Blind Recognition of Touched Keys on Mobile Devices

Qinggang Yue
University of Massachusetts
Lowell, USA
qye@cs.uml.edu

Zhen Ling
Southeast University, China
zhenling@seu.edu.cn

Xinwen Fu
University of Massachusetts
Lowell, USA
xinwenfu@cs.uml.edu

Benyuan Liu
University of Massachusetts
Lowell, USA
bliu@cs.uml.edu

Kui Ren
University at Buffalo, USA
kuiren@buffalo.edu

Wei Zhao
University of Macau, China
weizhao@umac.mo

ABSTRACT

In this paper, we introduce a novel computer vision based attack that automatically discloses inputs on a touch-enabled device while the attacker cannot see any text or popup in a video of the victim tapping on the touch screen. We carefully analyze the shadow formation around the fingertip, apply the optical flow, deformable part-based model (DPM), k-means clustering and other computer vision techniques to automatically locate the touched points. Planar homography is then applied to map the estimated touched points to a reference image of software keyboard keys. Recognition of passwords is extremely challenging given that no language model can be applied to correct estimated touched keys. Our threat model is that a webcam, smartphone or Google Glass is used for stealthy attack in scenarios such as conferences and similar gathering places. We address both cases of tapping with one finger and tapping with multiple fingers and two hands. Extensive experiments were performed to demonstrate the impact of this attack. The per-character (or per-digit) success rate is over 97% while the success rate of recognizing 4-character passcodes is more than 90%. Our work is the first to automatically and blindly recognize random passwords (or passcodes) typed on the touch screen of mobile devices with a very high success rate.

Categories and Subject Descriptors

K.4.1 [**COMPUTERS AND SOCIETY**]: Public Policy Issues—*Privacy*

General Terms

Human Factors, Security

Keywords

Computer Vision Attack; Mobile Devices; Privacy Enhancing Keyboard

1. INTRODUCTION

Touch-enabled devices are ubiquitously used in our daily life. However, they are also attracting attention from attackers. In addition to hundreds of thousands of malwares [19], one class of threats against mobile devices are computer vision based attacks. We can classify those attacks into three groups: the first group of attacks directly identify text on screen or its reflections on objects [2, 1]. The second group of attacks detect visible features of the keys such as light diffusion surrounding pressed keys [3] and popups of pressed keys [28, 33]. The third group of attacks are able to blindly recognize the text input on mobile devices while text or popups are not visible to the attacker [43] .

In this paper, we introduce a novel attack blindly recognizing inputs on touch-enabled devices by estimating touched points from a video of a victim tapping on the touch screen, as shown in Figure 1. In the attack, the deformable part-based model (DPM) is used to detect and track the target device and the optical flow algorithm is used to automatically identify touching frames in which a finger touches the screen surface. We use intersections of detected edges of the touch screen to derive the homography matrix mapping the touch screen surface in video frames to a reference image of the software keyboard, as shown in Figure 2. DPM and other computer vision techniques are applied to automatically estimate a tiny touched area. We carefully derive a theory of the shadow formation around the fingertip and use the k-means clustering algorithm to identify touched points in the tiny touched area. Homography can then map these touched points to the software keyboard keys in the reference image. We performed extensive experiments. The victim target devices include the iPad, Nexus 7 and iPhone 5. Both login keyboard and QWERTY keyboard are examined. The cameras include a webcam, a phone camera and Google Glass. The camera is positioned from different distances and angles. We are able to achieve a per-key success rate of over 97% and success rate of more than 90% recognizing 4-digit passcodes in various scenarios.

We also show that DPM can be used to directly estimate the touched point, which can be mapped to the reference image in order to derive the touched key. This method of direct use of DPM for recognizing touched keys is called the baseline method. However, the baseline method achieves a success rate of around 26% since DPM cannot accurately locate touched points.

