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 system, secure fingertip mouse. First, the system is a finger mouse. The (back) camera on a mobile is used to capture the finger movement in the physical space (control space), which is mapped to the cursor movement on the touch screen (display space). Click gestures are used to "click" the keys. Second, it is secure in the sense that an adversary cannot infer the on-screen mouse trajectory by analyzing a recorded video of the hand movement. We add randomness into the mouse acceleration algorithm, i.e., the mapping from the control space to the display space. A video demo of the secure finger mouse on Samsung Galaxy Note 3 is given at https://youtu.be/-akGArD0deE.

The major contribution of this paper is summarized as follows. First, the secure finger mouse is the first of its kind for mobile devices. Our system does not use any accessories [19], [20] and only a mobile device's camera is used to sense finger movement. Second, to implement a smooth mouse and improve the usability, we employ various computer vision techniques to increase the rate of frames per second (FPS). Third, the secure finger mouse can defeat various attacks. For example, residue-based attacks fail since no heat or oily residues are left on the touch screen. The randomized mouse acceleration function uses a sequence of random acceleration factors to disrupt the correlation between the physical fingertip movement and the on-screen cursor movement. The sequence of random acceleration factors works like a "secret key" encrypting the on-screen cursor trajectory. We carefully select the range of acceleration factors to balance the security and usability. The concept of randomized mouse acceleration goes beyond the secure finger mouse and should also be adopted by traditional mouse since modern wireless mice do not encrypt their communication. An adversary may sniff the raw mouse data and reconstruct the on-screen trajectory to derive sensitive information such as passwords [21].

The rest of this paper is organized as follows: We review related work in Section II. We present the secure fingertip mouse, including the threat model, the basic idea, and the detailed design of our system in Section III. In Section IV, we conduct theoretical analysis of the security and usability of our developed system. In Section V, we perform extensive real-world experiments to demonstrate the security and usability of the secure fingertip mouse. We conclude this paper in Section VI.

II. Related Work

Touch-screen enabled mobile devices suffer from various side channel attacks, which may disclose individuals' passwords or pins. Example of these attacks include sensor-based malware attacks [14]–[17], residue-based attacks [1]–[4], and computer vision-based attacks [7]–[13]. In sensor-based malware attacks, the malware could be installed on the victim's device, collecting data from sensors (e.g., accelerometer, etc.)

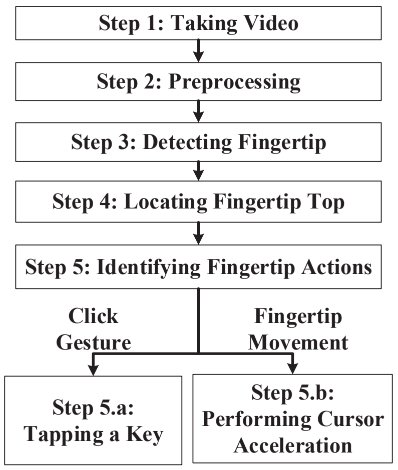
Step 1: Taking Video
Step 2: Preprocessing
Step 3: Detecting Fingertip
Step 4: Locating Fingertip Top
Step 5: Identifying Fingertip Actions
Click Gesture
Fingertip Movement
Step 5.a: Tapping a Key
Step 5.b: Performing Cursor Acceleration

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 al.* 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 al.* [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 Model

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: *control space* where a user moves her fingertip in the physical space; *display 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:** If a click gesture is detected, we check the position of the on-screen cursor and generate the corresponding key. **Step 5.b. Performing Cursor Acceleration:** If a click gesture is not detected, we perform the mouse acceleration and move the cursor on the touch screen. That is, we transfer the raw fingertip movement into the on-screen cursor movement. The mapping from the raw fingertip movement to the on-screen cursor movement is randomized to hide the cursor movement from a potential adversary. We use two random variables to control the mapping and implement the obfuscation. Without knowing the sequence of values of these two random variables, the adversary will not be able to recover the on-screen cursor movement trajectory and know what are clicked on a keyboard.

We elaborate these five steps in detail below.

