# Secure Fingertip Mouse for Mobile Devices

Zhen Ling*, Junzhou Luo*, Qi Chen*, Qinggang Yue†, Ming Yang*, Wei Yu‡ and Xinwen Fu†
*Southeast University, Email: {zhenling, jluo, qichen, yangming2002}@seu.edu.cn
‡Towson University, Email: wyu@towson.edu
†University of Massachusetts Lowell, Email: {qye, xinwenfu}@cs.uml.edu

***Abstract*—Various attacks may disclose sensitive information such as passwords of mobile devices. Residue-based attacks exploit oily or heat residues on the touch screen, computer vision based attacks analyze the hand movement on a keyboard, and sensor based attacks measure a device's motion difference via motion sensors as different keys are tapped. A randomized soft keyboard may defeat these attacks. However, a randomized key layout is counter-intuitive and users may be reluctant to adopt it. In this paper, we introduce a novel and intuitive input system, secure finger mouse, which uses a mobile device's camera sensing the fingertip movement, moves an on-screen cursor and performs clicks by sensing click gestures. We design a randomized mouse acceleration algorithm so that the adversary cannot infer keys clicked on the soft keyboard by observing the finger movement. The secure finger mouse can defeat attacks including residue, computer vision and motion based attacks too. We perform both theoretical analysis and real-world experiments to demonstrate the security and usability of the secure fingertip mouse.**

## I. Introduction

Touch-enabled mobile devices have become a burgeoning attack target. Many attacks target sensitive information such as passwords entered on mobile devices by exploiting the soft keyboard. In residue-based attacks [1]–[4], oily or heat residues left on the touch screen indicate which keys are tapped. By measuring the heat residue left on the touched positions, even the order of tapped keys may be determined. In computer vision-based attacks [5]–[13], the interaction between the hand and the keyboard is exploited. For example, the hand movement and finger position indicates which keys are being touched [12], [13]. In sensor-based attacks [14]–[17], the malware senses a device's motion difference via its accelerometer (acceleration) and gyroscope (orientation) when different keys are touched and the device moves slightly.

Intuitively, these attacks are feasible because of the static layout of the soft keyboard of a mobile device. A straightforward countermeasure is to use a randomized keyboard. Such randomized keyboards have been developed for Android and iOS platforms [12] and [18]. However, those soft keyboards are not adopted broadly. One reason is that since a randomized keyboard is not intuitive, it can be hard to find keys on a randomized layout and the usability is limited.

In this paper, we introduce a novel and intuitive input

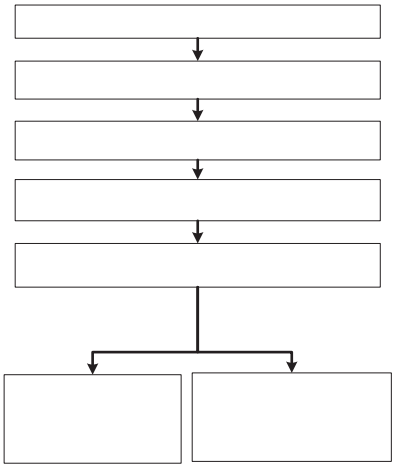


Fig. 1. Workflow of Secure Finger Mouse

and infer the tapped password from the user. For example, TouchLogger [14] is an Android malware that uses the device orientation data to infer keystrokes. Owusu *et a* showed [15] that a malware could use accelerometer data to infer the entered keys on a virtual keyboard. TapLogger [16] used motion sensors to infer a user's tap inputs on a smart mobile. Residue-based attacks exploit oily or heat residues on the touch screen while computer vision based attacks analyze the interaction between the hand and touch screen.

There are existing research efforts on improving the security of authentication on mobile devices [22]–[30]. In the most related work by De Luca *et a* [28], [29], a special touchable device on the back of a mobile device is used to perform pointing and dragging operations for the purpose of authentication.

There are also existing works exploring the back camera of mobile devices for human computer interaction. In [31], a finger is used to cover or uncover the camera lens. The change of brightness is sensed for the interaction with mobile devices. Oh and Hong [32] proposed a finger gesture based mobile user interface. To the best of our knowledge, there is no comparable work to the secure finger mouse in this paper.

## III. Secure Finger Mouse

In this section, we first define the threat model and present the basic idea of the secure finger mouse. We then elaborate the detailed design of our proposed system.

