We recruited 80 participants from 11 countries (see Table 1). Participants were shown a scenario video to help understand the *motivating scenario*. The key questions in online survey included: Q1) "*Which privacy settings(s) do you use for preventing information leak in such situations?*", Q2) "*How often do you come across such situations?*", Q3) "*Are you satisfied with the effort required in choosing a privacy option from the app or device's privacy setting?*", and Q4) "*Are you satisfied with the available privacy options in the app or device's privacy setting?*"

As shown in Figure 1, 17 people (25%) did not consider the scenario as privacy threatening. 14 participants out of 17 belonged to the age group of 31-35. The remaining 3 participants were in the age group of 26-30. Thus, such privacy threatening situations are more common in the age group of 16-30. For the rest 63 people, 18 people (36%) used notification Hide/Silent for their app, Figure. 15 people (30%) mentioned that they categorized their photographs under public or private using a third-party app. 17 people (34%) used both settings to hide their information. 13 (21%) participants chose to "make excuses and keep the device under possession" to prevent information leak.

Regarding occurrences of privacy threatening situations in participants' daily life. 12 people (19%) out of 63 participants faced privacy situations "At least once a day". 17 participants (27%) experienced such situations "At least once a week", while the rest 34 participants (54%) do not come across such situation quite often. 22% came across such situations "At least once a month" and the rest 32% encountered such situations "At least once a year". The result shed light on the fact that such device borrowing or screen sharing scenarios occur quite frequently in some cases.

For the participants who used their respective privacy settings (50 out of 63 participants, i.e., excluding 13 participants who "make excuses and keep the device under possession"), we asked them to rate their satisfaction on 1) effort required in choosing a privacy option and 2) available privacy options in their app or device's privacy setting on a 6-point Likert scale − strongly dissatisfied − dissatisfied − slightly dissatisfied − slightly satisfied − satisfied − strongly satisfied. Figure 2 shows the distribution of satisfaction level across all 50 participants. Most responders were not satisfied with the *effort required in choosing privacy options.* As shown in Figure 2 (Left),

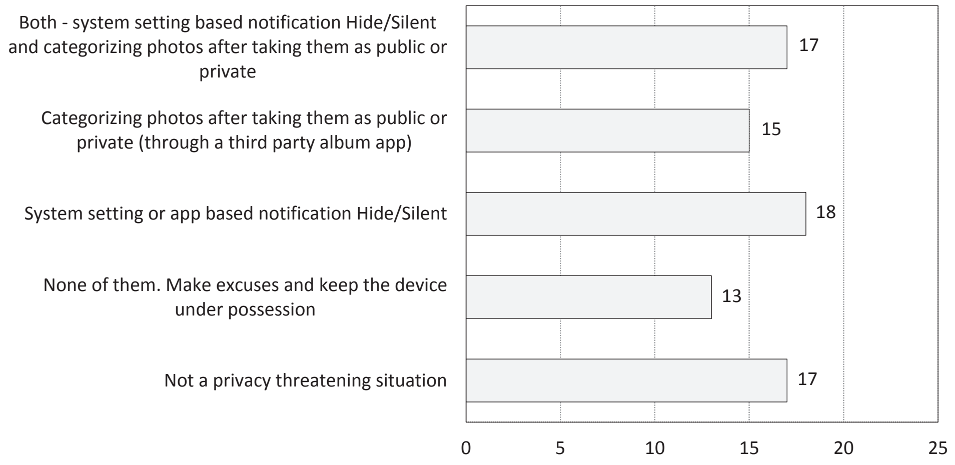


Fig. 1. Privacy Settings. X-axis represents no. of participants.

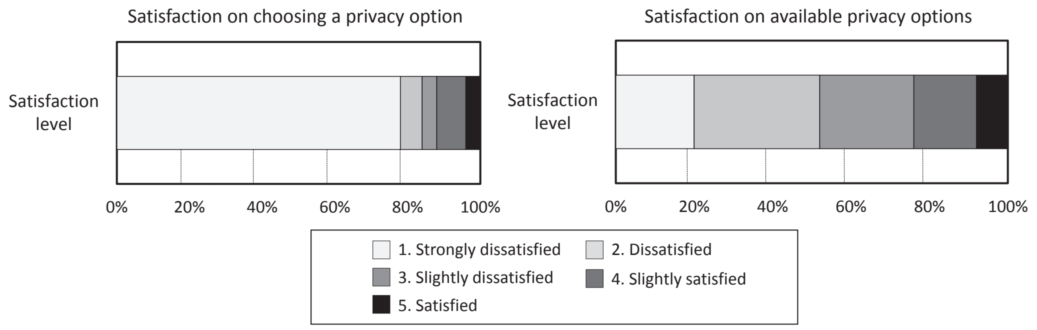


Fig. 2. Participants' response distribution of satisfaction level. (Left) Effort required in choosing privacy options. (Right) Available privacy options. Note that we did not receive any responses corresponding to "Strongly satisfied" option.

44 participants (88%) out of 50 chose from "Strongly dissatisfied" ~ "Slightly dissatisfied". 6 participants were (slightly) satisfied with the effort required to configure a privacy option.

Majority of the participants, 38 out 50 (76%), were not satisfied with the *available privacy option*s, Figure 2 (Right). 13 people (26%) were the ones who came across such privacy situations frequently in their daily life, i.e., from the option sets: "At least once a day" or "At least once a week". 9 people from the same sets chose that they were either "Satisfied" (1 person) or "Slightly satisfied" (8 people) with their privacy-provisioning options. From the remaining 28 people who belonged to the opinion sets "At least once a month" or "At least once a year", 25 of them (50% of the total participants) were not satisfied with the privacy options, while the other 3 participants were "Satisfied". An interesting fact which we found from the collected data was that the 6 participants who were (slightly) satisfied with "effort required in choosing privacy options" were also (slightly) satisfied with "available privacy options in their app or device's privacy setting".

2.2 System Design Workshop

To educe the design considerations of PrivacyShield, we conducted a focus group discussion with fifteen survey participants. 9 participants were from three groups (3 participants in each group): **a)** who did not use privacy provisioning settings, but frequently came across situations involving device borrowing or screen sharing (P_AX), **b)** who did not use privacy settings frequently (P_BX), and **c)** who already were (slightly) satisfied with the existing privacy options and with the effort required to configure privacy options (P_CX). Rest 6 participants were ones who frequently used existing privacy-provisioning settings under device sharing situations but were not satisfied with the effort required to configure privacy options and available privacy options (P_DX). We wanted to open the possibility of collecting a broader range of responses and implications of PrivacyShield for potential service to people with different usage rate of privacy-provisioning.

For the system design workshop, we built a mock-up PrivacyShield system. It is a working system providing basic swiping gestures that can hide message notifications and pictures from the gallery applications. The system provides gesture swiping functionality over a custom lock screen app. In the beginning, we provided the participants with a short tutorial about our notion of just-in-time privacy provisioning, demonstrated using the mock-up system, and let them freely use the mock-up system. We then conducted a three-hour semi-structured focus group discussion. The discussion was audio-recorded and transcribed, and the transcripts were analyzed individually by four researchers in our group. All participants finally reached a consensus on the high-level themes of the discussion [49]. The key questions were: "*How do you configure a privacy setting in front of an observer?*", "*What do you expect from the basic system design, which if fulfilled, will make you use the system around an observer?*", and "*What difficulty would you expect to face while performing just-in-time privacy provisioning around an observer?*" As an incentive, each participant was additionally rewarded with a $10 gift card.

Unavailability of easy *in-situ* selective hide option to information. Most participants commonly stated that *usability* is the primary limitation associated with hiding message notifications or photos with PrivacyShield. In fact, participants P_A1 P_A2, and P_A3 mentioned that they did not use existing privacy settings because there are some difficulties in information accessibility. P_A2, P_B3 reported that applying a message hide option typically blocks all the notifications. P_D1 mentioned: "*Even though I would be able to apply the message hide option, it deprives me from viewing message contents from the other important notifications.*" P_D3 stated: "*Upon activating notification hide, I would be checking each message every time I get one. The problem is very serious considering the frequent daily usage of mobile messaging.*" Most of the participants mentioned that in the current mobile messengers, users should choose either privacy or usability, and sacrifice the other.

Participants who used external photo album apps showed concerns with the current PrivacyShield system design. P_D1, P_D2, and P_B1 were worried about situations when they are browsing or screen sharing for showing a particular photo(s). P_A1 stated: "*We're a young couple, we take personal pics. We have many members in our family, they all love seeing the amazing pictures from my phone. [Well] personally speaking, and as well as on behalf of my family, nobody wants to be looking through my pictures and saying: ohh this is a nice scenery you got there behind you, and then, oops! I think I stepped in the wrong place.*"

Participants proposed to make PrivacyShield pass "*hints*" through gesture input. P_A3 questioned: "*May be it would be better to convey our thoughts to PrivacyShield regarding hiding notification (or photos) from (of) a particular person?*" Ideally, participants wanted those hints to convey the need of privacy configuration on a particular person's photos or messages. Some participants suggested to have multiple pre-defined profiles associated with gestures but many opposed to it as maintaining profiles sounded difficult to them. P_D3 mentioned: "*I don't know whether a couple of profiles is enough. But I feel I need more gesture rules as time goes by.*" P_B2 further mentioned: "*It will require some [thinking] effort to manage several profiles as I need to imagine who I will be meeting tomorrow.*"

Difficulty in subtly configuring just-in-time privacy options. While participants liked the idea of using gesture functionality, some were curious on using back-of-the device concept [24] for swiftly hiding information instead of using on-screen gestures. With further discussion among the participants, it became clear to them that many times they will not have the option to pick their phones to input gestures. P_D2 mentioned: "*I always keep my phone on the table, and may like to activate notification hide without picking the phone.*" To this, P_D4 suggested:

"*PrivacyShield should also be able to trigger any privacy configuration(s) without turning on the device screen. It would look like finger scribbling and could be deceptive to the person around.*"

Need for functional extension. Some participants enthusiastically suggested PrivacyShield to allow them to launch gesture functionality on top of any foreground app instead of launching it from the lock screen. P_D5 said: "*Sometimes I and my friend read articles on the web together. It would be annoying if I need to interrupt our reading and go through all the steps to launch the lock screen.*"

Applicability of PrivacyShield to users who are satisfied with current privacy-provisioning settings. Participants P_C1, P_C2, and P_C3 always used the current privacy-provisioning settings. They always had their message notification hide option activated. P_C4 mentioned: "*Since I frequently come across privacy threatening situations, I always tend to keep a particular privacy setting.*" Although, P_C1, P_C2 agreed that PrivacyShield would be a much flexible option and that they want to try the system. Their feedback strengthened our target user base. We design PrivacyShield to target people who desire usability while achieving privacy.

2.3 Summary

Online user survey and focus group discussion. The results indicate that people are concerned about privacy leaks from message notifications and accidental snooping through their personal photographs. They take various measures to prevent information leaks whenever they find themselves in device- or screen-sharing situations. Such measures, though, make them sacrifice either privacy or usability. Moreover, since such situations are sporadic in nature, therefore users are not able to use their current privacy-provisioning approaches *swiftly*, *selectively*, *flexibly*, and *readily*.

Design study limitation and scope of PrivacyShield. The online user survey and system design workshop was done with a small population size consisting of university students, leading to bias in their understanding of mobile device usability, privacy, and security issues. However, we believe that the results obtained still have merit as they represent views from different ethnicities and age groups. Thus far, the current version of PrivacyShield is oriented towards people with a fair amount of dexterity and mental agility. It also targets people who have a lot of experience with mobile devices.

3 EXAMPLE USE CASES AND THREAT MODEL

We envision two types of use cases based on the findings from the design workshop: subtle-privacy-aware apps and subtle system-wide configurations. We describe some example use cases below.

3.1 Subtle Privacy-aware Apps

Subtle privacy-aware apps work together with the PrivacyShield system for fine-grained privacy protection to selectively protect in-app data.

An app may receive a gesture query recognized by the PrivacyShield system and act accordingly. For example, In the above mentioned motivating scenario, Sam can simply put notification pop-ups from the messenger app to **"Hide"**. A messenger app may accept the gesture of (multi-touch) **"2-finger Right Swipe"** as hiding message pop-ups. Subsequently, a photo album app may accept a multi-touch gesture **"2-finger Left Swipe"** followed by first character (single-touch) **"S"** from Sam's name (mnemonic gesture), and thus, selectively hiding his name tagged photos. Therefore, along with hiding message notifications, Sam can concurrently hide his personal photographs. This fine-grained privacy control allows developers to enable users to achieve a balance between *privacy* and *usability* from their apps.

For hiding or prioritizing message notifications, existing platforms, like Android Lollipop [6] and Marshmallow [9], come with options such as prioritizing and Hide-All Messages [7]. However, those are limited to "all-or-nothing" functionality, i.e., stopping all the messages or blocking nothing, and thus affect the usability. Moreover, they are also neither subtle nor fast in nature, i.e., users have to explicitly configure such privacy settings after browsing through phone or app setting.

Similarly, for hiding photographs, there are no on-the-fly hide solutions. A device user has to infer a set of photos which she thinks are private. Thereupon, users choose a certain third-party app, e.g., [17][18], etc., and create a private folder. The inferred photos are then selected one-by-one and are moved to the private folder. Existing photo album apps can tag images automatically under a person's name using face recognition, location, or voice commands, e.g., iPhoto, Google Picasa, Google Photos, Smart Gallery [10] etc. Leveraging those tags these apps may receive subtle privacy configuration hints from the PrivacyShield system and selectively hide certain photos.

In Section 8, we show how to build a subtle-privacy-aware app using the PrivacyShield APIs.

3.2 Subtle System-wide Configurations

In this type of use case, users use gestures to switch a smartphone system into a specific mode or configure other system-wide settings. For example, using the xShare system or Cells [34][22], users may pre-define some sharing modes such as a guest mode where no private data or apps are accessible. Before giving a smartphone to a borrower, the phone owner may simply pull out the phone and draw a touch gesture (e.g., a **"2-finger up swipe"**) without turning on the phone's display. The phone will then silently switch into the guest mode so that the borrower cannot access any private information of the phone owner.

3.3 Threat Model

Our threat model targets social insiders who acts as curious and careless borrowers and screen-sharers and without a primary intent to extract data or make changes. These people can involuntarily access information on or from the phone (case of message-pop-ups, photo browsing) due to their behavioral tendency to snoop on others device [37]. For the device borrowing scenario, we assume that the borrower has her/his hand on the device whereas in case of screen sharing, the sharer is viewing the device while the owner is holding it. We assume a screen-sharer having approximately the same visual angle as the device owner and with device distance either equal or slightly larger. For the borrower, the person may or may not use the phone next to the owner. Shoulder surfing is not considered in our scenarios. Also, we don't consider password compromised cases.