To the best of our knowledge, we are the first to be able to reliably and blindly recognize passwords (or passcodes) typed on the touch screen of mobile devices of various kinds. Since passwords are random and do not contain meaningful text or pattern, natural language processing techniques cannot be used. This challenges the design of automatic recognition of the password. Our recognition system incorporates recent advancement of object detection techniques and our own analytical model of the touching process, and is able to achieve a very high success rate. We have also extended our work to the scenario of touching with both hands and multiple

CCS'14, November 3–7, 2014, Scottsdale, Arizona, USA.

http://dx.doi.org/10.1145/2660267.2660288.

fingers and are able to recognize the touching finger from 10 fingers and achieve a high success rate of more than 95% recognizing touched keys.

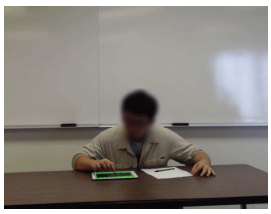


Figure 1: Touching Frame

Figure 2: Soft Keyboard

To defeat many computer vision based attacks including the one in this paper, we designed and implemented a simple context aware randomized software keyboard for Android, denoted as Privacy Enhancing Keyboard (PEK). PEK automatically shows a conventional QWERTY keyboard for the normal text input and pops up a randomized keyboard for inputting sensitive information such as passcodes. The first PEK prototype was demonstrated at an ACM workshop [47] in October, 2012[1]. To the best of our knowledge, PEK is the first generic software keyboard for a mobile platform while a similar app *CodeScrambler* for iOS [23] appeared in August 2013. PEK is a full fledged software keyboard while CodeScrambler is designed only for the unlock screen and does not provide the context-aware functionality. Please refer to the appendix for the implementation and evaluation of PEK.

The rest of the paper is organized as follows. We introduce the homography and DPM in Section 2. Section 3 introduces the attack for the case of tapping with one finger. In Section 4, we discuss how to recognize touched points from touching frames. Experiment design and evaluations are given in Section 5. We extend the attack to tapping with multiple fingers and two hands in Section 6. Section 7 discusses the related work. We conclude the paper in Section 8.

2. BACKGROUND

In this section, we introduce two major computer vision techniques employed in this paper: planar homography and the DPM (Deformable Part-based Model) object detector.

2.1 Planar Homography

Planar homography is a 2D projective transformation that relates two images of the same planar surface [7]. Assume $p = (s, t, 1)$ is any point in an image of a 3D planar surface and $q = (s', t', 1)$ is the corresponding point in another image of the same 3D planar surface. The two images may be taken by the same camera or different cameras. There exists an invertible 3×3 matrix $\mathbf{H}$, denoted as homography matrix,

$$q = \mathbf{H}p. \tag{1}$$

2.2 Deformable Part-based Model (DPM)

DPM [11] is the state-of-art object detector and contains three main components: a mixture of star-structured part based models, the efficient matching process for object detection and the latent SVM (Support Vector Machine) training process. DPM builds multiple star-structured models for the object of interest from different viewpoints. Each star-structured model has a root model that characterizes the object as a whole and several (usually six) part models that characterize each part of the object, their anchor position relative to the root and associated deformation parameters. The models are represented by the Histogram of Oriented Gradients (HOG) [10] feature, which is insensitive to lighting variation.

To detect an object in an image, DPM uses a sliding-window approach and calculates a score $f_\beta(x)$ for each possible object sample x at each location,

$$f_\beta(x) = \max_{z \in Z(x)} \beta \cdot \Phi(x, z), \tag{2}$$

where z is the latent values, β is a vector of model parameters, and $\Phi(x, z)$ is the feature vector of x. A high score indicates the location of the object. Dynamic programming and generalized distance transforms are employed for efficient matching.