### A Threat Mode

In this paper, we use the following threat model to demonstrate the security of the secure finger mouse while our technique can defeat many other attacks. A touch-enabled mobile device is used in a public environment. An adversary records videos of a victim performing touch input. The victim is cautious about the surroundings and does not input sensitive information when the adversary is too close. Therefore, the adversary cannot directly see the input on the screen in a recorded video. It is assumed that the adversary can obtain the accurate information of the finger movement via various computer vision techniques.

### B Basic Idea

Figure 1 illustrates the workflow of the secure finger mouse. To input a password, a user taps a password input box on the touch screen. After a keyboard pops up, the user puts her index finger beneath the device. When the finger moves, the on-screen cursor moves. When the cursor moves onto a key, the user performs a click gesture in order to enter the key. Therefore, the interaction between a user and her mobile device occurs in two spaces: *contro space* where a user moves her fingertip in the physical space; *disp ay space* where the cursor movement is displayed on the touch screen. Figure 2 illustrates the use of the secure finger mouse. Please note that the secure finger mouse can be used for entering any information while we use password inputting as the example.

The secure finger mouse works in five steps:

**Step 1. Taking Video:** When the user touches a password input box, a keyboard pops up and the camera is activated to take the video and capture the back-of-device interaction between the finger and the mobile. We can display the video on the screen. The display is called a video viewer.

**Step 2. Preprocessing:** We preprocess the video with skin segmentation techniques to remove the background and keep the region with the human skin color in each video frame.

**Step 3. Detecting Fingertip:** After preprocessing, a finger detection classifier is employed to identify the finger frame by frame and compute the position of the fingertip.

**Step 4. Locating Fingertip Top Position:** We use the fingertip top as the actual physical "mouse" and its movement is the raw mouse movement. Noise reduction methods are developed to suppress the impact of fingertip shaking.

**Step 5. Identifying Fingertip Actions:** The secure fingertip mouse has two types of events: click and movement. **Step 5.a. Tapping a key:** 2.94557]TJ40.963071(a)-1.6630.6i3262(P)20.7857(o

on

Fig. 2. Using finger mouse

Fig. 3. Original finger image

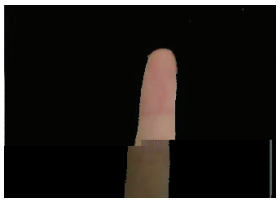


Fig. 4. Preprocessed finger image

Fig. 5. Detected fingertip

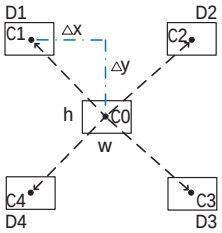



Fig. 6. Region of Interest

fps. However, since we perform extra processing of each video frame with various computer vision algorithms and the mobile device's computing power is limited, the actual FPS for the secure finger mouse decreases.

### *D Step Preprocessing*

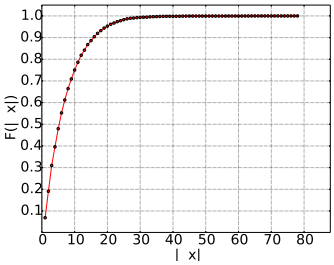
Since we are only interested in the fingertip area, we apply skin segmentation techniques [33] to subtract background and identify the human skin region in each frame in order to improve the finger detection accuracy and processing speed in later steps. The objective of skin segmentation is to determine whether a pixel in a color image has a skin color or non-skin color. A skin color distribution model [34] is a generic and efficient skin segmentation method. Extensive research has been performed to find the fine bounds of skin color in different color spaces, including RGB, normalized rg, HSV and YCbCr [34]–[36]. In this study, we adopt the popular RGB space. A widely used RGB skin color space model [36] is defined as follows,

$$\begin{aligned} R > 95 \; and \; G > 40 \; and \; B > 20 \; and, \\ max\{R,G,B\} - min\{R,G,B\} > 15 \; and, \\ |R-G| > 15 \; and \; R > G \; and \; R > B, \end{aligned} \tag{1}$$

where $R$, $G$, and $B$ are the red, green and blue values in the range of $[0, 255]$ respectively. Due to the lighting, the RGB-based skin segmentation may not be always perfect. Consequently, we apply the two computer vision operations, erosion and dilation, to the segmented image to further remove the noise. Figure 3 illustrates an original image obtained via a phone camera while Figure 4 shows the fingertip after preprocessing.