4 SYSTEM DESIGN OVERVIEW

We design PrivacyShield considering the lessons from the design workshop and implementation feasibility. Figure 3 shows the system architecture of PrivacyShield. In the kernel space, the Touch Input Processor receives the raw touch inputs and sends them to the Gesture Recognizer in the user space (§9). PrivacyShield runs gesture recognition for multi-touch and single-touch gestures. The Gesture Recognizer uses the pre-trained Gesture Features database to recognize the gesture from the raw touch inputs. The recognized gesture is then forwarded to the Command Engine, which looks up the pre-configured Gesture Commands database to decide whether the gesture was registered as a gesture command to the app or not. For a registered gesture command, the Command Engine will execute the registered action of the gesture command, e.g., changing the system-wide settings or calling back to the PrivacyShield app. Otherwise, the gesture is dropped without performing any further action. When switching from PrivacyShield mode to the normal mode or to check its privacy configuration status, user authentication is required from the system settings. PrivacyShield also ensures that no performance degradation is encountered, e.g., when the phone is inside user's pocket. The system then goes into a suspend

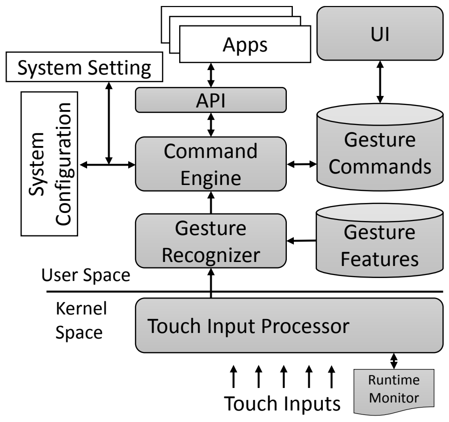


Fig. 3. System architecture of PrivacyShield. Gray-colored blocks are PrivacyShield's components.

mode, leveraging the proximity sensor on device with low-power co-processor (§7 and §9). Otherwise, the system would start processing accidental touch events, which would cause energy drain. We also choose simple 2-finger swipe gestures against single-finger swipes to make sure that there's no error in separating simple gesture commands from character gestures and to avoid accidental single-finger touch events. Using the PrivacyShield API, app developers may register gesture commands to a corresponding app. For each gesture command, a callback function is provided. Thus, when such a gesture command is recognized, the Command Engine may use the callback to notify the corresponding apps. In that case, the Command Engine will call all the callback functions provided by those apps. All the gesture commands and their callback functions are stored in the Gesture Commands database. PrivacyShield leverages the concept of name indexing [19] to extract name initials from apps using the PrivacyShield API. It then profiles and maps all the extracted characters from names with the gesture command(s) associated with an app. The PrivacyShield system also provides a UI for users to manage all the gesture command mapping(s) to an app(s). Users may browse, disable or enable existing gesture commands, and save the changes. Users may also choose different mappings to gesture commands.

Providing users with feedback and command activation flow. To provide feedback to users on whether a gesture command is correctly recognized or not, we take a vibration-based approach. In this paper, the gesture command is activated as **`gesture command(s)`** → **`system time-out`** → **`screen lock`**[1] → **`system activation`**. If the user decides to enter the pass-code or simply unlocks the device, the system activates the desired privacy setting. If the entered gesture command succeeds with a Horizontal swipe input, no vibration is triggered. A long vibration of one second is triggered when gesture command is entered with Vertical swipe input. Otherwise, two short vibrations of half a second each are used to indicate failure. Apart from notifying the user about gesture command success and failure; another important rationale behind giving vibration as a feedback is to notify the user about wrong gesture command usage, e.g., if the user accidentally (and correctly) enters a Vertical swipe gesture instead of an intended Horizontal swipe gesture. The device owner can always update their entry by simply switching off the screen and re-entering the command. We term such actions as **screen reset**.

Runtime Monitor. As users may accidentally touch their smartphones, e.g., while holding the phone inside their pockets, it is important not to process those accidental gestures. As PrivacyShield keeps the touch panel always on, such random touch events would trigger the Gesture Recognizer and the CPU, and so cause power

[1]Smart Lock based implicit authentication scheme [47][28][20] may co-exist with PrivacyShield. This will help in activating the system without launching the lock screen app.

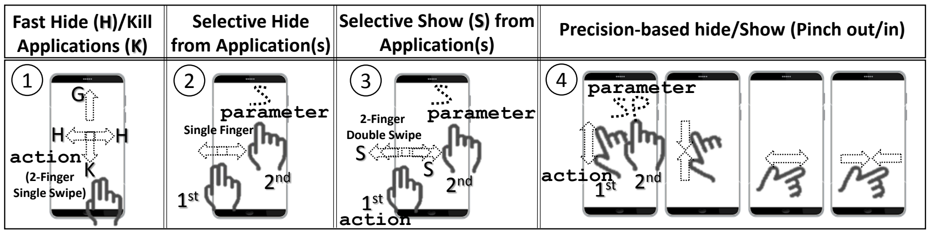

Fast Hide (H)/Kill Applications (K)
Selective Hide from Application(s)
Selective Show (S) from Application(s)
Precision-based hide/Show (Pinch out/in)
1
G
H
H
action
K
(2-Finger Single Swipe)
2
parameter
Single Finger
1st
2nd
3
2-Finger Double Swipe
parameter
S
S
1st
2nd
action
4
parameter
action
1st
2nd

Fig. 4. PrivacyShield "action" gesture commands. Multiple commands are supported at the same time. All gesture definitions (Hide/Show) are override-able. For selective and precision-based hide/show, "parameter" gestures are supported. 1st and 2nd denote the gesture input sequence. Users can select the system's pre-defined gesture mappings (in this paper) or can choose their own gesture mappings. "G" denotes the guest mode or multi-user [34].

consumption. We design to use a novel runtime monitor to leverage a low-power co-processor to prevent PrivacyShield from processing accidental touch events, thus preventing unnecessary waking up of the CPU.

Launching PrivacyShield in display-on case. PrivacyShield also works in cases where the smartphone owner is continuously interacting with the foreground app, e.g., when the device owner is using the phone for navigation purpose while her friend is sitting beside her. In such cases, we design the system to use the home button on the screen. A simple double tap on the home button will launch a transparent PrivacyShield overlay on the foreground app. The phone owner then can use gesture commands to enter the desired privacy configuration.

Privacy upon gesture input failure. On every screen-lock entry, if the gesture command input attempt results in correct gesture usage but a wrong character gesture, a Hide-All privacy configuration is enabled for the respective application case. Else, if there's a wrong gesture usage (irrespective of the character gesture input), the Guest Mode [34] is enabled. For accidental but correctly inputted gesture usage, the accidental gesture command will come into its effect. Device owner may revert gesture usage from the system settings.

5 GESTURE COMMAND FORMAT AND CHALLENGES IN RECOGNITION AND SEGMENTATION

We design PrivacyShield to support simple gesture commands consisting of 2-finger Right Swipe, Left Swipe, Down Swipe, and Up Swipe for Hide-All or Kill functionality. We also support 2-finger gestures followed by up to three lower-case characters for enabling selective (precision) hide/show functionality. The considerations for this design choice are: **1)** a few simple gestures are easy to remember and have high recognition accuracy; **2)** lower-case characters have fewer strokes and thus can be drawn faster than upper-case ones.

We define 2-finger gesture commands using the format of **"`action[parameter]`"**. The **"`action`"** part consists of a PrivacyShield gesture (consistently defined throughout the paper). The **"`[parameter]`"** part is optional and consists of up to three characters (selective (precision) Hide/Show case). Figure 4 shows the gesture commands.

PrivacyShield is in-built with 2-finger (*Single* and *Double*) **"Right Swipe"**, **"Left Swipe"**, **"Down Swipe"** and **"Up Swipe"**, along with (*Horizontal* and *Vertical*) **"Pinch in(out)"** gestures. The user may assign a character gesture(s) stating the intended target person(s) or app(s). In this case, PrivacyShield just matches the gesture command by using its action part and sends the whole command string to the corresponding app. For example, for the command **"2-finger Left Swipe sp"** in Figure 4, PrivacyShield only knows that the Photo app(s) is registered to the command **"2-finger Left Swipe sp"**. PrivacyShield just sends **"2-finger Left Swipe sp"** to the Photo app(s), which translates it into **"hide photos of Sam and Patrick"** (**"2-finger double Left Swipe"** for reverting to **show all**[2]), as per the user's intent along with the names starting with the same initials. This

[2]On a similar note, a Single 2-finger Right Swipe associated with "message-pop-up hide" will be overridden with a 2-finger double Right Swipe (configured to show pop-ups).

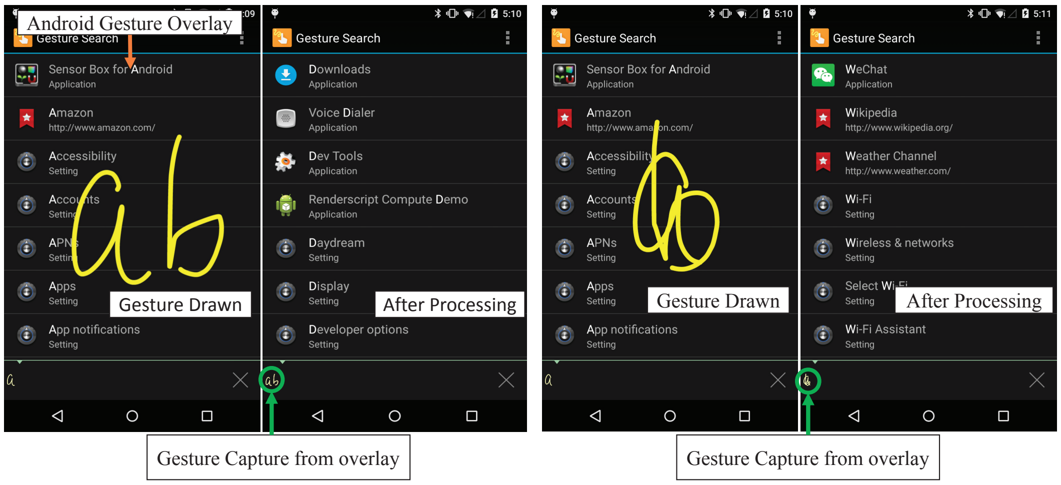


Fig. 5. Gesture Search relies on users for character segmentation and cannot handle (a) Gestures drawn next to each other (stroke interval 290ms) & (b) Gestures drawn on top of each other (interval 270ms). The Gesture Captures result in wrong recognition outputs.

functionality provides usability to the user as it results in the selective hiding of photos from **Sam** and **Patrick** but leaves out the rest from the hide operation.

To provide flexibility to scale the hide functionality for several message-pop-ups and photos, the system supports the use of a selective show option. For example, the command **"2-finger double Right Swipe sp"** will result in the system hiding all message-pop-ups except those from **Sam** and **Patrick**. PrivacyShield further provides additional functionalities, e.g., switching between system profiles (normal and guest modes (**G**)) or killing existing applications (**K**) through **"Up Swipe"** and **"Down Swipe"**, respectively.

PrivacyShield facilitates device owners to achieve precision on name selection. They only have to choose an extra command when they are about to enter name initials. Such a command is useful when the device owner finds herself in a situation where she intends to show photos from, say, **Alan**, but not, say, **Alice**. PrivacyShield configures privacy settings on the first and last letters of the name of a person. For example, for the gesture input **"Horizontal 2-finger Pinch out"** followed by **"ae"**. PrivacyShield gets to know that the **`parameter`** **"ae"** is for Alice and the **`action`** **"Horizontal 2-finger Pinch out"** is for accessing and showing content from a photo app (**"Horizontal 2-finger Pinch in"** for **hide**[3]). For the situation wherein the device owner has a friend other than Alice whose name has its first and last letters **"a"** and **"e"**, the precision-based name selection approach also provides users with an option to enter first three characters of a person's name. On a single character gesture entry with **"Pinch in(out)"**, the system will consider it as a wrong entry.

Challenges in recognizing multiple gesture. It is challenging to recognize these multi-character gesture commands, particularly in the context of PrivacyShield, as they are drawn in a fast and continuous way without any visual feedback to users, and without any careful gesture-input consideration. The maximum *inter-character* **[Set 2]** (in §10) and *intra-character* (**[Set 1]**[4] (in §6) and **[Set 2]**) time differences were reported as 368ms and

[3]Similarly, a Vertical 2-finger Pinch in associated with "message-pop-up hide" will be overridden with a 2-finger double Right Swipe (configured to show pop-ups).
[4]Each Set# in this paper denotes information collected from a separate set of participants with a specific study goal. No participant from these sets participated in the Design Study (§2) and the Usability Study (§11).

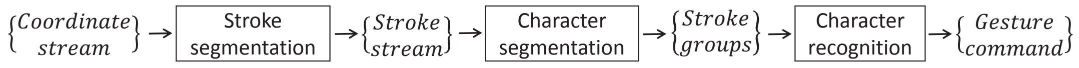


Fig. 6. Stroke-based gesture segmentation and recognition.

430ms, respectively. As shown in Figure 5, existing gesture-recognition systems, such as Gesture Search [33], rely on users to separate individual characters through visual feedback. For example, to input the two-character string **"ab"**, a user will first input **"a"**, wait for **"a"** to be recognized, and then continue to input **"b"** only after **"a"** has been correctly recognized. The user is able to do this because she can know when **"a"** is correctly recognized by seeing the recognition result of **"a"** on the screen. In PrivacyShield, as the user cannot decide when a character has already been recognized, the system may wrongly send both **"a"** and **"b"** to the recognizer, either side by side or overlapped. In both cases, Gesture Search would fail to recognize the characters.

Challenges in gesture segmentation and need for a new gesture segmentation method. One may think about some simple approaches to solve the problem of multiple gesture recognition, for example by using a long-enough wait time. In the above example of **"ab"**, after inputting **"a"**, the system could require that the user wait for 1-2 seconds before inputting **"b"**. As a result, it is likely (or hopeful) that **"a"** has been recognized during the 1-2 seconds time window. However, this approach would significantly increase the gesture-input time. As a quick input is critical, this approach would significantly affect the usability of PrivacyShield, and thus, is undesirable. Another possible solution may be by using Optical Character Recognition (OCR) techniques (e.g., Google Handwriting Input [25][26]), which are designed to scan a character stream and do the character-segmentation work. However, OCR may solve the side-by-side case, but not the overlapped one. However, in the past, there have been several researches on overlapped character segmentation. All researches [48][56][30] assume to have a user interface (UI) on top of which the users draw their gesture queries. In PrivacyShield, such concepts do not fit, as the system deals with a turned-off display.

The fact that some characters may consist of more than one stroke (e.g., **"t"**, **"f"**, **"i"**, etc.) makes this character-segmentation problem even more complicated. In this case, the character recognizer may receive a partial character (e.g., **"-"** or **"l"** in **"t"**) and thus fail to recognize it. Furthermore, grouping strokes together for a full character is non-trivial. For example, for a three-stroke sequence of **"l"**, **"."**, **"l"**, it is hard to decide whether they are **"i"**, **"l"** or **"l"**, **"i"**. That is, it is hard to decide whether the second stroke **"."** should be grouped with the first stroke or the third stroke, because, while drawing **"i"**, users may draw the top stroke (i.e., **"."**) first and then the bottom stroke (i.e., **"l"**), or they might do it in the opposite order.

The above challenges are unique in PrivacyShield. Next, we describe how we address these challenges.

6 SEGMENTING AND RECOGNIZING FAST-DRAWN GESTURES

To address the challenges in recognizing swiftly drawn gestures, we develop a novel stroke-based gesture-segmentation-and-recognition approach. The key idea is to divide the whole gesture-recognition procedure into two parts: segmentation and recognition. Before recognizing a character, we first try to determine which stroke or strokes represent a full character, to avoid recognizing a partial character.

There are three steps in our stroke-based approach, Figure 6. The raw touch inputs are a stream of coordinate pairs, each coordinate pair consisting of the {**x**,**y**} coordinates of a touch input. The first step lies in dividing the coordinate stream into strokes, called *stroke segmentation*. This step is simply done by analyzing the coordinates of the raw touch inputs. Each stroke contains a set of continuous touch inputs, and the output of the stroke-segmentation step is a stroke stream. The second step is *character segmentation*, which divides the stroke stream into stroke groups. Each stoke group may have one or two strokes that are considered as a full character. The

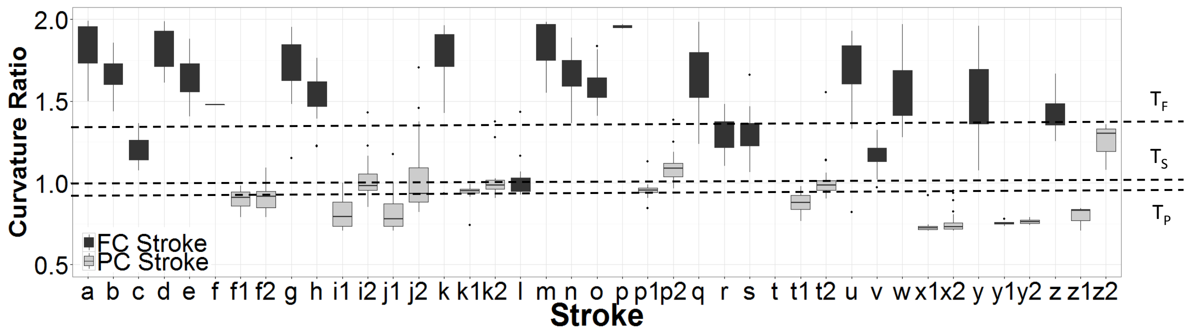


Fig. 7. Differentiation of full-character (FC) strokes from partial-character (PC) strokes using curvature ratio.

third step, *character recognition*, further recognizes the character of each stroke group and generates the final gesture command, which has up to three lower-case characters. The above steps work in pipeline fashion rather than one after another. For example, once the stroke segmentation step decides a stroke, it sends the stroke to the character segmentation step immediately and continues to search for the next stroke at the same time. As a result, the three steps may work simultaneously for faster segmentation and recognition.