C. Step 1. Taking Video

We use the back camera of the device to take videos of finger movement. The process of taking a video is a process of sampling the continuous finger motion in the physical control space. The sampling rate is the frames per second (FPS). The sampling rate has to be high enough to satisfy the Nyquist sampling theory to capture the details of the finger movement. Most modern mobile device cameras can record a video at 30

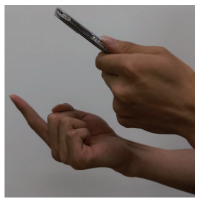
Fig. 2. Using finger mouse

Fig. 3. Original finger image

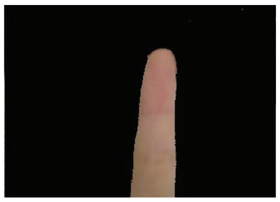
Fig. 4. Preprocessed finger image

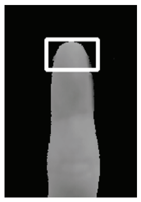
Fig. 5. Detected fingertip

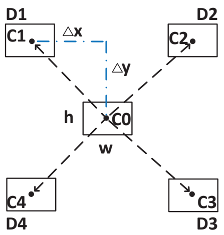

Fig. 6. Region of Interest

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 2. 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 record their finger actions as shown in Figure 2. We design and implement a tool to segment the skin area in an image and then manually use a rectangle to mark the fingertip area. We collect 1600 samples of fingertip images, denoted as positive samples. We also collect 3500 negative samples, in which fingertips are not present. The parameters of the cascade classifier are trained based on these positive and negative image samples. To achieve fast finger detection, the Local Binary Patterns (LBP) feature is employed during the training process. We also test these detectors with different number of stages, and find that the detector with 13 stages achieves the best speed and accuracy. Figure 5 shows the fingertip detected by the cascade classifier.

We adopt the following two strategies to speed up the fingertip detection in order to improve FPS: (i) We reduce the resolution of each video frame to 320 $\times$ 240; (ii) We operate on the region of interest (ROI), i.e., the region of the fingertip, and feed the ROI to the fingertip detector. For the first frame, the ROI is set as the whole image. The detector finds the fingertip area in a bounding box such as the bounding box with center C_0, width w and height h in Figure 6. In two consecutive frames, the finger movement is limited given a specific FPS. Denote the maximum value of the movement along the x and y axes as ΔX and ΔY respectively. Then, the ROI in the subsequent frame can be estimated by the rectangle $D_1D_2D_3D_4$, and its size is $(2*\Delta X+w)$ by $(2*\Delta Y+h)$. The new ROI is fed into the cascade classifier and the processing speed improves because of the small ROI.

The size of the ROI is critical for a decent FPS. We use experiments to derive the range of a single fingertip movement (i.e., Δx and Δy), while w and h is generated by the cascade classifier. Figures 7 and 8 show the empirical cumulative distribution function (ECDF) of fingertip movement. The preferred ΔX and ΔY are the values where the corresponding ECDFs reach 100%. The dark area with the finger in Figure 9 demonstrates the ROI.

F. Step 4. Locating Fingertip Top

The cascade classifier produces a gray image of the fingertip. Since we use the fingertip top as the mouse to control the on-screen cursor movement, we need to accurately locate the fingertip top as illustrated in Figure 10. From Figure 10, we can see that the fingertip is bright compared with the background. We use Otsu's method [39] to conduct the clustering-based thresholding and obtain a binary image, as shown in Figure 11. The fingertip contour can then be derived by looking for contours in the binary image. We fit a line over the central points of each horizontal line of the contour. The intersection between this line and the top of the contour is the fingertip top as shown in Figure 12.

To control the cursor, we need to translate the motion of the fingertip top into the motion of the cursor. Denote $F = \{f_0, f_1, \ldots, f_i\}$ as a series of sequential video frames, where f_i is the latest frame. Denote the coordinate of the fingertip