### *E Step 3 Detecting Fingertip*

In our system, Viola and Jones' cascade-like Adaboost classifiers [37], [38] are adopted for its high accuracy and low computational complexity for real-time fingertip detection. The cascade classifier is derived in the following way. We first collect sufficient gray-scale training images of fingers. 50 volunteers participate in the experiments in diverse backgrounds and use the back camera of a Samsung Galaxy Note 3 to



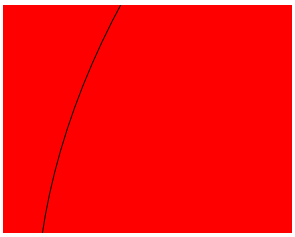



Fig. 7. Empirical CDF of $|\Delta x|$

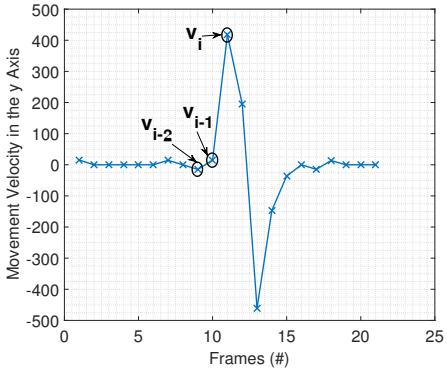



Fig. 15. Velocity along y axis

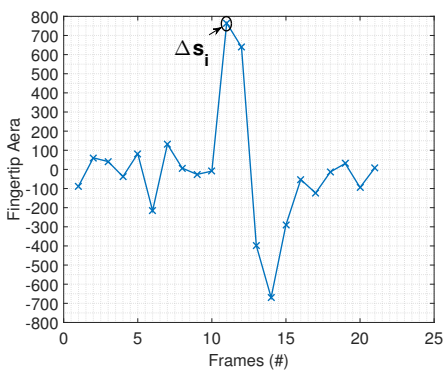



Fig. 16. Verifying the start of a user click

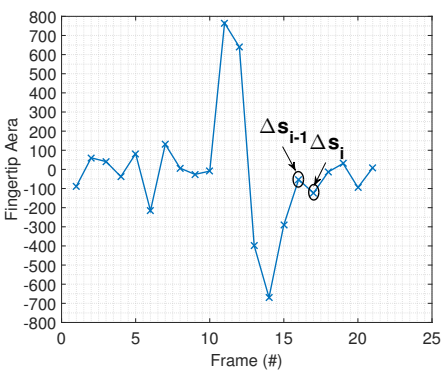



Fig. 17. Detecting the end of a user click

towards the camera or not. Denote the area of the fingertip as $s_i$ in the $i^{th}$ frame and the fingertip area change as $\Delta s_i$, where $\Delta s_i = s_i - s_{i-1}$. Figure 16 illustrates the change of the fingertip area. It can be observed that $\Delta s_i$ rises dramatically. A threshold $T_s$ is used to confirm the click gesture,

$$\Delta s_i \leqslant T_s. \tag{5}$$

With the prediction and confirmation, we can accurately detect a user's click gesture and stop the cursor movement when a user clicks a key.

Recall that we buffer 3 frames in order to determine the start of a click gesture. This delays the response to the fingertip movement. The frame rate should be large enough to reduce this delay. For example, if the frame rate FPS is 20, the latency is $2 * \frac{1}{20} = 100ms$, which does not affect the performance of our system very much. FPS will also increase with the increasing computing power of mobile devices we see nowadays.

To determine the end of a click gesture, we again use the change of the fingertip area. After completing the click gesture, the user moves her fingertip backward to the original position. The fingertip area in the video decreases. In practice, a user may not move her fingertip to exactly the same position and the cascade classifier may also introduce errors. We use another threshold $T_s'$ to determine whether a user stops or not. If the fingertip area change is smaller than $T_s'$, the user stops and finishes the click gesture. Figure 17 shows the change of the fingertip area corresponding to a click gesture. It can be observed that $\Delta s_{i-1}$ is around 0 in the boundary $T_s'$ in the $i-1^{th}$ frame. To confirm the end of the click, we use two continuous frames to measure the change of the fingertip area, that is,

$$|\Delta s_{i-1}| \leqslant T_s' \quad \& \quad |\Delta s_i| \leqslant T_s'. \tag{6}$$

Once Formula (6) is satisfied, we know that the $i-1^{th}$ frame is the end of a click.