6.1 Character Segmentation

The goal of character segmentation is to decide which stroke or strokes are a full character. To do this, we take a data-driven approach. We conducted a user study with 25 participants who were all young graduate students aged from 22 to 30 **[Set 1]**. All gestures were drawn in one attempt by all the participants. They each picked up the phone with one hand and drew gestures with the other. We asked them to draw all the 26 lower-case characters without any visual feedback on a Nexus 5 smartphone and collected the stroke data. Then we analyzed how those strokes were drawn. All the lower-case characters may have one or two strokes. From the collected stroke data, we confirmed that a single lower-case character has at most two strokes.

For a given stroke s, we use the feature of **"Curvature Ratio (CR)"** to decide whether it is a **"full-character (FC)"** stroke or a **"partial-character (PC)"** stroke. The **"CR"** of stroke **"s"** is defined as

$$\mathbf{CR(s)} = \frac{\mathbf{L(s)}}{\mathbf{H(s)+W(s)}},$$

where **L(s)**, **H(s)** and **W(s)** are the length, height, and width of stroke s, respectively.

We calculate the **CR**-values of all the strokes collected in our user study. Figure 7 shows the results in box-plot format, including the maximum, minimum, average, 25th percentile, 75th percentile, and outlier values. The black boxes are FC strokes and the gray boxes are PC strokes. For a two-stroke character, we name its short stroke as **stroke #1** and the long stroke as **stroke #2**. For example, for the character **"i"**, **"i1"** means the stroke **"."** (i.e., the top part of **"i"**) and **"i2"** means the stroke **"l"** (i.e., the bottom part of **"i"**). Some characters may be drawn by either one stroke or two strokes, e.g., **"k"**. For such strokes, we include both cases on the x-axis of Figure 7. For example, there are three strokes (**"k"**, **"k1"**, and **"k2"**) for the character **"k"**.

Figure 7 shows that the **FC** strokes and **PC** strokes are roughly divided into two groups by their **CR** values, although there are some overlapping cases. In general, the **CR** values of **FC** strokes are larger than those of **PC** strokes. To differentiate **FC** strokes from **PC** strokes, we define three **CR** thresholds. The first threshold, T_F is defined as the maximum **CR**-value of all the **PC** strokes. As a result, if the **CR**-value of a stroke is larger than T_F, the stroke is a **FC** stroke. The second threshold, T_P, is defined as the minimum **CR**-value of all the **FC** strokes.

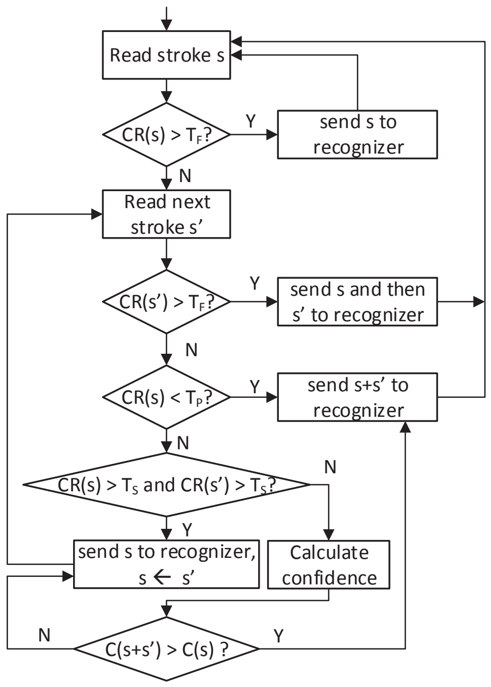


Fig. 8. Flowchart of our segmentation algorithm.

Consequently, if the **CR** value of a stroke is smaller than T_P, the stroke is a **PC** stroke. To further distinguish the strokes whose **CR**-values are between T_F and T_P, we use the third threshold, T_S, which is defined as the maximum **CR**-value of all the short strokes of the two-stroke characters. T_S may help when we cannot decide whether a stroke is a **FC** stroke or a **PC** stroke. In such a case, we combine the next stroke to see whether the two strokes can be combined into a two-stroke character or not. If the **CR**-values of both the strokes are larger than T_S, they cannot be combined as a two-stroke character because a two-stroke character must have a short stroke. Consequently, we infer that the first stroke is a **FC** stroke.

We design our character-segmentation algorithm to be driven by these three thresholds. Figure 8 shows the main flow of our algorithm. We first read a stroke **s**. If **CR(s)** is larger than T_F, we infer that **s** is a **FC** stroke. Then we send **s** to the character recognizer (i.e., move to the character-recognition step in Figure 8) and go back to read another stroke. Otherwise, we read the next stroke **s′**. If **CR(s′)** is larger than T_F, we infer that **s′** is a **FC** stroke, and thus, **s** is also a **FC** stroke. Then we send **s** and then **s′** (separately, as they are two characters) to the recognizer and go back to read another stroke. Otherwise, if **CR(s)** is less than T_P, we infer that **s** is a **PC** stroke and must be combined with **s′**. Then we send the combination of **s** and **s′** to the recognizer and go back to read another stroke. Otherwise, we further decide whether both **CR(s)** and **CR(s′)** are larger than T_S. If yes, we infer that **s** cannot be combined with **s′**, and thus, **s** is a **FC** stroke. Then we send **s** to the character recognizer, keep **s′** as the new **s**, and go back to the **"Read next stroke s′"** step. Otherwise, we run into the situation where we cannot make a decision using the threshold. In this case, we leverage our character-recognition algorithm (to be described in the next sub-section) to calculate how likely i **s** should be combined with **s′**. If **s** should be combined with **s′**, we send the combination of **s** and **s′** to the recognizer and go back to read another stroke. Otherwise, we decide that **s** is a **FC** stroke. Thus, we send **s** to the character recognizer, keep **s′** as the new **s**, and go back to the **"Read next stroke s′"** step.

6.2 Character Recognition

We leverage existing Nearest Neighbor (NN) and Dynamic Time Warping (DTW) [27] techniques to build our character-recognition approach. DTW is designed to find the optimal alignment between two dissimilar time-series-based sequences, by calculating the dissimilarity or distance between the two sequences. In general, for two sequences **S1** and **S2**, their DTW distance is computed using a function of the two sequences:

$$\mathbf{Distance} = f(\mathbf{S1},\mathbf{S2})$$

As strokes in PrivacyShield are all time-series-based sequences of coordinates, DTW is an effective way to calculate the distance/dissimilarity between strokes.

Our character-recognition approach has an offline part and an online part. In the offline part, for each character, we train a template using our collected user data and an online database [1]. In the online part, for a given stroke group, we calculate its distance to every character template and recognize it as the character of the template which has the shortest distance to the stroke group. In Figure 8, the character-recognition step contains only the online part. We use DTW to calculate the distance in both the offline training and the online recognition. Moreover, we associate a confidence value (between 0 and 1) corresponding to the recognition results. A confidence value is the ratio of the number of similar feature samples (coordinate values in our case) from a test stroke group to the total number of feature samples in a class (character template). The confidence values are calculated across all the classes. Therefore, the output of the DTW-based recognition algorithm is the recognized character and the highest confidence value that shows how likely the recognition is correct. The recognition algorithm is also used in the character-segmentation step when we cannot make a segmentation decision using the **CR** thresholds, as we mentioned in §6.1. In that case, we need to decide whether the first stroke is a **FC** stroke and should be combined with the second stroke. To do this, we send the first stroke and the combination of the two strokes to the recognition algorithm and get the returned confidence values. Based on the confidence values, we can make the decision as shown in Figure 8.

One may wonder: why not just use the recognition algorithm without doing the segmentation first, i.e., for a given gesture command, we may wait for all the user inputs and try all the stroke combinations using the DTW-based recognition algorithm. We call this the DTW-only approach. The key reason why we do segmentation is for speed and power efficiency. DTW is *time consuming* and *energy draining*. Without doing segmentation, we have to run DTW on many combinations and thus introduce significant *latency* and *energy cost*. Our segmentation algorithm makes most decisions based on the **CR** thresholds, which is very fast. Only in very few cases we have to call the recognition algorithm for help, but we only need to test two cases. As we do the segmentation first, we only need to run DTW once for each stroke group. In §10, we show that our approach is much faster and more accurate than the DTW-only approach.

Choice of DTW based recognition method. Complex gestures are stroke and shape independent. A simpler template-based approach works well for easy and user-dependent gestures. DTW outperforms other simpler and state-of-the-art techniques' recognition algorithms [53] in the case of user-independent and multi-stroke gestures [54]. Nevertheless, existing gesture algorithms, like Protractor [33], which runs on Android's core framework [2], can also be used with our gesture segmentation algorithm. We compare the performance of our gesture segmentation algorithm with both DTW and Protractor in §10. In the future, our gesture segmentation algorithm would find usability in any advanced recognition algorithm.

Listing 1. PrivacyShield runtime monitor for proximity sensor inside co-processor's sensor device interface. For brevity, we have not shown PrivacyShield device work suspension, register write prevent operation and page fault exception under virtual device definition for touch panel.

```
 #ifdef PRIVACY_SHIELD
 .....
/*Code block for sensor write detect*/
 CWMCU_i2c_write(mcu_data, CWSTM32_ENABLE_REG+i, &data,1);
 if ((mcu_data->input != NULL) && (sensors_id == Proximity) && (enabled == 1))
/*Code block for reading sensor write*/
 int PrivacyShield runtime_monitor(void)
 {u8 data[10]={0};
 int ret;
 CWMCU_i2c_read(mcu_data,CWSTM32_READ_Proximity,data, 2);
 I("[PS] check PrivacyShield:data0=%d data1=%d\n", data[0], data[1]);
 ret = data[0];
 return ret;}
 .....
 #endif
```

7 HANDLING ACCIDENTAL TOUCHES

With the present Android's touch-processing mechanism, PrivacyShield will interrupt the main CPU every time there's a touch event. To avoid false CPU interrupts caused by accidental touches, e.g., when the phone is inside the user's pocket, PrivacyShield uses two policies. For devices like Nexus 5, we use *user-triggered initialization* as it has no low-power co-processor for its touch panel I/O. With user-triggered initialization, the user controls the switching on and off of the screen digitizer, thus negating the permanent usage of PrivacyShield system. We implement user-triggered initialization in the power button driver layer for the touch screen. Every time the user wants to initiate (or deactivate) the PrivacyShield system, she needs to quickly double-press the power button. This functionality triggers the display processor but with no LCD back-light. On Nexus 5, the system may also result in performance degradation from accidental touches when the user is attending a phone call. In such situations, the main CPU is already awake. Thus, the system will process those accidental touches. To overcome this problem, we simply utilize **voc_start()** and **voc_end()** functions under the device microphone's driver to infer a situation of phone call.

On devices like HTC M8 with a low-power co-processor, we implement a runtime monitor which can help PrivacyShield to run continuously by leveraging the low-power co-processor (§9). It monitors the sensor device's write operations (at present for proximity sensor only), leveraging the sensor-MCU interface (Listing 1). The monitor checks the sensor device's state (**proximity_flag()**) and then prevents any register write operation from the touch panel's driver (leveraging ARM architecture memory mapping via, e.g., **I^{2}C**) and puts PrivacyShield's work function to sleep.

8 PRIVACYSHIELD API AND EXAMPLE PRIVACYSHIELD APPS

PrivacyShield API. As shown in listing 2, PrivacyShield API has two functions and one callback. The **RegisterGestureListener()** function is for apps to register gesture commands to our PrivacyShield system. The **gestureCommand** parameter is a string of up to three lower-case characters. The **callback** parameter is a callback

Listing 2. API of PrivacyShield.

```
/*gestures to enable*/
 enable_gesture_actions(2-fingerswipes, ...)
  *gesture_actions = {2fingersinglerightswipe, 2fingersingleleftswipe, 2fingersingledownswipe,
       2fingerdoublerightswipe, ...}

/*Register Gestures*/
 Bool RegisterGestureListener(String gestureCommand, Bool hasParameter, GestureListener callback);

/*Unregister Gestures*/
 Bool UnregisterGestureListener(String gestureCommand);

/*PrivacyShield App mapping*/
 Interface GestureListener {
   public Bool onCommandReceived (String command); }
```

function provided to the app for PrivacyShield to call the app back. The **`hasParameter`** parameter indicates whether the app-registered command has a parameter or not. If its value is **`true`**, PrivacyShield will use the value of the **`gestureCommand`** parameter (from **`string matching`**) for prefix matching in deciding whether a gesture command should be sent to the app or not. For example, if an app-registered **"`action`"** command and a **"`parameter`"** are allowed, PrivacyShield will send both the **"`action`"** gesture and the **"`parameter`"** gesture **"b"** to the app. Otherwise, PrivacyShield will do **"`action`"** gesture matching, and thus, only the **"`action`"** gesture but not the gesture **"b"** is sent to the application. If needed, application-set may call the **`UnregisterGestureListener()`** function to cancel a registered gesture command. It has only one parameter to tell which gesture command is to be canceled. The **`GestureListener()`** callback interface has only one callback function, called **`onCommandReceived()`**, defined, which is used by PrivacyShield to send a recognized gesture command to the corresponding app.

Using the above-mentioned PrivacyShield API, we built one new example PrivacyShield app called Talking Album and customized two existing Instant Messenger (IM) apps to run with PrivacyShield.

PrivacyShield-aware IM. We didn't build a new IM app, as nobody would use our new app. Instead, to make it real and useful to existing users, we customized two existing IM apps for PrivacyShield. Android provides a service called **`AccessibilityService`** that can be used for an app to access the internal data of other apps if the permission is granted. We find that two popular IM apps on Android, *Kakao Talk* (com.kakao.talk) [3] and *WeChat* (com.tencent.com) [4], support **`AccessibilityService`** for accessing their ongoing messages. Therefore, our system leverages **`AccessibilityService`** to enable dynamic control of pop-up messages in *Kakao* and *WeChat*. Specifically, our system keeps monitoring the incoming messages in *Kakao* and *WeChat* and can know when those messages pop up. Using PrivacyShield's APIs, we register the gesture commands. E.g., one gesture command is for blocking incoming messages from a certain sender. For example, **"2-finger Right Swipe s"** carries the intention **"Do not show the content of the messages sent from Sam"**. Then, if our system receives the command, it will prevent showing both the *content* and *name* from the messages received from **Sam**. To do this, our system shows our own pop-up window to hide the pop-up window of *Kakao* and *WeChat*. In our pop-up window, we do not show the content or name from a message if the message is received from **Sam**.