During the training, a bounding box is used to specify the object of interest in each image, while its parts are unlabeled. DPM treats these unlabeled parts as latent (hidden) variables. It automatically finds and labels the parts, and employs the latent SVM to train the model. Denote a training data set as $D = (< x_1, y_1 >, \ldots, < x_n, y_n >)$. x_i is the image patch in the corresponding bounding box. $y_i \in \{-1, 1\}$, indicating whether x_i is the object of interest ($y_i = 1$) or not ($y_i = -1$). DPM trains β by minimizing the objective function,

$$L_D(\beta) = \frac{1}{2} \parallel \beta \parallel^2 + C \sum_{i=1}^{n} max(0, 1 - y_i f_\beta(x_i)), \tag{3}$$

where $max(0, 1 - y_i f_\beta(x_i))$ is the standard hinge loss and the constant C controls the relative weight of the regularization term [11]. The purpose of minimizing Formula (3) is to classify an object x correctly and reduce the modulus of β.

3. HOMOGRAPHY BASED ATTACK AGAINST TOUCH SCREEN

In this section, we introduce the basic idea of the attack and each step in detail.

3.1 Basic Idea

Figure 3 shows the flow chart of the automatic and blind recognition of touched keys on mobile devices.

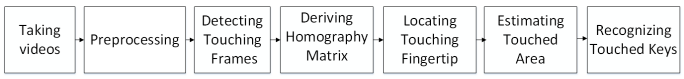


Figure 3: Work flow of Blind Recognition of Touched Keys

Without loss of generality, we often use the four-digit passcode input on iPad as the example. **Step 1** - Take a video of the victim tapping on a device. We do not assume the video records any text or popups while we assume the finger movement and the target device's screen surface are recorded. **Step 2** - Preprocess the video and keep only the touch screen area with moving fingers. We assume that the type of device is known or can be detected so that we also obtain a high resolution image of the corresponding software keyboard on the touch screen surface, denoted as *reference image*, as shown in Figure 2. **Step 3** - Detect the *touching frames*, in which the finger touches the screen surface, as shown in Figure 1. **Step 4** - Identify features of the touch screen surface and derive the planar homography matrix between the touching frames and the reference image. **Step 5** - Employ DPM and various computer vision techniques to obtain a large box bounding the touching fingertip. This is a key step of implementing an automatic touched key recognition. However, extra steps are needed to actually find the touched point that can be mapped to the reference image and recognize the touched key. We denote the direct use of DPM finding the touched point as the baseline method. **Step 6** - Find the fingertip contour

[1]No paper was published on PEK.

in the large bounding box and train a tiny bounding box around the fingertip top as the accurate touched area. **Step 7** - Build a model of the touching process, identify the touched points from the estimated tiny touched area and map them to the reference image via the homography. If the touched points can be correctly located, we can disclose the corresponding touched keys. We introduce the seven steps in detail below.

3.2 Step 1 - Taking Videos

The attacker takes a video of a victim tapping on a device from a distance. Such scenarios include students taking classes, researchers attending conferences, and tourists gathering and resting in a square. With the development of smartphones and webcams, a stealthy attack at such a crowded location is feasible. For example, cameras of iPhone, *Google Glass* and even a smartwatch have decent resolution. *Galaxy S4 Zoom* has a 16-megapixel (MP) rear camera with a 10x zoom lens, weighting only 208g. Amazon sells a webcam-like plugable 2MP USB 2.0 digital microscope with a 10x-50x optical zoom lens [32].

Three factors in taking videos affect the success of the attack: camera angle, distance between the target and the camera and lighting over the target. The success of the attack relies on accurate identification of touched points. The camera angle needs to be adjusted in order to record the finger movement over the touch screen. For example, in a conference room, an attacker in the front can use the front camera of her phone to record a person tapping in the back row. The camera cannot be too far away from the victim. Otherwise, the keys and fingers in the image are too small to be differentiated. Intuitively, a camera with an optical zoom lens can help in such a case. However, the scenes of interest in our context may not allow cameras with big lens. Lighting affects the brightness and contrast of the video and thus the recognition result.

3.3 Step 2 - Preprocessing

Since we are particularly interested in the fingertip area, where the finger touches a key, our first preprocessing step is to apply DPM to detect and locate the touch device in the video. We then crop the video and keep the region of the touch device with moving fingers. Cropping removes much the background and makes later processing simpler.