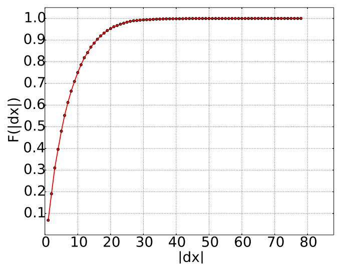

Fig. 7. Empirical CDF of $|\Delta x|$

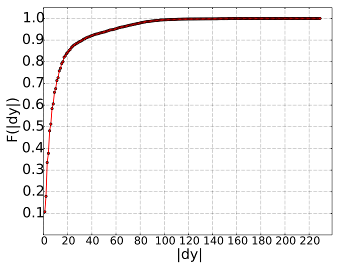

Fig. 8. Empirical CDF of $|\Delta y|$

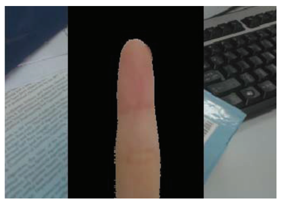

Fig. 9. Finger in ROI

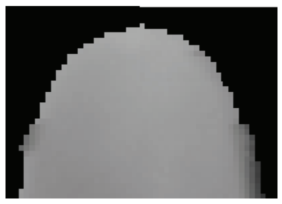

Fig. 10. Gray image

Fig. 11. Binary image

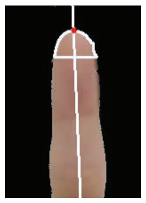

Fig. 12. Located fingertip top

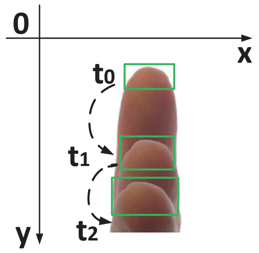

Fig. 13. Clicking process taken from back camera

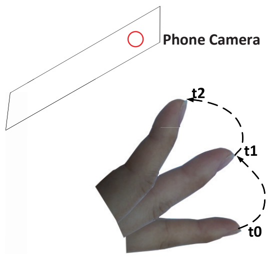

Fig. 14. Clicking process

top in the i^{th} frame as (x_i, y_i). The raw movement along the x and y axes is denoted as $(\Delta x_i, \Delta y_i)$, where $\Delta x_i = x_i - x_{i-1}$ and $\Delta y_i = y_i - y_{i-1}$. $(\Delta x_i, \Delta y_i)$ will be used to control the cursor motion.

In practice, the coordinate of the fingertip top may not be static even if the user tries to hold it statically in front of the camera. There are two error sources. First, the finger may shake slightly. Second, computing $(x,\ y)$ introduces errors. We have observed that the shaking causes continuous fingertip movements in oppositive directions within a tiny area. Therefore, we can identify the shaking with Equation (2).

$$\begin{aligned}
&0 \leqslant \Delta x_i, \Delta y_i \leqslant T_e \ and \ -T_e \leqslant \Delta x_{i+1}, \Delta y_{i+1} \leqslant 0\\
or &\ -T_e \leqslant \Delta x_i, \Delta y_i \leqslant 0 \ and \ 0 \leqslant \Delta x_{i+1}, \Delta y_{i+1} \leqslant T_e\\
or &\ -T_e \leqslant \Delta x_i, \Delta y_{i+1} \leqslant 0 \ and \ 0 \leqslant \Delta x_{i+1}, \Delta y_i \leqslant T_e\\
or &\ 0 \leqslant \Delta x_i, \Delta y_{i+1} \leqslant T_e \ and \ -T_e \leqslant \Delta x_{i+1}, \Delta y_i \leqslant 0.
\end{aligned} \tag{2}$$

If the shaking is detected, the fingertip top's coordinate (x,y) will not be updated to achieve a stable and accurate cursor.

G. Step 5. Identifying Fingertip Actions

We design two motion events for the secure fingertip mouse: movement and click. The challenge is to differentiate these two events. The click is indicated by the click gesture, which is defined as bending the finger toward the camera quickly and then returning to its original position. We have to perform an accurate detection of the click gesture from the fingertip movement. If a click gesture misses the detection and is misidentified as a movement, the cursor will just move fast on the screen and no key will be entered.

1) Step 5.a: Tapping a key: Figures 13 and 14 illustrate the click gesture from different angles. Figure 13 shows that when a click occurs, the fingertip moves much more along the y axis. It can also be observed that the fingertip accelerates while a click gesture is being performed. A third observation is that the area of the fingertip increases while the fingertip moves upward towards the camera.