Talking Album. This app is built on top of the open source camera app called *OpenCamera* [15]. The existing functions of OpenCamera are just for taking photos and saving them as albums. We extend it in the following aspects. First, after taking a photo, we allow the user to use speech to tag it, e.g., speaking out the name of the person in the photo to tag the photo. We leverage *Bing Speech Recognition* and *Face* APIs (*Project Oxford*) [12] to **1)** translate speech into text and use the recognized text to tag the photo and **2)** use facial recognition on the tagged photo to auto-tag the rest similar person's photos, except those which are explicitly tagged to remain under a given name. Thus, reducing the burden of tagging same person's photo every time that person's photo is taken. Presently, we only support English names. Second, we organize all the photos according to their tags. For example, we may group all the photos of the same person into a photo album. Finally, we use our PrivacyShield API to register some gesture commands to manage the photo albums in a quick way. Similar to the PrivacyShield-aware IM app, we may use the gesture command **"2-finger Left Swipe s"** to hide all the photos tagged by the name **"Sam"**. After receiving the **"2-finger Left Swipe s"** command, when a user launches the app, none of the photos of **Sam** will be shown. The app accesses the Android's "Image Picker" interface which returns an internal URI (Uniform Resource Identifier). It then leverages the **`ContentResolver`** object to read a photo and then perform the user desired photo hide/show through a file permission change. With the PrivacyShield API, app developers attest only the gesture functionality to their app. For the Talking Album app, the gesture attestation part requires only 58 LoC.

9 IMPLEMENTATION

We have implemented the PrivacyShield system on two smartphones: a LG Nexus 5[5] [5] and a HTC One M8 [8]. Both phones have a quad-core 2.3 GHz Krait 400 CPU. The M8 phone also has a low-power co-processor of 72 MHz. The Nexus 5 phone runs Android 4.4.4 KitKat and the M8 phone runs the HTC-customized Android 5.0.2 Lollipop with the HTC Sense UI. Driver implementation on Nexus 5/(5X) and HTC M8 took 640 and 547 LoC, respectively. PrivacyShield user-space took 2,190 LoC.

Getting Touch Inputs with Display off. On the Nexus 5/5X phone, we figured out that the LCD display panel and the touch panel (i.e., the screen digitizer) are coupled together in hardware and they can only be powered on or off at the same time. It is not possible to turn off the LCD panel but keep the touch panel on. To implement PrivacyShield, we emulate the display-off-but-touch-on state by modifying the display driver and the touch-panel driver in the kernel space.[6] In the display driver, when the display is turned off (in case of a screen reset), we only turn off the backlight but not the LCD panel itself. This is done by modifying the **`mdss_dsi_panel_off`** function. When the backlight is totally off, the display shows nothing and looks like it is turned off. We call this the *emulated-off* state. In the touch panel driver, when the display is in the emulated-off state (determined via the **`lcd_notify`** notification), we do not turn off the touch panel and continue to receive new touch inputs. To achieve this, we implemented a PrivacyShield virtual device. We also create a dedicated **sysfs** [39] device file entry for the Gesture Recognizer[7] to read the raw touch inputs with time-stamp for single-touch, and time-stamp and **`_TRACKING_ID`** for multi-touch. When the display is turned on, we turn on the backlight in the display driver and disable our own **sysfs** device entry in the touch-panel driver. Consequently, new touch inputs will be dispatched to the Android framework via the original **sysfs** device entry, and the user can continue to normally use the phone.

On the M8 phone, the display panel and the touch panel are separate in hardware. We may power-off the display panel but keep the touch panel powered always on to receive raw touch inputs (instead of the existing sensor-based wake-up mechanism [8], to avoid any extra effort to activate the screen). Thus, we do not need to modify the display driver. Similar to the Nexus 5/5X phone case, we modify the touch panel driver to create a

[5]We also implemented PrivacyShield on a Nexus 5X smartphone. The hardware design (digitizer and display panel coupling) was similar to Nexus 5. Thus, we only present the details from the Nexus 5 implementation.

[6]We emulate screen I/O as a power-button. In this way, each touch input resembles a button press.

[7]The system uses the Input Dispatcher daemon in the Android OS and leverages the binder mechanism to pass all touch inputs from the Input Dispatcher to the Gesture Recognizer.

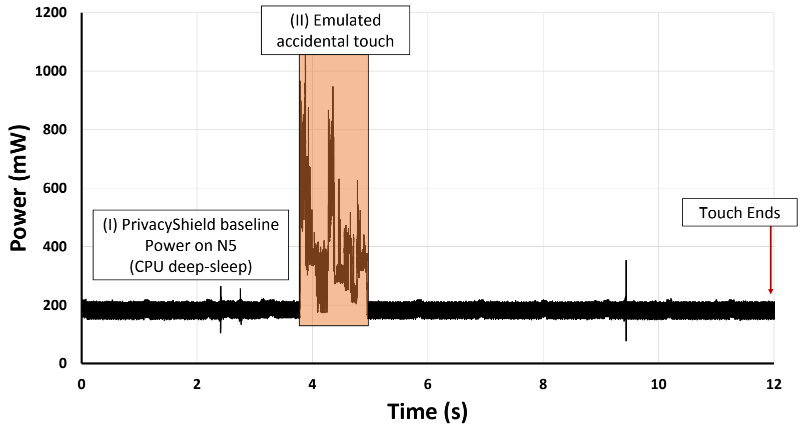


Fig. 9. (I) Baseline power of N5 after display processor is turned on. (II) Wake lock time-out makes sure that even an accidental touch, if resides on the screen without any movement, will not cause extra energy drain.

dedicated **sysfs** device entry for the gesture recognizer to read the raw touch inputs when the display is turned off. Different from the Nexus 5/5X phone case, we leverage the low-power co-processor of the M8 phone to listen to touch inputs. This is done by modifying the configuration of the low-power co-processor (via the `CwMcuSensor` function). As a result, when there's no touch input, the main CPU can sleep to save power. When a touch input comes, the co-processor wakes up the main CPU to process the input.

Further Power Optimization. To further reduce power consumption, we take the following approach. We allow the main CPU to frequently sleep to save power when there is duplicate touch inputs. After passing the check from the runtime monitor, i.e., `proximity_flag(false)` → `PrivacyShield active`, PrivacyShield enables the system to wake up the main CPU when it hears a touch input. After being woken up by a touch input, the following touch inputs are sent to the Gesture Recognizer. We configure the touch panel not to generate duplicated touch inputs with same touch coordinates. This approach is different from the current *Android touch-processing mechanism*, which continuously processes the touch input, and thus holds a wake-lock on the CPU. Therefore, the system helps in reducing the CPU's need to be waken frequently when a user holds a phone still; i.e., when there is no movement on the touch panel, the main CPU will not continuously receive the duplicated touch inputs, and thus may go to sleep to save power. This configuration is done by adding a reset function when repeated touch coordinates are encountered. The function resets the driver's touch listener function by using `gpio_reset` to its initial touch input listening state and simultaneously releases CPU wake-lock. Figure 9 shows an example of emulated accidental touch input when running PrivacyShield on Nexus 5. The system interrupts the CPU when it detects a touch but immediately puts CPU to sleep, the moment it finds out no movement from the touch (case of duplicate coordinates).

10 EVALUATION

We evaluate our segmentation algorithm and PrivacyShield system in terms of segmentation and recognition accuracy, processing latency, and power consumption. For measuring the performance of the segmentation algorithm, we performed experiments on a MacBook Air running a 1.6GHz dual-core Intel Core i5 with 4 GB of RAM. To measure on-device algorithm-processing time and processing latency, we used a Nexus 5 smartphone running Android 4.4.4 KitKat. The evaluation results show that our approach is more accurate than the DTW-only and Protractor-only gesture recognition approaches in terms of both segmentation and recognition outputs. In terms of processing latency, our segmentation approach is faster than the DTW-only approach. Moreover, our power-optimization techniques are effective in reducing the system's power consumption.

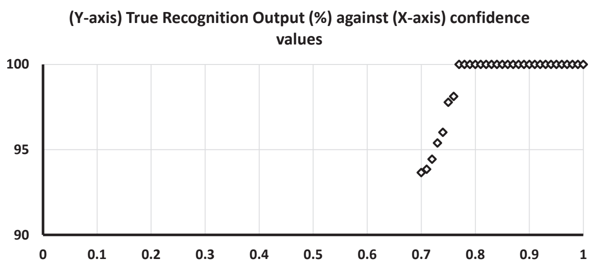
10.1 Data Collection and Replay

We take a trace-driven approach for the evaluations. To do this, we collected data in two stages. In the first stage, we collected data for measuring the segmentation algorithm's performance in terms of segmentation and recognition accuracy against recognition-only algorithms. We collected data from 24 participants, who were all young graduate students aged 21 to 34, 13 of whom were females **[Set 2]**. We introduced the PrivacyShield system to them and asked them to draw single-character (1-c) gestures of all the 26 lower-case characters on the screen digitizer. As it is hard to collect large combinations (676 combinations of 2-character (2-c) and 17,576 combinations of 3-character (3-c) gestures) of gestures from each participant, we leveraged the 1-c data to generate a synthetic data set. To generate 2-c gesture combinations, we used 1-c gesture sets from two participants each (to avoid bias due to character similarity), e.g., {26 (P(1)) × 26 (P(2))}, {26 (P(3)) × 26 (P(4))}, ... {26 (P(n-1)) × 26 (P(n))}, where n = 24 participants, and generated 676 × 12, i.e., 8,112 2-c gesture combinations. Similarly, we utilized three participants' 1-c gesture sets to generate a total of 1,40,608 (17,576 × 8) 3-c gesture combinations.

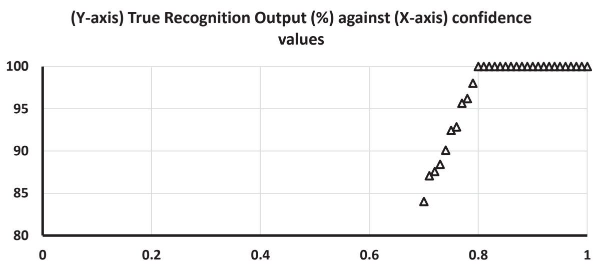
To understand the performance of the segmentation algorithm on the varying shapes and sizes of continuously drawn gestures, our second stage of data collection focused on logging actually drawn 2-c and 3-c gestures. As mentioned before, logging a large number of gesture combinations from each participant is difficult. Therefore, we asked participants to draw limited but important cases of 2-c and 3-c gesture combinations. The cases included scenarios where all gesture combinations were drawn using two stroke (2-s) gestures or were drawn in combination with single-stroke (1-s) gesture(s). Moreover, in the first stage of our data collection, we saw participants using two strokes in drawing gestures for **"d"**, **"f"**, **"i"**, **"j"**, **"k"**, **"p"**, **"t"**, **"y"**, **"z"**, and **"x"**. Therefore, in the second phase, participants drew these gestures as 2-s gesture(s) in combination with single-stroke 1-s gesture(s). Participants drew: **(1)** 3-strokes (3-s) : 2-c (10 × 16 × 24; 3,840 gesture combinations), and 4-strokes (4-s) : 2-c (10 × 10 × 24; 2,400 gesture combinations); **(2)** 4-s : 3-c (16 × 16 × 10 × 24; 60,000 gesture combinations), 5-strokes (5-s) : 3-c (10 × 10 × 16 × 24; 38,400 gesture combinations), and 6-strokes (6-s) : 3-c (10 × 10 × 10 × 24; 24,000 gesture combinations). All participants volunteered for a 2-week data recording period. Each participant recorded data every day for at least 1 hour. Participants were shown a series of random 2-c and 3-c gesture combinations on a laptop screen. Thereafter, they entered character combinations as quickly as possible. In all the experiments, the participants were given single attempts to draw the gesture(s). Moreover, all participants drew gestures while sitting. All participants chose to draw gestures with their index finger while holding the device in the other hand. Participants were offered a $30 gift card. We re-played the recorded data on our PC version of the algorithms to get our evaluation results. By separating performance evaluation from data collection, we avoided the impact of performance measurement, and thus got clean user-input data with accurate timing information and ground truth.

10.2 Avoiding False Positives in Recognition

Hand-drawn gestures are susceptible to device orientation caused by hand movements; therefore we made the gesture recognition resistant to false positives. To do so, we determined an appropriate threshold for the confidence value (in recognizing a gesture set), above which the possibility of any false positive becomes negligible. To find a confidence value threshold, we used the synthetic 2-c and 3-c gesture combinations obtained from the the first stage of our data collection. As the gesture sets were replayed, gesture recognition was performed and each detection of the correct gesture set was flagged. To investigate the effect of varying the confidence



(a) The 2-c recognition true recognition output rate versus confidence values.



(b) The 3-c recognition true recognition output rate versus confidence values.

Fig. 10. The recognition precision rate against confidence value threshold (related to DTW recognizer).

Table 2. Segmentation accuracy and recognition accuracy (with confidence value of 0.8 (DTW) and 0.75 (Protractor)) of one-character (1-c), two-character (2-c), and three-character (3-c) gestures.

	Segmentation			**Recognition**		
	1-c	**2-c**	**3-c**	**1-c**	**2-c**	**3-c**
Our approach with DTW	95%	94%	89%	87%	74%	63%
DTW-only	91%	75%	69%	83%	61%	43%
Our approach with Protractor	-	-	-	84%	68%	56%
Protractor-only	85%	76%	62%	80%	56%	35%

value threshold of the recognizer, the program iterated through the complete data-set by incrementing the threshold distance for each iteration. Figure 10 shows the relationship between the true recognition output (precision) and the threshold level for the confidence value measured from the DTW recognizer. Based on the populated data, the true recognition output converges to 100% at a confidence value of 0.77 and 0.80 for 2-c and 3-c recognition, respectively. For the Protractor recognizer, we followed a similar approach. The retrieved threshold confidence values in this case were 0.72 and 0.75 for 2-c and 3-c gestures, respectively. In both cases, we chose a higher confidence value of 0.8 (DTW) and 0.75 (Protractor) to consider the output of recognition as valid (including 1-c gestures). Note that a high confidence value would also inflict true negatives through the system. However, this approach provides the owner the ability to achieve an accurate usability configuration, but not at the cost of sacrificing privacy. Also, a higher confidence value threshold provides a tight criterion for the users of PrivacyShield to draw gestures as nearly as to the elicited templates. In terms of the system's perspective, any gesture recognition output below this value results in the system enabling Hide-All, meaning that the gesture was not recognized.

10.3 Segmentation and Recognition Accuracy

We evaluated the performance of our segmentation algorithm in two steps. In the first step, we used the synthetic data set and compared our approach with the DTW-only and Protractor-only [51] approaches, which processes various combinations of all the strokes of a gesture command to decide the best recognition results. For the sake of fairness, in the DTW-only and Protractor-only approaches, we also assume that a character can have up to 2 strokes, and thus use a 2-stroke window to reduce the number of stroke combinations. In the second step, we measure the segmentation and recognition accuracy of the segmentation algorithm with the DTW-only approach in relation to actually drawn 2-c and 3-c gestures.

Table 3. Segmentation and recognition accuracy of 3-stroke and 2-character (3-s:2-c), 4-stroke and 2-character (4-s:2-c), 4-stroke and 3-character (4-s:3-c), 5-stroke and 3-character (5-s:3-c), and 6-stroke and 3-character (6-s:3-c) gesture combinations.

	Segmentation					**Recognition**				
	3-s:2-c	**4-s:2-c**	**4-s:3-c**	**5-s:3-c**	**6-s:3-c**	**3-s:2-c**	**4-s:2-c**	**4-s:3-c**	**5-s:3-c**	**6-s:3-c**
Our approach with DTW	91%	90%	92%	89%	86%	78%	71%	68%	63%	55%
DTW-only	78%	73%	76%	71%	63%	64%	55%	50%	42%	37%

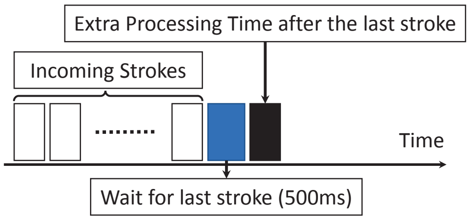

Extra Processing Time after the last stroke
Incoming Strokes
Time
Wait for last stroke (500ms)

Fig. 11. Processing latency is the extra processing time after the last stroke arrives.