When the click gesture is detected, we can determine the intentional key is the one over which the cursor hovers. To generate the key, we use the Android input method service for the password input box and send the key value to the input box. We implement an input method by extending the Android input method service so that the user can use either a 12-key numeric keypad or a full size keyboard.

*) Step b: Perfor ing Cursor Acce eration:* Traditional mouse acceleration algorithms translate the mouse raw movement data to the on-screen cursor movement with a fixed static algorithm. Given the raw mouse movement data, the mapping from the control space to the display space is fixed. If we apply such a static acceleration algorithm to the fingertip mouse, an adversary may record the video of the finger movement and reconstruct the on-screen cursor trajectory to infer the entered keys.

The basic idea of securing the fingertip mouse is to use acceleration algorithms with random parameters. We add randomness into a classic mouse acceleration algorithm shown in Equation (7), i.e., a two-level transfer function, and use a pair of random variables to transfer a raw two-dimension movement in the control space to the movement in the display space. This static two-level transfer function is currently used as a "lightweight" pointer acceleration technique [40] in Xorg, the open-source reference implementation of the X window system. There are two key variables in the transfer function: acceleration $g$ and threshold $T$. The acceleration factor defines a series of gains from the control space to the display space (CD), while the threshold defines the minimum distance required to change the gain (default value is 1) to a new one. Denote the movement in the control space and display space as $C = (\Delta x, \Delta y)$ and $D = (\Delta x', \Delta y')$ respectively. The two-level transfer function can be defined by

$$D = f(g, T) = \begin{cases} g \times C & , \quad |\Delta x| + |\Delta y| \geqslant T \\ C & , \quad |\Delta x| + |\Delta y| < T \end{cases} \tag{7}$$

Algorithm 1 introduces the secure lightweight pointer acceleration algorithm. As long as the movement exceeds the threshold, a random CD gain will be used to accelerate the movement. The integer part of the accelerated movement advances the cursor while the remainders are accumulated in

and

a sequence of Eccleratiou.6y

**Algorithm 1** Secure Lightweight Acceleration Algorithm

**Require:**

(a) $\Delta x, \Delta y$, finger movement in the control space;
(b) $\mathcal{T}$, a set of thresholds;
(c) $T$, a threshold selected from $\mathcal{T}$;
(d) $R_x, R_y$, remainders of the cursor movement;
(e) $X, Y$, cursor position in the display space;
(f) $\mathcal{G}$, a set of CD gains.
(g) $g$, a CD gain selected from $\mathcal{G}$.

1: Randomly select a $T$ from $\mathcal{T}$
2: **if** |

the attacker knows the trajectory, but does not know the starting point of the trajectory. That is, she does not know the starting key of the password. This is actually a good attack strategy. Even if we assume the attacker can record a video of finger movement from the time the user powers up the device, the attacker may have to resort to this attack given that the large sequence of $g$ and $T$ distorts the trajectory too much. Therefore, after deriving the trajectory generated from a chosen sequence of $g$ and $T$, the attacker tries to fit this cursor trajectory to the keyboard, moving the trajectory from top left to bottom right of the keyboard. If the trajectory lands on valid keys, this sequence of keys becomes a password candidate. Therefore, one cursor trajectory may generate a number of password candidates.

We use two metrics to evaluate this attack: *hit rate* and *nu ber of password candidates*. Recall one sequence of $g$ and $T$ creates one cursor trajectory. The *hit rate* is the number of trajectories generating the correct password divided by the total number of trajectories. However, even if there is a hit with a trajectory, the attack is still not feasible when the *nu ber of candidate passwords* generated by that trajectory is too large.

## V. Evaluation

We have implemented the secure fingertip mouse as a third party keyboard app for the Android platform and conducted extensive real-world experiments to demonstrate the usability and security of our developed secure fingertip mouse system. In this section, we first present the experiment setup and then show the evaluation results.