Based on these three observations, we use the velocity change to predict the start of the click gesture and use the change of the fingertip area to confirm whether it is a click gesture or not. In the prediction phase, to identify the start of a click, we use a queue to buffer the fingertip movement data and timestamps (i.e., $(\Delta x_i,\ \Delta y_i,\ \Delta t_i)$, where $\Delta t_i = t_i - t_{i-1}$) and derive the average velocity of the movement. To identify the start of the click fast, we need to reduce the queue size. We conduct extensive tests and our data shows that the click gesture can be detected if the queue size is 3. Denote the movement data in the queue as $\{(\Delta x_{i-2},\ \Delta y_{i-2},\ \Delta t_{i-2}),\ (\Delta x_{i-1},\ \Delta y_{i-1},\ \Delta t_{i-1}),\ (\Delta x_i,\ \Delta y_i,\ \Delta t_i)\}$. The velocity along the y axis can be derived by

$$v_i = \frac{\Delta y_i}{\Delta t_i}. \tag{3}$$

Denote the sequence of velocity as $\{v_{i-2}, v_{i-1}, v_i\}$ and the average velocity is $a_{i-2} = \frac{v_{i-2}+v_{i-1}+v_i}{3}$. Figure 15 illustrates the velocity along the y axis in one click. It can be observed that the velocity significantly increases along the y axis. The frame where a user starts the click is the $(i-2)^{th}$ frame in Figure 15. We use a threshold T_A to determine whether the user performs the click gesture or not as follows:

$$\begin{cases} \text{Click} &, \quad a_{i-2} \geqslant T_A \\ \text{Non-click} &, \quad a_{i-2} < T_A. \end{cases} \tag{4}$$

Once the click gesture is detected, we need to confirm the click gesture in order to reduce the false positive rate of detecting click gestures. A user may quickly move her fingertip and this may incur a false positive click gesture. Since the fingertip area increases during a click in the video, we use this feature to determine whether a user's fingertip moves

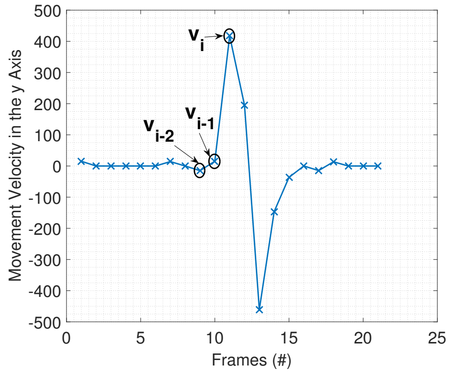

Fig. 15. Velocity along y axis

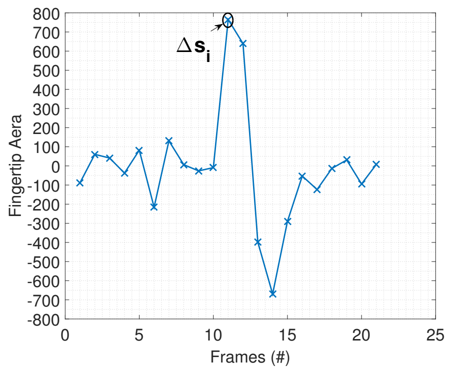

Fig. 16. Verifying the start of a user click

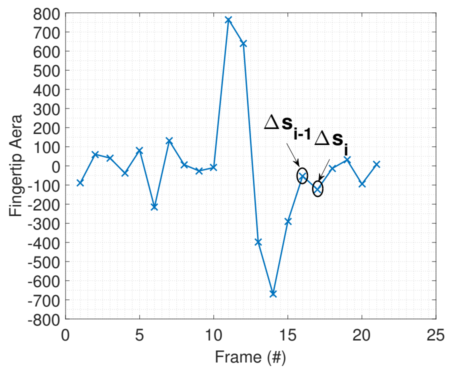

Fig. 17. Detecting the end of a user click

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 Δs_i, where $\Delta s_i = s_i - s_{i-1}$. Figure 16 illustrates the change of the fingertip area. It can be observed that Δs_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 Δs_{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.

2) Step 5.b: Performing Cursor Acceleration: 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 later calculation. Because of the performance requirements, the CD gain g and threshold T are constrained and their ranges are $\mathcal{G}$ and $\mathcal{T}$ respectively.

According to Algorithm 1, inputting a password involves a sequence of acceleration. For each movement (Δx, Δy), a random g and T are selected. Assume there are k movements, (C_1, $\cdots$, C_k) are the movements in the control space and (D_1, $\cdots$, D_k) are the movements in the display space. The