Segmentation and Recognition Accuracy on synthetic data. The results in Table 2 show that our approach greatly outperforms the DTW-only and Protractor-only ones in terms of both segmentation accuracy and the final recognition accuracy. The accuracy of our approach is practical for one-character and two-character gestures, which would be widely used in real-life situations. For the segmentation, our approach shows 95% and 94% accuracy for one- and two-character gestures, respectively. In comparison, DTW-only shows 91% and 75% segmentation accuracy, respectively; and Protractor-only shows 85% and 76% segmentation accuracy, respectively. For the final recognition accuracy, our approach shows 87% and 74% for one-character and two-character gestures, respectively; whereas DTW-only shows 83% and 61%, respectively. On the other hand, Protractor-only shows 80% and 56% recognition accuracy, respectively. Our approach also outperforms DTW-only and Protractor-only for three-character gestures, with 89% segmentation and 63% recognition accuracy. The gap between segmentation and recognition accuracy becomes larger since the error multiplies as the number of characters increases. While three-character gestures would not be widely acceptable by users due to their long operation times in our scenario or their non-frequent usability requirement, they could be applied in different application scenarios, e.g., in case of name-encoding collisions, controlling a music player's play list without turning on the screen, etc. We leave the improvement of the accuracy for multiple-character gestures as future work.

Segmentation and Recognition Accuracy on actually drawn 2-c and 3-c gestures. Table 3 shows the segmentation and recognition accuracy after replaying different combinations of strokes for various 2-c and 3-c character gestures. Our approach maintains its performance with respect to the segmentation accuracy. For 3-s : 2-c and 4-s : 2-c gesture combinations, the algorithm results in 91% and 90% segmentation accuracy, respectively; and 78% and 71% in recognition accuracy, respectively. In cases of 4-s : 3-c, 5-s : 3-c, and 6-s : 3-c gesture combinations, the recognition performance from the segmentation algorithm shows a higher accuracy of 68%, 63% and 55%, respectively; whereas, DTW-only shows 50%, 42% and 37%, respectively. In segmentation performance, our approach shows 92%, 89% and 86% segmentation accuracy, respectively. In comparison, the DTW-only approach results in 76%, 71% and 63% segmentation accuracy, respectively.

10.4 Processing Latency

Figure 11 shows how the algorithm-processing time may impact the system latency. As a set of strokes arrives, our algorithms start to process them in pipeline. To decide the last stroke, our PrivacyShield system takes a

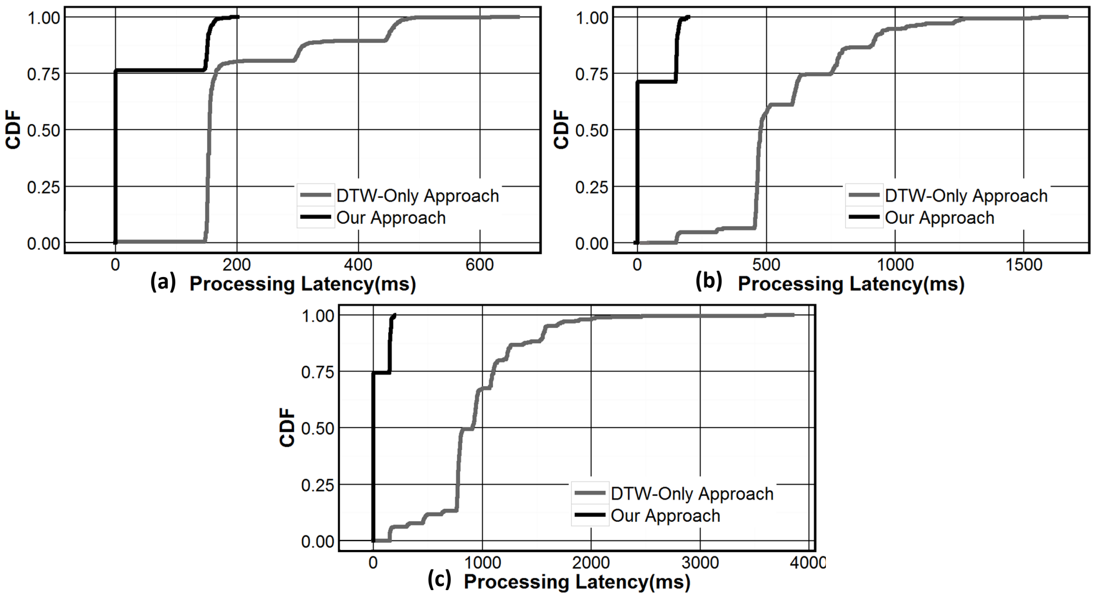


Fig. 12. Processing latency of our approach and the DTW-only approach for (a) one-character gestures, (b) two-character gestures, (c) and three-character gestures. Note that, for DTW, more stroke combinations to test, more the energy is drained.

timeout-based method. Specifically, if no more strokes come within 500 ms, we decide that the previous stroke is the last one. We choose 500 ms as the threshold time, since we found that all inter-stroke time intervals were less than 500 ms in our collected data and this timeout value can be further tuned. If all the segmentation and recognition processing can be done before this 500ms timeout expires, it does not bring in any extra system latency. Therefore, what matters is the extra processing time after the last stroke arrives. We call this the processing latency. Figure 12 shows a case of processing latency of our approach and the DTW-only approach after re-playing 624 1-c, 312 2-c and 216 3-c gesture sets on a Nexus 5 phone. It shows that our approach has a much smaller processing latency than the DTW-only approach. Our segmentation algorithm usually takes less than 10 ms and running DTW-recognition once costs about 150 ms. In our approach, most of the time (75%), all the processing is finished before the 500ms timeout expires, causing zero processing latency. At most, we need to run DTW once (for the last stroke), and thus, the processing latency is always less than 200 ms. More importantly, our approach performs the same no matter how many characters a gesture has. However, in the DTW-only approach, its performance depends heavily on the number of characters of a gesture. As it starts to process the strokes after the 500ms timeout expires, it needs to run DTW-recognition at least once and up to three times for even one-character gestures. For two-character and three-character gestures, it may run DTW-recognition many times, and thus may lead to a very high processing latency, up to more than 3,000 ms. These results show that our threshold-based segmentation algorithm can significantly reduce the processing latency, and thus, energy cost.

10.5 Power Consumption

To measure power consumption, we performed battery drain comparison on the stock ROM for the respective mobile phones against PrivacyShield's ROM. On Nexus 5, Figure 13 (Left) shows the power consumption in all set-ups. Measurements were performed with Monsoon Power Monitor [16]. To highlight the results, Type 4

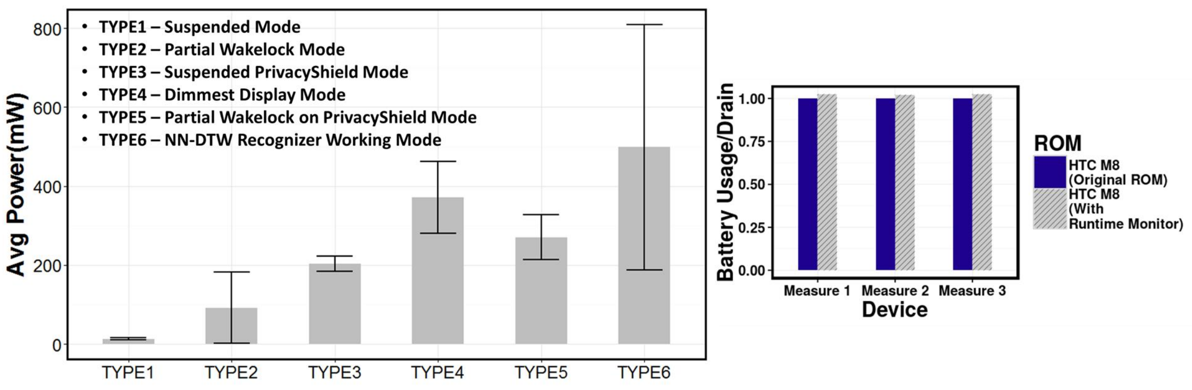


Fig. 13. (Left) Energy consumption of Nexus 5 for different set-ups. (Right) Normalized remaining battery capacity measured on M8 in three trials (Lower is better).

shows an emulated black screen [21]. Type 3 shows the power usage when PrivacyShield is running and the CPU is in a suspended state. Type 5 shows the overhead of keeping the CPU active. PrivacyShield would incur 70mW extra power (which thus negates the fact that the CPU wake-lock results in high power consumption [36]). Type 2 shows the reference partial wake-lock power consumption without PrivacyShield. Type 6 shows the average power consumption for running the whole system in processing a gesture command with a single (two stroke) character gesture. For HTC One M8, we took a software based solution to measure power consumption. Since the device has a non-removable battery we measured the remaining battery from device's battery interface. The device was left idle on a table for 12 hours. To measure the power consumption due to PrivacyShield, we turned off all Wi-Fi and cellular communication so that the measurements would not get effected. Figure 13 (Right) shows the remaining battery capacity of M8 after sitting idle for 12 hours. The power consumption was within 2.5% of the baseline (native ROM with motion launch enabled). Thus, there was no substantial increase in the power usage when running PrivacyShield. There was a minuscule difference when we recorded battery usage while the **`proximity_flag(true)`** because the digitizer was always on while the monitor performed register write and touch-processing work function status check from **`proximity_flag`** state.

11 USABILITY STUDY

Our study procedure consists of two phases. The goal of the *first phase* was to measure users' memorizability of the gesture usages (Initial Testing Phase, §11.1). The *second phase* was to investigate user experience and effectiveness of the gesture usage of PrivacyShield in comparison to an alternate non-touch-screen-based gesture input method (System Deployment Phase, §11.2). Participants for the initial testing and system deployment phases were recruited from our previously used motivational survey.[8] We collected data from 14 participants who participated in the study for compensation.

Table 4 shows the list of participants who participated in the usability study. In the first phase, 8 participants, P_1 to P_8, were invited. They frequently encountered the privacy situations "At least once a day" or "At least once a week" and capitalized upon different privacy-provisioning approaches whenever they encountered such situations. PrivacyShield's **`action`** gesture-*usages* were pre-defined as they are difficult to remember in comparison to user-defined gestures [40]. In the second phase, 6 out of these 8 participants (from the first phase) volunteered for a six-week system deployment study along with 6 newly recruited participants, P_9 to P_{14}. Participants P_9 to

[8]None of the participants participated in the Design Workshop.

Table 4. All participants' demographics and their current privacy-provisioning approaches to avoid information leaks when encountering a device sharing situation.

ID	Gender	Age	Occupation	Hand-orientation	Privacy setting(s) used so far
P_1	M	28	Graduate student	Right handed	Third-party app
P_2	M	21	Undergraduate student	Right handed	Notification hide
P_3	F	22	Undergraduate student	Right handed	Notification hide and third-party app
P_4	M	24	Undergraduate student	Right handed	Notification hide
P_5	F	26	Undergraduate student	Left handed	Notification hide and third-party app
P_6	M	23	Undergraduate student	Right handed	Notification hide and third-party app
P_7	F	24	Undergraduate student	Right handed	Make excuses
P_8	F	18	Undergraduate student	Right handed	Make excuses
P_9	M	29	Graduate student	Right handed	Make excuses
P_{10}	M	26	Graduate student	Right handed	Make excuses
P_{11}	F	25	Graduate student	Right handed	Make excuses
P_{12}	M	27	Graduate student	Right handed	Make excuses
P_{13}	F	22	Undergraduate student	Right handed	Make excuses
P_{14}	F	20	Undergraduate student	Right handed	Make excuses

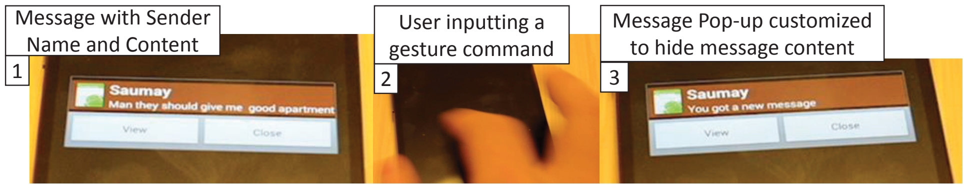

Message with Sender Name and Content
1
Saumay
Man they should give me good apartment
View
Close
User inputting a gesture command
2
Message Pop-up customized to hide message content
3
Saumay
You got a new message
View
Close

Fig. 14. A motivational example from an Instant Messaging application (WeChat) using Privacy Shield on a Nexus 5 phone.

P_{14} never utilized any privacy-provisioning approaches, and were always at risk of exposing their information under device-sharing situations. They would make excuses and avoid any device sharing. P_7 and P_8 did not participate in this phase. After the deployment study, participants P_1 to P_6 compared their user experience with their previously used approaches and PrivacyShield's method of providing privacy-provisioning. Later, we compared every participant's experience in using PrivacyShield and the alternate non-touch-screen-based gesture method.

11.1 Initial Testing Phase

The main aim of this study is to provide empirical evidence on the cognitive cost of the gesture command's `action`-usage corresponding to our system through a four-week field study.

Procedure. After providing their consent and before starting the deployment phase of the experiment, the participants were shown a demo video of the actual PrivacyShield system (running on a Nexus 5 device). The video provided a short version of an example `action` gesture's usage for hiding the message content but not the name from a sender's message notification, as shown in Figure 14. We provided PrivacyShield-enabled Nexus 5 (running KitKat 4.4.4) devices to all the participants and asked them to use the device as their own. After the video tutorial, the participants were taught about the gesture usage of all the `action` gestures. Teaching per participant took about 45 minutes. A roll-back feature followed by an experience-sampling questionnaire was introduced to them. The roll-back feature showed the correct gesture-usage input upon an incorrect gesture-usage entry and requested the user to draw it again. Thereafter, the experience-sampling questionnaire asked the user about the cause for wrong gesture-usage input. We collected participants' gesture-usage data for a week.

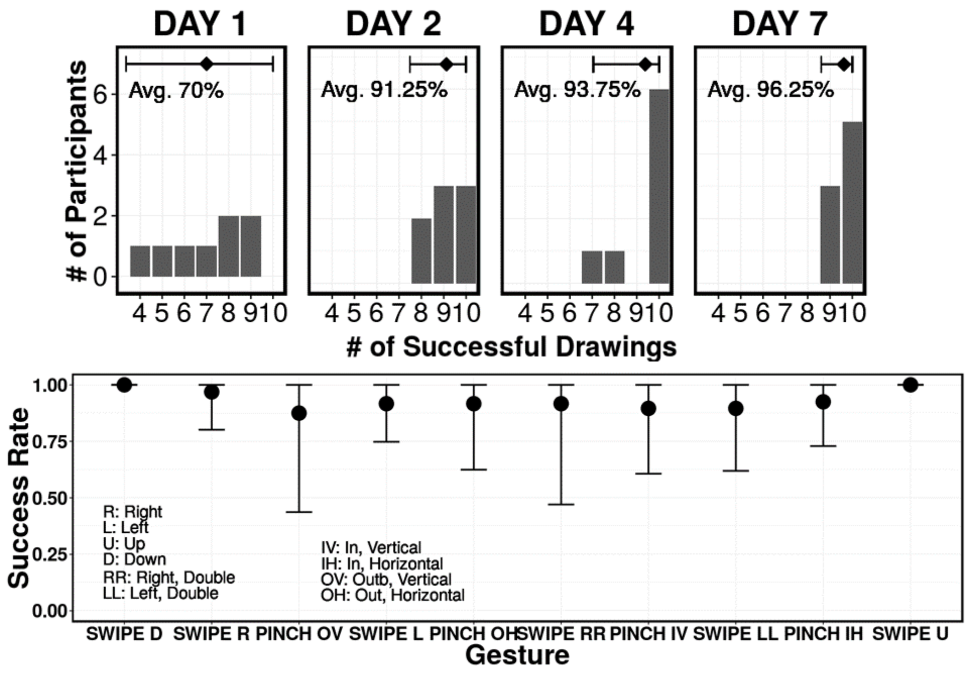


Fig. 15. Top: Histogram showing # of participants correctly recalling *n* gesture usages (x-axis) during the initial testing phase. Bottom: Mean Success Rate of all the participants with respect to all gesture usages. The experiment was performed in 4 periods (Day #). The 3rd and 4th periods started after a gap of one day and two days, respectively. The error bars show a 95% confidence interval.