### A Experi ent Setup

We conduct experiments on a Samsung Galaxy Note 3. It has a resolution of $1080 \times 1920$ pixels and the screen size is $5.7$ inches. We implement two services, input method service and cursor display service, and runs two threads, fingertip detection thread and secure mouse acceleration thread. *The input ethod service* provides a new input method, which has four keyboards including a numeric keypad, QWERTY keyboard, and two symbol keyboards. A user can install our third party keyboard app to use this novel input method. When an input box is touched, our keyboard pops up. *The ngertip detection thread* takes videos using the device's back camera, preprocesses each frame, detects the fingertip using our trained cascade classifier and then calculates the raw fingertip movement. The click gesture detection method is implemented in this thread. If a click gesture is detected, the clicked key will be identified and the thread will send the clicked key event to the input method service to enter this key. Otherwise, it will send the raw movement data to the secure mouse acceleration thread. *The secure ouse acce eration thread* chooses random acceleration factors to accelerate the raw mouse movement and sends the accelerated movement to the cursor display service. *The cursor disp ay service* then moves the on-screen cursor. The code uses the Jave Native Interface and the computer vision library OpenCV [42].

### B sabi ity Eva uation

The moving time of the cursor between different keys is the essential metric to evaluate the usability of the secure fingertip mouse. The threshold and acceleration affect the moving time. We need to carefully determine the ranges for the threshold and acceleration to achieve both good usability and security. We recruit 15 unpaid volunteers, $11$ males and $4$ females, aged $24$ years on average (standard deviation = $2.15$ years) to perform a classic within-subjects experiment [43] and measure the cursor moving time between two objects. As a common practice, the volunteers are asked to move the cursor forward and backward between keys "4" and "6" using a numeric keypad for 5 times and also click these two keys.

Figure 18 shows the average cursor moving time versus different acceleration and threshold values. Notice that our secure finger mouse achieves a frame rate of $15$. It can be observed that less acceleration will increase the moving time. However, too much acceleration also increases the moving time. The reason is with large acceleration, a user indeed can move the cursor fast from one key to another, but it also becomes harder for the user to accurately control the cursor to stop on the right key. The chance that the user misses the intended keys increases so that more time is needed to click the right keys. It can also be observed that the threshold can affect the moving time as well. Recall that the threshold controls what ($\Delta x$, $\Delta y$) will be accelerated. A lower threshold will accelerate more finger movements. But it can be hard for the user to control the cursor and click the right keys. However, a large threshold does the opposite and the cursor movement will be too slow. According to the results from Figure 18, the range of threshold is chosen as $[2, 8]$, while the range of acceleration is selected as $[8, 13]$. When the threshold and acceleration is $3$ and $10$, respectively, the performance of cursor moving time is the best. Beyond the chosen ranges, the moving time increases dramatically.

Figure 19 compares moving time using a numeric keypad and a QWERTY keyboard with the threshold $3$. To measure the moving time using a QWERTY keyboard, we ask the volunteers to move the cursor between key "S" and key "K" for 5 times and click these two keys, and then calculate the average cursor moving time. We can see that the moving time using a numeric key is slightly better than the one using a QWERTY keyboard. Since the size of the target key is much smaller on a QWERTY keyboard, the index of difficulty (ID) of QWERTY keyboard is larger than a numeric keypad's ID. With smaller keys on the QWERTY keyboard, the users may undershoot or overshoot the intended key while moving the cursor. It takes time for the user to correct the cursor's position. The results match the theoretical analysis in Equation (11). It can also be observed that the moving time slightly increases after using random thresholds and accelerations on two different keyboards.

Figure 20 compares the input time between touch-inputting on a numeric keypad and using the secure finger mouse. The 15 volunteers are asked to tap a random 4-digits pin. It can be

93K-9 and by Collaborative Innovation Center of Novel Software Technology and Industrialization. Any opinions, findings, conclusions, and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

## REFERENCES


[1] A. J. Aviv, K. Gibson, E. Mossop, M. Blaze, and J. M. Smith, "Smudge attacks on smartphone touch screens," in *Proceedings of orkshop on Offensive Techno ogy OOT*, 2010.

[2] M. Zalewski, "Cracking safes with thermal imaging," http://lcamtuf.coredump.cx/tsafe/, 2005.

[3] K. Mowery, S. Meiklejohn, and S. Savage, "Heat of the moment: Characterizing the efficacy of thermal camera-based attacks," in *Proceedings of orkshop On Offensive Techno ogies OOT)*, 2011.

[4] Y. Zhang, P. Xia, J. Luo, Z. Ling, B. Liu, and X. Fu, "Fingerprint attack against touch-enabled devices," in *Proceedings of the nd orkshop on Security and Privacy in S artphones and Mobi e Devices SPSM)*, 2012.