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 $|\Delta x| + |\Delta y| \geqslant T$ then
3:     Randomly select a $g$ from $\mathcal{G}$
4:     $X = g \times \Delta x + R_x$, $Y = g \times \Delta y + R_y$
5:     $R_x = X - \lfloor X \rfloor$, $R_y = Y - \lfloor Y \rfloor$
6:     $X = \lfloor X \rfloor$, $Y = \lfloor Y \rfloor$
7: else
8:     $X = X + \Delta x$, $Y = Y + \Delta y$
9: end if
```

corresponding CD gains are $(g_1, \cdots, g_k)$ and the sequence of thresholds are $(T_1, \cdots, T_k)$. Even if an attacker records the video of the finger movement and derives $(C_1, \cdots, C_k)$, it will be hard for the attacker to derive $(D_1, \cdots, D_k)$ since she does not know $(g_1, \cdots, g_k)$ and $(T_1, \cdots, T_k)$. Therefore, we disrupt the correlation between the control space movement and the display space movement by employing the random acceleration factors.

IV. Analysis

In this section, we first analyze the usability of the secure finger mouse system and then perform the security analysis.

A. Usability Analysis

From a user's perspective, the time for entering a password is an important performance metric for the secure finger mouse system. Fitts' law model [41] has been commonly used to evaluate the efficiency of interaction techniques and input devices. Assume a password has l keys. According to Fitts' law, we can derive the total amount of moving time MT inputting the password. Denote the moving time between the $(i-1)^{th}$ key and the i^{th} key as MT_i, we have

$$MT = \sum_{i=1}^{l} MT_i, \tag{8}$$

$$MT_i = a + b\, ID_i, \tag{9}$$

$$ID_i = \log_2 \frac{D_i}{W_i} + 1, \tag{10}$$

where a and b are two constants. D_i is the distance from the $i-1^{th}$ key to i^{th} key (the target) and W_i is the width of the target. ID_i refers to the index of difficulty moving to the i^{th} key. Therefore,

$$MT = l \times a + b \sum_{i=1}^{l} \log_2 \left(\frac{D_i}{W} + 1 \right). \tag{11}$$

From Equation (11), we can see that the moving time increases if the password length increases and a key is smaller.

We use the true positive (TP) and false positive (FP) to measure the accuracy of our click detection mechanism. The true positive rate is defined as

$$TP = \frac{N_s}{N_c}, \tag{12}$$

where N_s is the number of successful detected click gestures and N_c is the total number of actual clicks. The false positive rate is defined by

$$FP = \frac{N_f}{N_n}, \tag{13}$$

where N_f is the number of non-click actions that are identified as click actions and N_n is the total number of non-click actions.

We measure the input accuracy as follows

$$P = \frac{R_s}{R}, \tag{14}$$

where R is the total number of typed passwords and R_s is the number of successful inputs.

B. Security Analysis

Naturally, the secure finger mouse can defeat various sensor, residue and computer vision based attacks. One potential threat against the secure finger mouse works as follows: the attacker records a video of the finger movement and analyzes the video to infer the password input. We assume that the attacker can obtain the movement in the control space $(C_1, \cdots, C_k)$ from the video[1]. If the attacker knows the acceleration factors $(g_1, \cdots, g_k)$ and $(T_1, \cdots, T_k)$, she may infer the movement $(D_1, \cdots, D_k)$ in the display space. However, the acceleration factors are random and are kept as a secret.

We now discuss the challenge for the attacker who deploys the brute force attack guessing the sequence of g and T, which behave like a key encrypting the cursor movement. Recall g and T are integers and constrained within ranges $\mathcal{G}$ and $\mathcal{T}$. The cardinality $|\mathcal{G}| = m$ and $|\mathcal{T}| = n$. The secure finger mouse changes g and T at each captured frame. Given a frame rate of FPS and the time needed to input a password as t_p, the total number of frames $k = FPS \times t_p$. The number of possible sequences of g and T is $|\mathcal{G}|^k |\mathcal{T}|^k = (mn)^k$. Therefore, we can increase m, n and k to increase the key space.

In practice, an attacker may deploy a trajectory based attack against the secure finger mouse as follows. The attacker records a video of the finger movement while a user inputs a password. We assume this video is synchronized with the frame stream seen by the camera of the secure finger mouse (while the actual unsynchronization between these two video streams gives extra protection for the secure finger mouse). The attacker randomly picks up a sequence of g and T to estimate the on-screen cursor trajectory. She also knows the clicking points on the cursor trajectory. In this attack,

[1]This is actually also hard since the attacker does not know exactly when frames are generated by the victim's camera.

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 *number 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 *number 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. Experiment Setup