Initial testing phase performance. This phase was started after a day's gap and comprised 4 periods of testing (Figure 15). Participants were asked to input all the gesture usages (no app customization took place). They were not asked to draw gesture usages in the same order to avoid effects due to sequential memorization. We developed a custom web-based text-message sending service which asked the participants to input the desired gesture action every hour. The reaction time to the text message was also logged by the app. On average, the maximum and minimum response times recorded were 6.59 seconds (*SD = 4.38*) and 0.45 seconds (*SD = 0.25*), respectively. The correctness of the gesture usages was evaluated by our system. The main focus of the study was to measure the success rate and recall. A gesture was considered as correctly recalled if it involved the correct number of fingers (2-fingers) and the right direction of the finger movement for a particular gesture usage (gesture association). A total of 320 gesture-usage inputs were performed by the participants. Each participant performed 10 different gesture usages per day. Figure 15 shows the *next-day* recall rate of all the participants for all the pre-defined gesture usages. A 70% average (*SD = 0.33*) recall was seen after a day's gap. From the experience-sampling questionnaire, we figured out that this lower recall rate was not because of the memorability issue (gesture association) but because of a lack of physical attention, resulting in errors due to *orientation of the finger movement* with respect to the phone. The recall rate improved on successive next-day tests, reaching 96.25% on average (*SD = 0.34*) on the fourth day of test. In Figure 15, the mean success rate was high and the errors were due to a lack of physical attention or focus on the respective gesture-usage inputs. Pinch-out vertical (Precision-based *show* photo) was the only gesture where many participants failed, resulting in an average 87.5% (*SD = 0.22*) success rate.

11.2 System Deployment Phase

In this phase, we verify whether a system like PrivacyShield works well in a realistic environment. To investigate user experience and the effectiveness of PrivacyShield, we conduct an evaluation study that includes: **1)** using PrivacyShield and an alternative *wrist gesture method* for 3 weeks each; and **2)** Post-deployment surveys and interviews. Our motivation to compare PrivacyShield with the wrist gesture method comes from the current trend in smartphone form factor, which is moving towards a button-less design [14]. Therefore, instead of developing, say, a method which uses a combination of button presses, we capitalize on the concept of leveraging wrist gestures.

Wrist gesture method design and implementation. We develop a simple energy-efficient wrist-gesture-based notification and photo hide/show application. The app requires a user to flick her wrist to communicate her intentions of hiding/showing message notifications or photos. To enable the selective hide/show feature on message-pop-ups or photos, the application provides the user with a UI to **1)** include contact group names from a messaging app or **2)** select photos which can be profiled under privacy sensitive groups of name(s) or photo(s) (hereinafter, privacy profile group), respectively. A user can freely add or delete any name(s) or photo(s) based on her privacy preferences. Excepting the privacy profile group, the rest of the message pop-ups' functionality is kept to the default "Don't silence or block" (e.g., from Android OS). Similarly, all photos are accessible to the user if they are not selected to be part of the privacy profile group. For switching from privacy modes to the normal mode, user authentication is required.

We implemented four wrist gesture commands: **"DownFlick"** → "**Selective message-pop-ups hide**", "**2-DownFlicks**" → "**All message-pop-ups show**", "**UpFlick**" → "**Selective photos hide**", and "**2-Upflicks**" → "**All photos show**" on Nexus 5 smartphones. The gesture commands are activated as **`gesture commands`** → **`screen lock`** → **`system activation`**. For all **`DownFlick`** gestures, no vibration (as a feedback) was given by the app, while for **`UpFlick`** gestures, a long vibration of one second was given. "**2-DownFlicks**" and "**2-Upflicks**" were used as override gesture commands. The wrist gesture application is implemented by leveraging a multi-situation HMM classifier running a collaborative gesture-sensing and segmentation architecture [43]. The app runs by acquiring a partial CPU wake-lock. The total average device power consumption in running the app is ~86mW. The application achieves a gesture recognition accuracy of ~96% for 180 gesture samples collected for each gesture types from all 12 participants who performed the gestures while standing (one-third used for testing). Moreover, the processing time per gesture usage is ~2,400ms. All experiments were performed before the start of system deployment. The application leverages the **`AccessibilityService`** object and **`ContentResolver`** object to enable the dynamic control of pop-up messages and photo hide/show functionality, respectively (§8).

System deployment procedure. After receiving their consent, we introduced the features of both PrivacyShield and the wrist gesture method to the participants; and gave them a tutorial on how to use both the methods. All participants used Nexus 5 as their personal device for 6 weeks. We divided the 12 participants into two groups: Group A and Group B. P_1 to P_6 belonged to Group A, while P_9 to P_{14} comprised Group B. Each group used both privacy-provisioning approaches in two stages. In the first stage of the deployment study, Group A participants used PrivacyShield while Group B participants used the wrist-gesture-based method for 3 weeks each. In the next phase, Group B used PrivacyShield while Group A used the wrist-based method for another 3 weeks. Each participant was asked to realistically use PrivacyShield and the wrist gesture method whenever they thought their information from the device could get snooped on. We narrowed down our deployment study to subtle privacy-aware apps to better understand PrivacyShield's **"`action[parameter]`"** gesture-command use against the profile-based wrist gesture method. Participants were asked about the number of people with whom they frequently interacted using their messaging client (participants used their preferred messaging app, i.e., Kakao Talk). Most of the participants reported that they frequently interacted with no more than five people on a regular

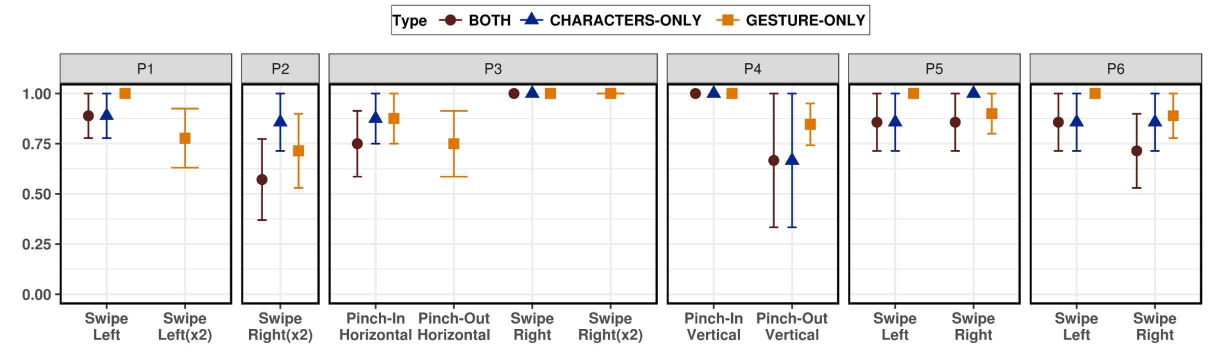


Fig. 16. The Mean Success Rate (first attempt) of all gesture usages. No participant failed on their second attempt.

basis. This excluded group chats as the interaction is more sporadic than regular in such chat rooms. Also, a previous study [44] reported that users mostly interact with not more than twelve people (on a regular basis) on their chat client (including group chat rooms). Through the experience-sampling questionnaire, we asked the participants to describe the episode(s) of just-in-time privacy provisioning,[9] along with names of the persons for whom they entered respective gesture commands.

11.2.1 Effectiveness of PrivacyShield in Terms of Privacy Provisioning. In total, we recorded 153 privacy sessions from participants of Group A and B based on the criteria that a participant used gesture command after sensing the gravity of a possible device sharing or borrowing situation; and reverted the privacy setting to the normal setting after such situation was over. We consider the start of a privacy session when a Hide or Show (with or without "**`[parameter]`**") is activated by a participant. We consider the end of a privacy session when a user either uses an override "**`action`**" command without a "**`[parameter]`**" or revert a privacy session from the system settings.

From the six participants belonging to Group A, we collected 74 privacy-configuration sessions. In total, 106 "**`action`**" gestures and 109 "**`[parameter]`**" gestures were recorded. Figure 16 shows the first-attempt success rate with respect to each gesture use and the participants. In all cases, no participant drew more than one gesture use at a time. We recorded a maximum number of failed first-time attempts (by monitoring screen reset). No participant failed on their successive attempt. The overall participants' success rate ("**`action[parameter]`**" combined) was ~81.4% (Min ~57.1% (P_2) and Max ~87.5% (P_3)). The maximum number of privacy sessions recorded was 19 (P_6). For ending a privacy session, 32 "**`action`**" gestures were recorded (P_1, P_3, and P_4), while 42 sessions were ended with system setting configuration (P_2, P_5, and P_6). The minimum and maximum times taken to revert from a privacy session to the normal setting were 3.09 minutes and 62.12 minutes (*M = 24.21, SD = 16.94*), respectively.

In terms of using "**`[parameter]`**" suffix in a gesture command, we found that most of the participants used parameter suffix frequently. "**`action`**" gesture alone was only used for 5 times in the case of P_5 (in 2 out of total 15 privacy sessions and gesture commands) and P_6 (in 3 out of 19 privacy sessions and gesture commands). P_1 (9 privacy sessions, total 18 gesture commands), P_2 (8 privacy sessions and gesture commands), P_3 (13 privacy sessions, total 26 gesture commands) and P_4 (10 privacy sessions, total 20 gesture commands) always used "**`[parameter]`**" suffix. P_3 and P_4 were the two people who used Pinch gestures, as shown in Figure 16. From the experience-sampling comments, we figured out that both participants had two names beginning with the same character and required separate priority in different scenarios. Also, P_2 used the selective show option to start all of his privacy sessions, while P_4 utilized the selective show option in 3 of his privacy sessions. Therefore, total 11

[9]Participants were asked to enter their experience at the end of each day of gesture-command use.

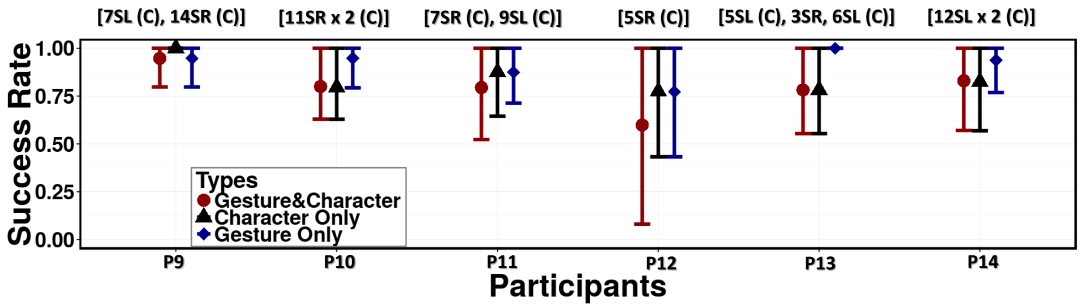


Fig. 17. First attempt Mean Success Rate of all gesture usages. Gesture-only statistics include reverting PrivacyShield to normal-setting cases. The values above show the type and number of times a gesture command was used to *start* a particular privacy session. *SL* : Swipe Left, *SR* : Swipe Right, $SL \times 2$: double Left Swipe, $SR \times 2$: double Right Swipe. *SL(C)/SR(C) or* $SL \times 2(C)/SL \times 2(C)$: "action" gesture followed by a "parameter" gesture. No participant failed on their second attempt.

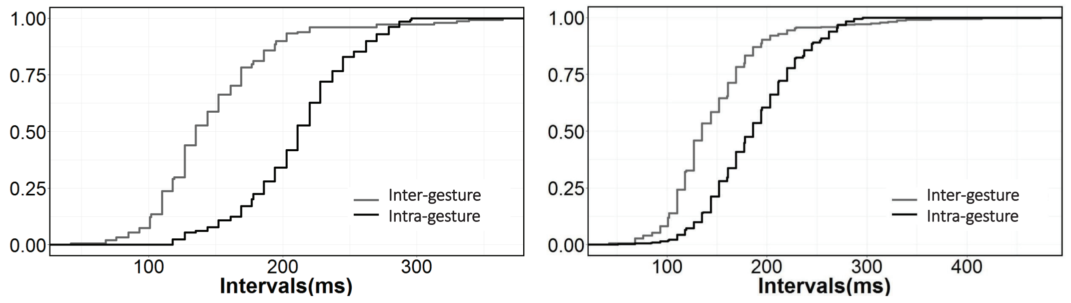


Fig. 18. (Left) Inter- and Intra-Stroke interval between gestures (for Group A participants). 27 two-length gestures and 82 three-length gestures were recorded for configuring privacy sessions. (Right) Inter- and Intra-Stroke interval between gestures (for Group B participants). Both measurements include "action" gestures in them.

times selective show gesture-usages were recorded to start a privacy session. The maximum number of names used from the participants' contact list and name-tagged photos was 3, for P_5; whereas minimum was 1 in the case of P_1. Rest of the participants: P_3, P_2, P_4 and P_6 utilized 2 names. Only P_2 used two names at once in each of his privacy sessions.

For the second group, Group B, we collected a total of 79 privacy-configuration sessions. As shown in Figure 17, the overall participants' success rate (**"action[parameter]"** combined) was ~80% (Min ~60% (P_{12}) and Max ~94% (P_9)). Selective hide/show (and no Pinch based hide/show) **"action[parameter]"** gestures were frequently used by all the participants. P_9 (21 privacy sessions (maximum privacy sessions), total 42 gesture commands), P_{10} (11 privacy sessions, total 22 gesture commands), P_{11} (16 privacy sessions, total 32 gesture commands), P_{13} (14 privacy sessions, total 28 gesture commands), and P_{14} (12 privacy sessions, total 24 gesture commands) all used the gesture-override option, except P_{12} (5 privacy sessions), who used the system settings to end a privacy session. 3 **"action"**-only gestures were recorded in case of P_{13}. P_{10} utilized a maximum of 4 names from his contact list and used the selective show option to hide message notifications in all of his privacy sessions. Similarly, P_{14} used the selective show option to hide her personal photographs in all her privacy sessions. In total, 23 privacy sessions were started with the selective show option. The rest of the participants utilized 2 names to configure their privacy settings. The minimum and maximum times taken to revert from a privacy session to the normal setting were 5.05 minutes and 56.55 minutes (*M = 23.56, SD = 19.32*), respectively.

Swiftness in configuring privacy settings. All the participants were able to perform gesture inputs within the processing latency of the system. As shown in Figure 18 (Left), for the Group A participants, the maximum speed between two gesture commands was 340ms. Furthermore, the maximum total time to input gesture commands was 3,200ms. Similarly, for the Group B participants, the maximum total time to input gesture commands was 2,900ms, while the maximum inter-gesture drawing time was 230ms (Figure 18 (Right)). The results highlight the fact that gesture commands were inputted in a swift way.

System utilization in the middle of device sharing. From the experience-sampling questionnaire, participants reported instances of using the system's gesture commands while they had their devices' screens shared with other people. We found that ~15.45% (9.45% in the case of Group A participants and 6% in the case of Group B participants) of the privacy sessions were initiated while the display was on, for example, when a participant was engaged in a Skype call with colleagues.

11.2.2 Effectiveness of the Wrist Gesture Method in Configuring Privacy. The wrist gesture method recorded a low number of privacy sessions. In total, 75 privacy sessions were recorded (Group A: 34 privacy sessions; and Group B: 41 privacy sessions), which was ~2x lower than PrivacyShield-based privacy-provisioning sessions. 64 privacy sessions were ended using the gesture-override concept while the rest were ended using the app settings. Overall, the participants' success rate was ~93.25% *(SD = 0.024)* from a total of 128 gesture usages, which was ~12% higher than from the PrivacyShield method. This reflects the fact that the wrist gesture method is simpler in provisioning a privacy configuration. The second attempt at gesture inputting recorded no failures. Participants updated their privacy profile group at least 3 times (P_{11}) during the deployment study. The maximum number of updates was from P_{10}. He updated the privacy profile group 14 times. The minimum and maximum times taken to revert from a privacy session to the normal setting were 5.21 minutes and 53.38 minutes (*M = 21.75, SD = 17.29*), respectively.