[5] M. Backes, M. Duermuth, and D. Unruh, "Compromising reflections - or - how to read lcd monitors around the corner," in *Proceedings of the th IEEE Sy posiu on Security and Privacy S P)*, 2008.

[6] M. Backes, T. Chen, M. D1rmuth, H. P. A. Lensch, and M. Welk, "Tempest in a teapot: Compromising reflections revisited," in *Proceedings of the 3 th IEEE Sy posiu on Security and Privacy S P)*, 2009.

[7] R. Raguram, A. White, D. Goswami, F. Monrose, and J.-M. Frahm, "iSpy: Automatic reconstruction of typed input from compromising reflections," in *Proceedings of Proceedings of the th ACM Conference on Co puter and Co unications Security CCS)*, 2011.

[8] D. Balzarotti, M. Cova, and G. Vigna, "Clearshot: Eavesdropping on keyboard input from video," in *Proceedings of the th IEEE Sy posiu on Security and Privacy S P)*, 2008.

[9] F. Maggi, A. Volpatto, S. Gasparini, G. Boracchi, and S. Zanero, "A fast eavesdropping attack against touchscreens," in *Proceedings of the 7th Internationa Conference Infor ation Assurance and Security IAS)*, 2011.

[10] Y. Xu, J. Heinly, A. M. White, F. Monrose, and J.-M. Frahm, "Seeing double: Reconstructing obscured typed input from repeated compromising reflections," in *Proceedings of the th ACM SIGSAC conference on Co puter and Co unications Security CCS)*, 2013.

[11] Q. Yue, Z. Ling, X. Fu, B. Liu, W. Yu, and W. Zhao, "My google glass sees your passwords!" in *Proceedings of the B ack Hat SA*, 2014.

[12] Q. Yue, Z. Ling, X. Fu, B. Liu, K. Ren, and W. Zhao, "Blind recognition of touched keys on mobile devices," in *Proceedings of the st ACM Conference on Co puter and Co unications Security CCS)*, 2014.

[13] D. Shukla, R. Kumar, A. Serwadda, and V. V. Phoha, "Beware, your hands reveal your secrets!" in *Proceedings of the st ACM Conference on Co puter and Co unications Security CCS)*, 2014.

[14] L. Cai and H. Chen, "TouchLogger: Inferring keystrokes on touch screen from smartphone motion," in *Proceedings of the th SENIX orkshop on Hot Topics in Security HotSec)*, 2011.

[15] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang, "Accessory: Keystroke inference using accelerometers on smartphones," in *Proceedings of The Thirteenth orkshop on Mobi e Co puting Syste s and App ications HotMobi e)*, February 2012.

[16] Z. Xu, K. Bai, and S. Zhu, "Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors," in *Proceedings of The ACM Conference on ire ess Network Security iSec)*, 2012.

[17] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. R. Choudhury, "Tapprints: your finger taps have fingerprints," in *Proceedings of the th internationa conference on Mobi e syste s, app ications, and services MobiSys)*, 2012.

[18] J. Koch, "Codescrambler," http://cydia.saurik.com/package/org.thebigboss.codescrambler/, 2014.

[19] Innovative Devices Inc., "Mycestro, the wearable gesture based mouse," http://www.mycestro.com/, 2015.

[20] Thalmic Labs, "Myo gesture control armband," https://www.thalmic.com/myo/, 2015.

[21] X. Pan, Z. Ling, A. Pingley, W. Yu, K. Ren, N. Zhang, and X. Fu, "Password extraction via reconstructed wireless mouse trajectory," *Accepted IEEE Transactions on Dependab e and Secure Co puting TDSC)*, vol. PP, no. 99, March 2015.

[22] T. Vu, A. Baid, S. Gao, M. Gruteser, R. Howard, J. Lindqvist, P. Spasojevic, and J. Walling, "Distinguishing users with capacitive touch communication," in *Proceedings of ACM Internationa Conference on Mobi e Co puting and Networking MobiCo )*, 2012.

[23] A. De Luca, A. Hang, F. Brudy, C. Lindner, and H. Hussmann, "Touch me once and i know it's you! implicit authentication based on touch screen patterns," in *Proceedings of the 3 th SIGCHI Conference on Hu an Factors in Co puting Syste s CHI)*, 2012.