We conduct experiments on a Samsung Galaxy Note 3. It has a resolution of 1080×1920 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 method 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 fingertip 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 mouse acceleration thread* chooses random acceleration factors to accelerate the raw mouse movement and sends the accelerated movement to the cursor display service. *The cursor display service* then moves the on-screen cursor. The code uses the Jave Native Interface and the computer vision library OpenCV [42].

B. Usability Evaluation

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 (Δx, Δy) 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

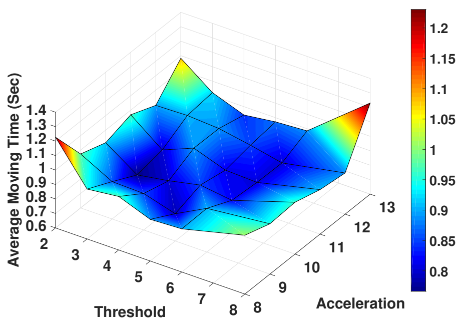

Fig. 18. Moving time versus acceleration and threshold

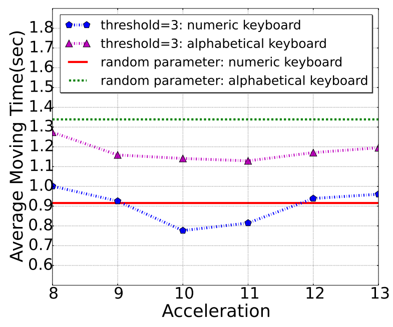

Fig. 19. Comparison of moving time on numeric keypad and alphabetical keyboard

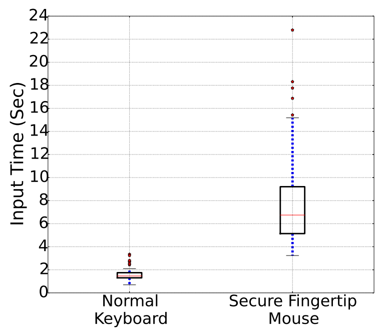

Fig. 20. Comparison of input time

observed that the median inputting time using the secure finger mouse is around 7 seconds while the median touching input time is around 2 seconds. This is reasonable since the secure finger mouse is new to users. Our analysis shows that most of the 7 seconds are spent on moving the cursor. Actually, the secure finger mouse has a similar performance of input time to the randomized keyboard in [12] while the secure finger mouse is much more intuitive as an input method. For future work, we plan to experiment on more powerful mobiles and design new computer vision techniques in order to increase the frame rate and reduce input time.

We now show the true positive and false positive rates for click detection. To collect the ground truth of click and non-click data, we ask the 15 volunteers to perform 20 clicks and 20 random movements. We use a decision tree and derive the thresholds of the fingertip top velocity along the y axis and the change of the fingertip area (i.e., T_A, T_s) as 78.6 and 392.5 respectively. The threshold T_s' in Formula (6), which is used to determine the end of the click, is 210. Under these thresholds, the true positive rate is 90.3%, while the false positive rate is 7.0%. To measure the accuracy of the secure finger mouse, we ask the volunteers to tap 100 random pins. 11 are tapped wrong. The accuracy is 89%.

C. Security Evaluation

Recall it is assumed that the adversary can obtain a victim's finger movement data in the control space and perform the brute force attack by enumerating the two random variables, acceleration g and threshold T, in the transfer function. The attacker knows $g \in \{2,3,4,5,6,8\}$ and $T \in \{8,9,10,11,12,13\}$. That is, the acceleration has 6 possible values and the threshold has 6 possible values. Assume the pin length is 4. In our experiments, the average number of movements (i.e. the number of $(\Delta x, \Delta y)$) for inputting a pin is 95.8. If the brute force attack is performed, the key space is $(6*6)^{95.8} \approx 10^{149}$.

As introduced in Section IV-B, the attacker can deploy the trajectory based attack. The volunteers are asked to input 30 four-letter random passwords using secure finger mouse on a QWERTY keyboard. Therefore, 30 raw password trajectories are generated in the control space. Recall that we assume the adversary knows the raw trajectory in the control space. She chooses a pair of random threshold and acceleration values in order to generate a trajectory in the display space and fit this trajectory onto the keyboard in order to discover password candidates. This attack is performed 1000 times for each password. In our experiments, the average number of password candidates generated by a trajectory is 2988.4. The average hit rate is 9.61% and the average number of password candidates for a hit trajectory is 774.7. Therefore, even if the adversary obtains a hit trajectory, she needs to try hundreds of times in order to derive the correct password. The overall success rate is $9.61\%/774.7 = 1.24 \times 10^{-4}$ and this attack is also not feasible in practice.

VI. Conclusion

In this paper, we introduce a novel secure fingertip mouse that is able to defeat various emerging attacks including sensor, residue attacks and computer vision based attacks against touch-enabled devices. The secure fingertip mouse uses the back camera of a mobile device sensing the fingertip movement and moves a on-screen cursor accordingly. A simple and effective click gesture is used for clicking on the soft keyboard and entering keys. Since an attacker may record videos of finger movement and reconstruct the on-screen cursor trajectory to infer entered keys, we design a randomized mouse acceleration function using a sequence of random acceleration factors to disrupt the correlation between the physical fingertip movement and the on-screen cursor movement. Through a combination of both theoretical analysis and real-world experiments, we demonstrate the feasibility and security of the secure fingertip mouse system.