System utilization in the middle of device sharing. No participants configured their privacy settings while their screen was turned on. This fact highlights the effectiveness of PrivacyShield over the wrist gesture method in utilizing gesture commands use even during screen-sharing. In the next subsection, we discuss some experiences of the participants which resulted in a lower number of privacy sessions with the wrist gesture method.

11.2.3 User Experience. We evaluated user experience via three post-deployment surveys and two interviews. The first survey compared satisfaction levels in using PrivacyShield against previously used privacy-provisioning methods (from the design study questionnaire (§2)). The second survey asked participants to compare PrivacyShield with the wrist gesture method. Finally, the third survey measured user experience specific to PrivacyShield and the study. We conducted two one-hour interviews to collect their experiences during episodes of using PrivacyShield. In the first interview, we investigated the change in the participants' (P_1 to P_6) perceptions (from their previously used methods) of configuring privacy settings after using PrivacyShield. In the second interview, we explored the differences between their experiences of using PrivacyShield and the wrist gesture method. We walked through the results together to ask follow-up questions on why such results occurred. All interviews were recorded, transcribed, and reviewed. After open coding of the transcripts, we used axial coding to examine user responses. For the first interview, we conducted affinity diagramming to narrow the codes into a set of two main themes [49]: *(a)* Swiftness in configuring privacy settings; and *(b) In-situ* usability with selective hide/show options. In the second interview, affinity diagramming narrowed the codes into a set of two main themes: *(a) In-situ* flexibility in configuring *subtle* just-in-time privacy; and *(b)* Multi-situational gesture command input accessibility.

Satisfaction of using PrivacyShield against previously used methods. For the participants who used past methods to prevent information leaks (P_1 to P_6), we asked them to rate their satisfaction on effort required to configure privacy settings and available privacy options (on a 6-point Likert scale, §2.1).

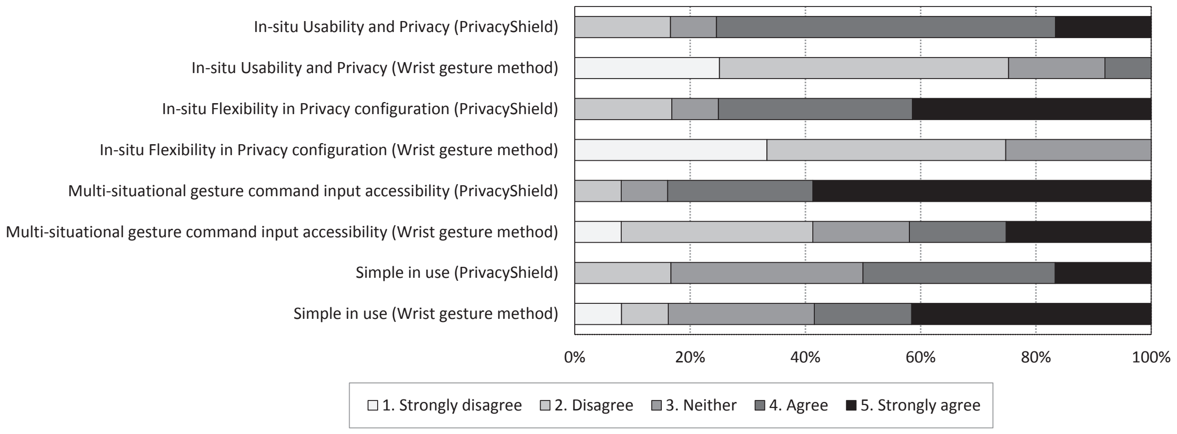


Fig. 19. Participants' ratings of PrivacyShield and the alternative wrist gesture method.

(a) Swiftness in configuring privacy settings. Overall, PrivacyShield improved the participants' level of satisfaction in terms of effort required in privacy configuration. The post-deployment questionnaire yielded a higher satisfaction score (*$M = 3.66$, $SD = 0.51$*) as compared to the initial questionnaire (*$M = 2.33$, $SD = 0.81$*). The difference was statistically significant (*$t = -3.38$, $p < 0.003$*). Four participants responded that their satisfaction with hiding information has increased in comparison with the previously used privacy method. P_3 recalled an experience from her meeting with her friend, where her boyfriend sent her a message asking her to come on Skype. She stated: "*When using PrivacyShield, I don't have to pay attention for inputting gestures. Earlier I used to precisely slide the notification hide bar, else, it removed all my messages instead of showing me notification hide options.*" P_5 mentioned: "*One slide gesture and it was done! I can do this while talking to anyone. No-one was able to notice.*" P_6 really liked the idea of swiftly hiding photographs at will. P_6 stated: "*It's very easy to hide photos with PrivacyShield, I don't have to invest time in categorizing the photographs under private and public. Anytime I input a gesture [command], I can get the desired result.*"

Two participants, P_2 and P_4, whose satisfaction did not change in comparison to their previously used methods stated that they already used the "always notification hide" option in their existing privacy configuring method. With PrivacyShield, they are supposed to enter a pass-code every time they had to configure a privacy setting, which was not very easy when hiding message notifications in front of observer(s). P_4 mentioned: "*Entering a pass-code is still an extra overhead for me and I do not think that this is any different from the existing privacy setting configuration step.*"

(b) Satisfaction in using the selective hide/show options. All the participants were quite satisfied after using PrivacyShield in terms of available privacy options. The post-deployment questionnaire showed a higher satisfaction score (*$M = 4.3$, $SD = 0.51$*) than the initial questionnaire (*$M = 1.83$, $SD = 0.40$*); the difference was statistically significant (*$t = -9.30$, $p < 0.01$*). P_2 mentioned: "*Using friend's name as a gesture was easier than what I thought before using PrivacyShield.*" P_1 stated: "*Earlier when I was supposed to show a photograph, I made sure that I perform the photo browsing task [discreetly] instead of the observer. Now [after using PrivacyShield] I don't have to bother about such situations while handing over the phone.*"

Comparing satisfaction between PrivacyShield and the wrist gesture method. Participants rated both PrivacyShield and the wrist gesture methods in terms of usability, flexibility, multi-situational gesture command input accessibility, and simplicity in provisioning privacy configurations, on a 5-point Likert scale. Figure 19 shows that most of the participants agreed or strongly agreed that PrivacyShield provided more usability and privacy in case of device borrowing or screen sharing, with ~8% agreeing in favor of the wrist gesture method. In terms of flexibility in configuring privacy, PrivacyShield received over ~60% agreement, while no participant agreed in favor of the wrist gesture method. These results illustrate the low number of privacy sessions recorded from Group A and Group B during the system deployment phase. In terms of multi-situational accessibility and simplicity, more than ~50% of the participants voted (in both the cases) in favor of PrivacyShield's method. Many participants also agreed or strongly agreed that the wrist gesture method was also simple in privacy-provisioning tasks (~58%). The differences between the two methods were not significant (Wilcoxon signed-rank tests, all $p > 0.05$).

Our post-deployment interview centered around two main themes which cover the important aspects of our system's design and implementation:

(a) *In-situ* flexibility in configuring *subtle* just-in-time privacy. Most participants agreed that PrivacyShield provided more *in-situ* flexibility in configuring selective hide/show options. In response, the wrist gesture method was not liked much by the participants. P_{10} mentioned: "*In the beginning, I tried to add the rule right after I faced the situation where my message was unwantedly seen. But it was weird to do it in front of that person, so I would call to mind about the rule every night at home. This required some thinking effort regarding whom I would meet the next day.*" P_9 mentioned: "*Many times I was in a state of confusion, as I wanted to hide some notification from a particular person only, but it [the wrist gesture method] resulted in hiding messages from all the contacts in the [privacy profile] group. Overall, I couldn't capitalize on the fastness of the [wrist gesture] method and many times ended up in making excuses.*" On a similar note, P_4 stated: "*I thought I had one name to hide message notifications from and that too when I was driving with my lab-mates. Later, it turned out that one name in the [privacy profile] group was not enough as all other notifications from other people popped-up. In such cases, I used the selective show option.*" P_{12} and P_{14} disagreed on the flexibility in configuring the selective hide/show options in case of both PrivacyShield and the wrist gesture method. P_{12} stated: "*Several times I missed inputting gestures and also forgot to update the [privacy profile] group. I know it sounds ridiculous, but can't the system configure the rules for itself?*"

(b) Multi-situational gesture command input accessibility. Many participants agreed that PrivacyShield provided better gesture command input accessibility, irrespective of the device's placement or sharing state. P_4 said: "*In most cases my phone was hooked to a placeholder in my car. I had to explicitly perform the task of enabling privacy [in case of the wrist gesture method].*" PrivacyShield was also effective in its use when participants wanted to hide information while screen sharing their device. P_{13} mentioned: "*It is nice! It [PrivacyShield] was especially helpful whenever I wanted to hide some photos of mine in the middle of any photo-sharing task. Doing it with the wrist gesture method was really not possible.*"

PrivacyShield experience feedback. Based on their experience in using PrivacyShield, we asked the participants about aspects which were specific to PrivacyShield and the user study, on a 5-point Likert scale (Figure 20). Participants also explained their opinions on the decisions in a comments section.

Seven participants mentioned that they would like to use PrivacyShield in the future. All the participants who disagreed or were neutral in their opinion left a comment to explain their decision. P_{12} and P_{14} said that the effort required in achieving usability was very high. P_2 and P_4 were undecided whether to stick to their previous methodology of keeping all the information hidden or use PrivacyShield. P_{11} stated that she was not certain which privacy method (PrivacyShield or the wrist gesture method) to adopt. We further highlighted some of the major findings from our experience-feedback study.

User-triggered system initialization was not easy for everyone. 7 participants agreed or strongly agreed that system initialization posed a hurdle in entering gestures. P_3 specifically mentioned: "*Since I use PrivacyShield for two different purposes [notification hide and photo hide], I would like the system to always be available.*" For such foreseen reasons, we have implemented a low-power co-processor version of the PrivacyShield prototype (§9).

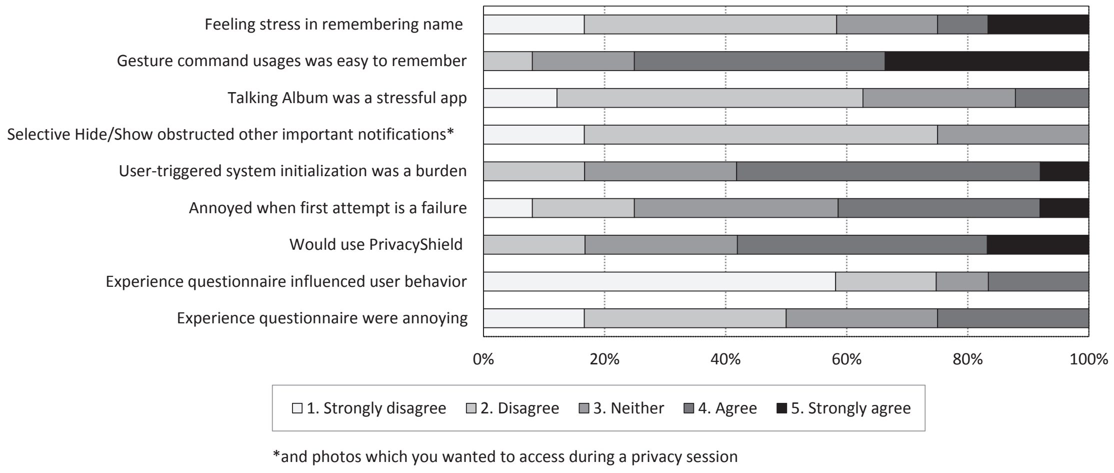


Fig. 20. User experience specific to PrivacyShield and the study.

Participants were asked whether they were unable to access any important notification(s) from a person or were unable to view photograph(s) of any person who was sharing the same first-name initials. More than two-thirds of the participants (9 participants) disagreed, while the rest gave neutral ratings. The 3 participants (P_5, P_6 and P_{13}) who had a neutral view point, stated that a few times they had difficulties in accessing information (from message notifications), but privacy was more important to them in such cases. They all used the all-hide option (**"2-finger Right Swipe"**) in such cases. Moreover, they were aware about the name-initial collision, and therefore, were able to check any (expected) important message discreetly. Also, such events were not frequent during their use of PrivacyShield.

Tagging photos using speech-generated name tags was not considered as an overhead by many participants. Out of 8 participants who used Talking Album, neutral (P_3, P_9) or agreement (P_{14}) votes were seen in cases wherein the participants mentioned that they were supposed to be slow and clear when pronouncing the name of a person. For such participants, explicit speech tagging was common (6 in the case of P_3, 11 in the case of P_9, and 19 in the case of P_{14}). The remaining 5 participants used speech tagging only once to categorize photos under a given name and used Talking Album's auto-tagging feature. Finally, all the participants had a common view point that the app inherently helped them provide awareness in deciding whether to tag (or not) an image, keeping in mind the privacy level associated with it. Previously (and in the wrist gesture method) they used to filter images under a privacy profile group, based on whether they would be leaked to a person or not.

11.3 Summary of Findings

First, the overall participants' success rate in using PrivacyShield was stable in the case of both groups. The system design was easy to make participants adapt to the gesture usages. They were quick in drawing gesture commands which helped them in swiftly configuring their desired privacy settings. Many participants also got accustomed to reverting a privacy configuration to the normal setting using the override **"action"** commands. Second, the type of gesture commands used by the participants to activate a privacy session was not more than two. Also, the number of names used was not more than 4. However, we figured out that the need to hide important information is sporadic and may often not be forethought. Therefore, a simple privacy-provisioning method (the wrist gesture method) was found limiting in providing participants *in-situ* flexibility to achieve a desired privacy configuration. From our experience study, we also figured out that people prefer to attain a desirable privacy configuration right after perceiving the risk of a tentative information leak. Third, PrivacyShield was able to provide scalability in hiding multiple contacts and name-tagged photos. In our deployment study, 34 privacy sessions (~22.2%) were initiated with the help of selective show-gesture commands. This additionally shed light on the fact that participants were aware of the nature and capability of gesture commands and usages.

12 RELATED WORK

Non-touchscreen RFID based gesture interaction. Before implementing the PrivacyShield mock-up system, we explored some non-touchscreen and RFID based gesture interaction techniques, e.g., LLAP [52], FingerIO [41], and Pantomime [46]. LLAP [52] and FingerIO [41] uses CW (Continuous Wave) signal and OFDM pulses to understand phase changes of sound signals to locate fingers. Both techniques allow device-free tracking of finger movement but are slow in drawing gestures and does not work for non-cursive gesture strokes. Pantomime [46] uses two RFID tags on an object and make use of a single antenna in the vicinity to track device motion. This approach is also slow in drawing gestures and depends upon the availability of a RFID reader/antenna. Since we target *in-situ* privacy configuration for privacy situations occurring sporadically in nature, RFID based gesture techniques do not fit our requirement.

Mobile systems for privacy protection. Several mobile systems have been proposed to protect unwanted access to specific applications and data in phone- sharing or borrowing situations [34][22][42][45]. xShare [34] enables multiple virtual environments on a single phone, each with different data and app access permissions. Similarly, DiffUser [42] provides differentiated user access control to system resources. Cells [22] virtualizes multiple android runtimes for different uses. These systems focus on access control to mobile data and resources, and they all require explicit configurations from users to change access permissions or switch between virtual environments. Our work is complementary to them, as we focus on how to make the configurations subtle and dynamic. In addition, TreasurePhone [45] proposes the concept of context-aware data-protection on mobile phones, such as providing different permissions depending on the user's location and actions. It focuses on a user study on the concept without building a working system.

Subtle and deceptive interactions. There are some studies on subtle and deceptive interactions [21][23] which focuses on miniaturizing and reducing the cognitive load of interactions. Without building solid systems, they demonstrate the need of subtle interactions in various scenarios and thus are motivational to our work.