To apply DPM to the detection of the target device in each video frame, we first need to generate positive data (such as iPad) and negative data (background) to train a target device model. To get the positive data, we take 700 images of the target device such as iPad from different viewpoints, and manually label the device with a bounding box. To get a tight bounding box of the device in an image, we first derive the homography relation between the device image and the reference image in Figure 2, and then map the four corners of the device (iPad in this example) in the reference image to the training image. The up-right bounding rectangle of the four points accurately delimits the device in the training image. To derive the negative data, we employ 900 background images from the SUN database [42] and label objects that have a similar shape to the target device. DPM will also generate the negative data itself using its own data mining methods.

A target device appears different in images from different viewpoints. Thus, we need to train a multi-component model. Figure 4 shows the four component model of iPad. The first row models iPad viewed from the right, and the second row models iPad viewed from the left, and the third and fourth rows model iPad viewed from the right front and left front of iPad. The first column shows the root model (the coarse model characterizing iPad as a whole), the second column shows its parts from different viewpoints, and the third column visualizes the spatial model of the location of each part relative to the whole object. This mixture model effectively characterizes the structure and features of iPad. After training, we apply the learned model to the video frames, and the device is accurately localized as shown in Figure 5.

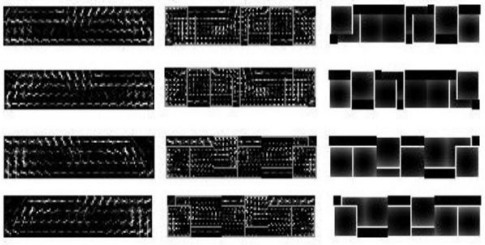


Figure 4: Trained iPad DPM Model

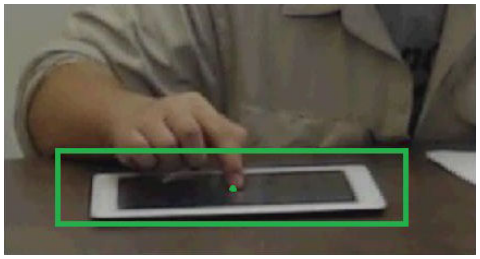


Figure 5: Detected iPad (Magnified)

DPM is a very time-consuming object detector and is not computation-efficient. If the target device is static in the video, we just need to detect the target device in the first frame and crop the same area of the target device in all the video frames. Otherwise, we have to use DPM and track the target device in every frame.

The second preprocessing step is to digitally enhance the image resolution of the target device. We digitally magnify the cropped video frames. For example, we resize each cropped frame to four times its original size.

The third preprocessing step is to obtain the reference image of the software keyboard on the target device. We assume the target device brand is known and the attacker can get a high quality image of the software keyboard on the touch screen. This image is the "reference image", as shown in Figure 2. The image shall show detailed features of the device, particularly the touch screen surface. For example, for iPad, we choose a black wallpaper so that the touch screen has a high contrast with its white frame. It is not difficult to recognize most tablets and smartphones since each brand has salient features. For example, walking past the victim, the attacker can know the device brand. The attacker may also recognize the device brand from the video.

3.4 Step 3 - Detecting Touching Frames

Touching frames are those video frames in which the finger touches the screen surface. To detect touching frames, we need to analyze the finger movement pattern of the touching process. Here we analyze the case of people using one finger to tap on the screen and input the passcode while we extend our work to tapping with multiple fingers and two hands in Section 6.

During the touching process, the fingertip first moves downward towards the touch screen, stops, and then moves upward away from the touch screen. The finger may also move left or right while moving downward or upward. We define the direction of moving toward the device as positive and the opposite direction as negative. In the process of a key being touched, the fingertip velocity is first positive while moving downward, then zero while stopping on the

screen and finally negative while moving upward. This process repeats for each touched key. Therefore, a touching frame is the one where the fingertip velocity is zero. Sometimes the finger moves so fast that there is no frame where the fingertip has a zero velocity. In such a case, the touching frame is the one where the fingertip velocity changes from positive to negative.