Acknowledgments

This work was supported in part by China National High Technology Research and Development Program under grants No. 2013AA013503, National Natural Science Foundation of China under grants 61502100, 61532013, 61272054, 61402104, 61572130, and 61320106007, by US NSF grants 1461060 and CNS 1350145, by Jiangsu Provincial Natural Science Foundation of China under Grant BK20150637, by Jiangsu Provincial Key Technology R&D Program under grants BE2014603, by Jiangsu Provincial Key Laboratory of Network and Information Security under grants BM2003201, by Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under grants

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 Workshop on Offensive Technology WOOT*, 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 Workshop On Offensive Technologies (WOOT)*, 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 2nd Workshop on Security and Privacy in Smartphones and Mobile 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 29th IEEE Symposium 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 30th IEEE Symposium 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 18th ACM Conference on Computer and Communications Security (CCS)*, 2011.

[8] D. Balzarotti, M. Cova, and G. Vigna, "Clearshot: Eavesdropping on keyboard input from video," in *Proceedings of the 29th IEEE Symposium 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 International Conference Information 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 20th ACM SIGSAC conference on Computer and Communications 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 Black Hat USA*, 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 21st ACM Conference on Computer and Communications Security (CCS)*, 2014.

[13] D. Shukla, R. Kumar, A. Serwadda, and V. V. Phoha, "Beware, your hands reveal your secrets!" in *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS)*, 2014.

[14] L. Cai and H. Chen, "TouchLogger: Inferring keystrokes on touch screen from smartphone motion," in *Proceedings of the 6th USENIX Workshop 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 Workshop on Mobile Computing Systems and Applications (HotMobile)*, 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 Wireless Network Security (WiSec)*, 2012.

[17] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. R. Choudhury, "Tapprints: your finger taps have fingerprints," in *Proceedings of the 10th international conference on Mobile systems, applications, 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 Dependable and Secure Computing (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 International Conference on Mobile Computing and Networking (MobiCom)*, 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 30th SIGCHI Conference on Human Factors in Computing Systems (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 Human Factors in Computing Systems (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 8th ACM Symposium on Information, Computer and Communications 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 19th ACM Annual International Conference on Mobile Computing and Networking (MOBICOM)*, 2013.

[27] L. Li, X. Zhao, and G. Xue, "Unobservable re-authentication for smartphones," in *Proceedings of ISOC Network and Distributed System Security Symposium (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 31nd SIGCHI Conference on Human Factors in Computing Systems (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 32nd SIGCHI Conference on Human Factors in Computing Systems (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 International Conference on Computer Communications (INFOCOM)*, 2015.

[31] X. Xiao, T. Han, and J. Wang, "Lensgesture: augmenting mobile interactions with back-of-device finger gestures," in *Proceedings of the 15th ACM on International conference on multimodal interaction (ICMI)*, 2013.

[32] B.-H. Oh and K.-S. Hong, "Finger gesture-based three-dimension mobile user interaction using a rear-facing camera," *International Journal of Multimedia and Ubiquitous Engineering*, 2013.

[33] S. L. Phung, A. Bouzerdoum, and D. Chai, "Skin segmentation using color pixel classification: analysis and comparison," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 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 International 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 8 of International Conference on Computer as a Tool (EUROCON)*, 2003.

[37] P. Viola and M. J. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2001.

[38] ——, "Robust real-time face detection," *International Journal of Computer Vision*, 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," *Journal of Experimental Psychology*, 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," *Human Computer Interaction*, 2008.