Techniques on gesture recognition. Many research efforts have been spent on various gesture-recognition techniques, such as tablature based [31] and example based [35][51][53]. Optical Character Recognition (OCR) is a widely used technique for handwriting recognition [25][26]. Most of those existing approaches are either designed to work assuming that visual feedback is available or depend on users for character segmentation, and thus, cannot address the unique challenges in recognizing subtle privacy-provisioning gestures, particularly the gesture-overlap problem.

Gesture shortcut technique. GestureOn presents an idea of using gesture shortcuts [36], but does not provide a detailed implementation on the system part. Moreover, GestureOn leverages Gesture Search framework which limits in drawing fast gesture inputs. Though this work is motivational, PrivacyShield is the first work which provides an end to end system implementation and addresses all the intricacies in developing the system.

Soft keypad based approach. Soft keypads consume the mobile phone's limited screen real estate, and thus limit users to draw a gesture in a given space. Pull-down or pull-up based on-screen keyboards take longer time. Moreover, it takes time for users to scan and locate the correct keys [55].

13 DISCUSSION

Limitations. Our current PrivacyShield system and its user study evaluation is limited in several aspects. First, our usability results are inferential. This type of study is difficult (owing to custom device use) and does not capture a person's personality with the system use. Moreover, results are subjected to frequency of device borrowing and sharing situations. Since the frequency of device borrowing and sharing situations in the current study is low, future work should consider a larger sample. Second, it supports only lower-case characters. Nevertheless, we plan to support upper-case characters to further improve the capability of PrivacyShield.

Knowledge of the observer and subtlety in inputting gestures. It is possible that if PrivacyShield enabled devices become prevalent, the set of possible privacy configurations may also become well known, and therefore easier for observers to spot. However, if this happens, then the set of possible privacy customizations will be so great as to make observers suspicious of every time the user interacts with her device.

Future Work. At last, we will continue to improve the gesture-recognition algorithms and the UI design to optimize the system performance and to make PrivacyShield easier to use. We will also continue to develop more PrivacyShield apps. We also plan to open source our PrivacyShield implementation along with the interesting testing application we built.

14 CONCLUSION

In this paper, we have designed, implemented and evaluated PrivacyShield, the first subtle just-in-time privacy-provisioning system on smartphones. PrivacyShield allows users to use touch gestures to dynamically change the privacy-protection policies of the system, and thus bridges the gap between privacy and usability in contextual privacy provisioning. We design a novel stroke-based segmentation and recognition approach for gesture recognition, and develop several power-optimization techniques inside the device kernel. Evaluation results using real-user data show that our gesture-recognition approach is swift and accurate, and our power-optimization techniques are able to effectively reduce the power consumption of PrivacyShield.

ACKNOWLEDGMENTS

We are thankful to all the anonymous reviewers who dedicated their time to provide instrumental and constructive comments which greatly helped bring the paper to its current form. We would also like to thank Bumsoo Kang for all the valuable discussions during the writing of this paper. This research was partly supported by Basic Science Research Program (NRF-2017R1A2B3010504), Next-Generation Information Computing Development Program (NRF-2017M3C4A7066473) through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT; Microsoft Research Asia and the Ministry of Science and ICT, Korea, under ICT/SW Creative research program supervised by the Institute for Information & Communications Technology Promotion (IITP) (IITP-2015-R2215-15-1023).

REFERENCES

[1] 1999. IRONOFF DATABASE UMR CNRS 6597. http://www.irccyn.ec-nantes.fr/~viardgau/IRONOFF/IRONOFF.html. [Online; accessed 18-April-2018].
[2] 2009. Protractor's Java implementation in the Android core framework. https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/gesture/GestureUtils.java. [Online; accessed 18-April-2018].
[3] 2010. Kakao Talk. http://www.kakao.com/services/8. [Online; accessed 18-April-2018].
[4] 2011. WeChat. http://www.wechat.com/en/. [Online; accessed 18-April-2018].
[5] 2013. LG Nexus 5. http://www.lg.com/us/cell-phones/lg-D820-Sprint-Black-nexus-5. [Online; accessed 18-April-2018].
[6] 2014. Android Lollipop. https://www.android.com/versions/lollipop-5-0/. [Online; accessed 18-April-2018].
[7] 2014. Android Lollipop Features. https://www.android.com/versions/lollipop-5-0/#features. [Online; accessed 18-April-2018].

[8] 2014. HTC one (M8). http://www.htc.com/us/smartphones/htc-one-m8/. [Online; accessed 18-April-2018].
[9] 2015. Android Marshmallow. https://www.android.com/versions/marshmallow-6-0/. [Online; accessed 18-April-2018].
[10] 2015. Smart Gallery - Photography. https://play.google.com/store/apps/details?id=com.ta.voicetag.main. [Online; accessed 18-April-2018].
[11] 2016. Nougat. https://www.android.com/versions/nougat-7-0/. [Online; accessed 18-April-2018].
[12] 2016. Project Oxford. https://www.projectoxford.ai/doc/speech/Get-Started/android/. [Online; accessed 18-April-2018].
[13] 2017. Apple - Change alert styles and settings for notifications. https://support.apple.com/en-au/HT201925. [Online; accessed 18-April-2018].
[14] 2017. iPhone X. https://www.apple.com/lae/iphone-x/specs/. [Online; accessed 18-April-2018].
[15] 2017. OpenCamera. http://opencamera.sourceforge.net/. [Online; accessed 18-April-2018].
[16] 2017. Power Monitor. Monsoon Solution Inc. https://www.msoon.com/LabEquipment/PowerMonitor/. [Online; accessed 18-April-2018].
[17] 2018. Gallery Vault. https://play.google.com/store/apps/details?id=com.thinkyeah.galleryvault&hl=en. [Online; accessed 18-April-2018].
[18] 2018. Hide Pictures Videos - Vaulty. https://play.google.com/store/apps/details?id=com.theronrogers.vaultyfree&hl=en. [Online; accessed 18-April-2018].
[19] 2018. Indexing and retrieving contact IDs. https://developer.android.com/training/contacts-provider/retrieve-names.html. [Online; accessed 18-April-2018].
[20] 2018. Smart Lock - Security simplified. https://get.google.com/smartlock/. [Online; accessed 18-April-2018].
[21] Fraser Anderson, Tovi Grossman, Daniel Wigdor, and George Fitzmaurice. 2015. Supporting Subtlety with Deceptive Devices and Illusory Interactions. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 1489–1498. https://doi.org/10.1145/2702123.2702336
[22] Jeremy Andrus, Christoffer Dall, Alexander Van't Hof, Oren Laadan, and Jason Nieh. 2011. Cells: A Virtual Mobile Smartphone Architecture. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP '11)*. ACM, New York, NY, USA, 173–187. https://doi.org/10.1145/2043556.2043574
[23] Daniel Ashbrook, Patrick Baudisch, and Sean White. 2011. Nenya: Subtle and Eyes-free Mobile Input with a Magnetically-tracked Finger Ring. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 2043–2046. https://doi.org/10.1145/1978942.1979238
[24] Alexander De Luca, Emanuel von Zezschwitz, Ngo Dieu Huong Nguyen, Max-Emanuel Maurer, Elisa Rubegni, Marcello Paolo Scipioni, and Marc Langheinrich. 2013. Back-of-device Authentication on Smartphones. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 2389–2398. https://doi.org/10.1145/2470654.2481330
[25] Thomas Deselaers, Daniel Keysers, Henry Rowley, Li-Lun Wang, Victor Cărbune, Ashok Popat, and Dhyanesh Narayanan. 2017. Google Handwriting Input in 82 languages on your Android mobile device. https://research.googleblog.com/2015/04/google-handwriting-input-in-82.html. [Online; accessed 18-April-2018].
[26] Dmitriy Genzel, Ashok C. Popat, Nemanja Spasojevic, Michael Jahr, Andrew Senior, Eugene Ie, and Frank Yung-Fong Tang. 2011. Translation-Inspired OCR. In *ICDAR-2011*.
[27] Nicholas Gillian, R. Benjamin Knapp, and Sile O'Modhrain. 2011. Recognition Of Multivariate Temporal Musical Gestures Using N-Dimensional Dynamic Time Warping. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. 337–342.
[28] Otto Huhta, Swapnil Udar, Mika Juuti, Prakash Shrestha, Nitesh Saxena, and N. Asokan. 2017. Pitfalls in Designing Zero-Effort Deauthentication: Opportunistic Human Observation Attacks. In *23nd Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 21-24, 2017*.
[29] Amy K. Karlson, A.J. Bernheim Brush, and Stuart Schechter. 2009. Can I Borrow Your Phone?: Understanding Concerns when Sharing Mobile Phones. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 1647–1650. https://doi.org/10.1145/1518701.1518953
[30] Wolf Kienzle and Ken Hinckley. 2013. Writing Handwritten Messages on a Small Touchscreen. In *Proceedings of the 15th International Conference on Human-computer Interaction with Mobile Devices and Services (MobileHCI '13)*. ACM, New York, NY, USA, 179–182. https://doi.org/10.1145/2493190.2493200
[31] Kenrick Kin, Björn Hartmann, Tony DeRose, and Maneesh Agrawala. 2012. Proton++: A Customizable Declarative Multitouch Framework. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12)*. ACM, New York, NY, USA, 477–486. https://doi.org/10.1145/2380116.2380176
[32] S. Kurkovsky and E. Syta. 2010. Digital natives and mobile phones: A survey of practices and attitudes about privacy and security. In *2010 IEEE International Symposium on Technology and Society*. 441–449. https://doi.org/10.1109/ISTAS.2010.5514610
[33] Yang Li. 2010. Gesture Search: A Tool for Fast Mobile Data Access. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology (UIST '10)*. ACM, New York, NY, USA, 87–96. https://doi.org/10.1145/1866029.1866044
[34] Yunxin Liu, Ahmad Rahmati, Yuanhe Huang, Hyukjae Jang, Lin Zhong, Yongguang Zhang, and Shensheng Zhang. 2009. xShare: Supporting Impromptu Sharing of Mobile Phones. In *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services (MobiSys '09)*. ACM, New York, NY, USA, 15–28. https://doi.org/10.1145/1555816.1555819

[35] Hao Lü, James A. Fogarty, and Yang Li. 2014. Gesture Script: Recognizing Gestures and Their Structure Using Rendering Scripts and Interactively Trained Parts. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 1685–1694. https://doi.org/10.1145/2556288.2557263

[36] Hao Lu and Yang Li. 2015. Gesture On: Enabling Always-On Touch Gestures for Fast Mobile Access from the Device Standby Mode. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 3355–3364. https://doi.org/10.1145/2702123.2702610

[37] Diogo Marques, Ildar Muslukhov, Tiago Guerreiro, Luís Carriço, and Konstantin Beznosov. 2016. Snooping on Mobile Phones: Prevalence and Trends. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*. USENIX Association, Denver, CO, 159–174. https://www.usenix.org/conference/soups2016/technical-sessions/presentation/marques

[38] Chulhong Min, Saumay Pushp, Seungchul Lee, Inseok Hwang, Youngki Lee, Seungwoo Kang, and Junehwa Song. 2014. Uncovering Embarrassing Moments in In-situ Exposure of Incoming Mobile Messages. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication (UbiComp '14 Adjunct)*. ACM, New York, NY, USA, 1045–1054. https://doi.org/10.1145/2638728.2641288

[39] Patrick Mochel. 2005. The sysfs Filesystem. In *Proceedings of the Linux Symposium, Vol. 1*. 313–326.

[40] Miguel A. Nacenta, Yemliha Kamber, Yizhou Qiang, and Per Ola Kristensson. 2013. Memorability of Pre-designed and User-defined Gesture Sets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 1099–1108. https://doi.org/10.1145/2470654.2466142

[41] Rajalakshmi Nandakumar, Vikram Iyer, Desney Tan, and Shyamnath Gollakota. 2016. FingerIO: Using Active Sonar for Fine-Grained Finger Tracking. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 1515–1525. https://doi.org/10.1145/2858036.2858580

[42] Xudong Ni, Zhimin Yang, Xiaole Bai, A. C. Champion, and D. Xuan. 2009. DiffUser: Differentiated user access control on smartphones. In *2009 IEEE 6th International Conference on Mobile Adhoc and Sensor Systems*. 1012–1017. https://doi.org/10.1109/MOBHOC.2009.5337017

[43] Taiwoo Park, Jinwon Lee, Inseok Hwang, Chungkuk Yoo, Lama Nachman, and Junehwa Song. 2011. E-Gesture: A Collaborative Architecture for Energy-efficient Gesture Recognition with Hand-worn Sensor and Mobile Devices. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys '11)*. ACM, New York, NY, USA, 260–273. https://doi.org/10.1145/2070942.2070969

[44] Lukasz Piwek and Adam Joinson. 2016. "What do they snapchat about?" Patterns of use in time-limited instant messaging service. *Computers in Human Behavior* 54 (2016), 358 – 367. https://doi.org/10.1016/j.chb.2015.08.026

[45] Julian Seifert, Alexander De Luca, Bettina Conradi, and Heinrich Hussmann. 2010. *TreasurePhone: Context-Sensitive User Data Protection on Mobile Phones*. Springer Berlin Heidelberg, Berlin, Heidelberg, 130–137. https://doi.org/10.1007/978-3-642-12654-3_8

[46] Longfei Shangguan, Zimu Zhou, and Kyle Jamieson. 2017. Enabling Gesture-based Interactions with Objects. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '17)*. ACM, New York, NY, USA, 239–251. https://doi.org/10.1145/3081333.3081364

[47] Michael Sherman, Gradeigh Clark, Yulong Yang, Shridatt Sugrim, Arttu Modig, Janne Lindqvist, Antti Oulasvirta, and Teemu Roos. 2014. User-generated Free-form Gestures for Authentication: Security and Memorability. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '14)*. ACM, New York, NY, USA, 176–189. https://doi.org/10.1145/2594368.2594375

[48] H. Shimodaira, T. Sudo, M. Nakai, and S. Sagayama. 2003. On-line overlaid-handwriting recognition based on substroke HMMs. In *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*. 1043–1047. https://doi.org/10.1109/ICDAR.2003.1227816

[49] Anselm Strauss and Juliet Corbin. 1994. Grounded theory methodology. *Handbook of qualitative research* (1994), 273–285.

[50] Emanuel von Zezschwitz, Sigrid Ebbinghaus, Heinrich Hussmann, and Alexander De Luca. 2017. You Can'T Watch This!: Privacy-Respectful Photo Browsing on Smartphones. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 4320–4324. https://doi.org/10.1145/2858036.2858120

[51] Yang Li. 2010. Protractor: A Fast and Accurate Gesture Recognizer. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA, 2169–2172. https://doi.org/10.1145/1753326.1753654

[52] Wei Wang, Alex X. Liu, and Ke Sun. 2016. Device-free Gesture Tracking Using Acoustic Signals. In *Proceedings of the 22Nd Annual International Conference on Mobile Computing and Networking (MobiCom '16)*. ACM, New York, NY, USA, 82–94. https://doi.org/10.1145/2973750.2973764

[53] Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. 2007. Gestures Without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST '07)*. ACM, New York, NY, USA, 159–168. https://doi.org/10.1145/1294211.1294238

[54] Yina Ye and Petteri Nurmi. 2015. Gestimator: Shape and Stroke Similarity Based Gesture Recognition. In *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction (ICMI '15)*. ACM, New York, NY, USA, 219–226. https://doi.org/10.1145/2818346.2820734

[55] Shumin Zhai and Per-Ola Kristensson. 2003. Shorthand Writing on Stylus Keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '03)*. ACM, New York, NY, USA, 97–104. https://doi.org/10.1145/642611.642630

[56] Y. Zou, Y. Liu, Y. Liu, and K. Wang. 2011. Overlapped Handwriting Input on Mobile Phones. In *2011 International Conference on Document Analysis and Recognition*. 369–373. https://doi.org/10.1109/ICDAR.2011.82

Received August 2017; revised February 2018; accepted April 2018