[24] N. Sae-Bae, K. Ahmed, K. Isbister, and N. Memon, "Biometric-rich gestures: A novel approach to authentication on multi-touch devices," in *Proceedings of the SIGCHI Conference on Hu an Factors in Co puting Syste s CHI)*, 2012.

[25] Q. Yan, J. Han, Y. Li, J. Zhou, and R. H. Deng, "Designing leakage-resilient password entry on touchscreen mobile devices," in *Proceedings of the th ACM Sy posiu on Infor ation, Co puter and Co unications Security AsiaCCS)*, 2013.

[26] M. Shahzad, A. X. Liu, and A. Samuel, "Secure unlocking of mobile touch screen devices by simple gestures – you can see it but you can not do it," in *Proceedings of the th ACM Annua Internationa Conference on Mobi e Co puting and Networking MOBICOM)*, 2013.

[27] L. Li, X. Zhao, and G. Xue, "Unobservable re-authentication for smartphones," in *Proceedings of ISOC Network and Distributed Syste Security Sy posiu NDSS)*, 2013.

[28] A. De Luca, M. Harbach, N. D. H. Nguyen, M. Maurer, E. Rubegni, M. P. Scipioni, and M. Langheinrich, "Back-of-device authentication on smartphones," in *Proceedings of the 3 nd SIGCHI Conference on Hu an Factors in Co puting Syste s CHI)*, 2013.

[29] A. De Luca, M. Harbach, E. von Zezschwitz, M. Maurer, B. Slawik, H. Hussmann, and M. Smith, "Now you see me, now you don't - protecting smartphone authentication from shoulder surfers," in *Proceedings of the 3 nd SIGCHI Conference on Hu an Factors in Co puting Syste s CHI)*, 2014.

[30] Y. Chen, J. Sun, R. Zhang, and Y. Zhang, "Your song your way: Rhythm-based two-factor authentication for multi-touch mobile devices," in *Proceedings of the 34th IEEE Internationa Conference on Co puter Co unications INFOCOM)*, 2015.

[31] X. Xiao, T. Han, and J. Wang, "Lensgesture: augmenting mobile interactions with back-of-device finger gestures," in *Proceedings of the th ACM on Internationa conference on u ti oda interaction ICMI)*, 2013.

[32] B.-H. Oh and K.-S. Hong, "Finger gesture-based three-dimension mobile user interaction using a rear-facing camera," *Internationa Journa of Mu ti edia and biquitous Engineering*, 2013.

[33] S. L. Phung, A. Bouzerdoum, and D. Chai, "Skin segmentation using color pixel classification: analysis and comparison," *IEEE Transactions on Pattern Ana ysis and Machine Inte igence*, vol. 27, no. 1, pp. 148–154, January 2005.

[34] P. Kakumanu, S. Makrogiannis, and N. Bourbakis, "A survey of skin-color modeling and detection methods," *Pattern Recognition*, vol. 40, no. 3, p. 1106C1122, March 2007.

[35] J. Brand and J. S. Mason, "A comparative assessment of three approaches to pixel-level human skin-detection," in *Proceedings of the Internationa Conference on Pattern Recognition*, 2000.

[36] J. Kovac, P. Peer, and F. Solina, "Human skin colour clustering for face detection," in *Proceedings of IEEE Region of Internationa Conference on Co puter as a Too E ROCON)*, 2003.

[37] P. Viola and M. J. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of Internationa Conference on Co puter ision and Pattern Recognition PR)*, 2001.

[38] ——, "Robust real-time face detection," *Internationa Journa of Co puter ision*, vol. 57, pp. 137–154, 2004.

[39] OpenCV, "Otsu's thresholding," http://docs.opencv.org/trunk/d7/d4d/tutorial_py_thresholding.html, 2015.

[40] X. Foundation, "Pointer acceleration," http://www.x.org/wiki/Development/Documentation/PointerAcceleration/, 2013.

[41] P. M. Fitts, "The information capacity of the human motor system in controlling the amplitude of movement," *Journa of Experi enta Psycho ogy*, vol. 47, pp. 381–391, 1954.

[42] "OpenCV," http://opencv.org/, 2015.

[43] G. Casiez, D. Vogel, R. Balakrishnan, and A. Cockburn, "The impact of control-display gain on user performance in pointing tasks," *Hu an Co puter Interaction*, 2008.