The challenge to derive the fingertip velocity is to identify the fingertip. The angle from which we take the video affects the shape of the fingertip in the video. The fingertip shape also changes when the soft fingertip touches the hard touch screen surface. People may also use different areas of the fingertip to tap the screen. We find that when people touch keys with the fingertip, the whole hand most likely keeps the similar gesture and moves in the same direction. Instead of tracking the fingertip to identify a touching frame, we track the hand, which has enough number of feature points for an automatic tracking.

We employ optical flow theory [38] to derive the velocity of feature points on the moving hand. Optical flow computes object motion between two frames. The displacement vector of the points between subsequent frames is called the image velocity or the optical flow at that point. We employ the KLT algorithm [46], which can track sparse points. To make the KLT algorithm effective, we select unique feature points, which are often corners in the image. The Shi-Tomasi corner detector [35] is applied to obtain these points. We track several points in case some points are lost during the tracking. Our experiments show that each touch with the fingertip may produce multiple touching frames. This is reasonable since the fingertip is soft. When a fingertip touches the screen, it deforms and this deforming process takes time. People may also intentionally stop to make sure that a key is touched. During the interaction between fingertip and touch screen, some tracked points may also move upward and create noise for detecting touching frames. We use a simple algorithm to deal with all the noise: if the velocity of most tracked points in a frame moves from positive to negative, that frame is a touching frame. Our experiments show that five features points are reliable for detecting all touching frames.

3.5 Step 4 - Deriving the Homography Matrix

In computer vision, automatically deriving the homography matrix H of a planar surface in two images is a well studied problem [14]. First, a feature detector such as SIFT (Scale-Invariant Feature Transform) [27] or SURF (Speeded Up Robust Features) [4] is used to detect feature points. Matching methods such as FLANN (Fast Library for Approximate Nearest Neighbors) [31] can be used to match feature points in the two images. The pairs of matched points are then used to derive the homography matrix via the algorithm of RANSAC (RANdom SAmple Consensus) [18].

However, those common computer vision algorithms for deriving homography $\mathbf{H}$ are not effective in our context. Because of the angle of taking videos and reflection by the touch screen, there are few good feature points in the video frames for the algorithms above to work effectively. Intuitively, touch screen corners are potential good features, but they are blurry in our context since the video is taken remotely and the resolution is poor. Therefore, SIFT or SURF cannot correctly detect these corners.

We derive the homography matrix $\mathbf{H}$ in Formula (1) as follows. $\mathbf{H}$ has 8 degrees of freedom. Therefore, to derive the homography matrix, we need 4 pairs of matching points of the same plane in the touching frame and reference image. Any three of them should not be collinear [14]. In our case, we try to use the corners of the touch screen as shown in Figure 1 and Figure 2. Because the corners in the image are blurry, to derive the coordinates of these corners, we first detect the four edges of the touch screen. The intersections of these edges are the desired corners. We apply the Canny edge detector [9] to extract the edges and use the Hough line detector [29] to derive candidate lines in the image. We manually choose the lines aligned to the edges. This is the only manual procedure in our entire system of blindly recognizing touched keys. After edges are derived, now we can calculate the intersection points and derive the coordinates of the four corners of interest. With these four pairs of matching points, we can derive the homography matrix with the DLT (Direct Linear Transform) algorithm [14]. If the device does not move during the touching process, this homography matrix can be used for all the video frames. Otherwise, we have to derive H for every touching frame and the reference image.

3.6 Step 5 - Locating the Touching Fingertip

In this step we locate the touching fingertip in the touching frame to identify where the fingertip touches the screen. Then we can map the touched point to the reference image by the homography matrix in order to get the touched key. Again, we turn to the DPM object detector to locate the touching fingertip in touching frames.

The process of employing DPM to locate the touching fingertip is similar to the process of applying DPM to the detection of the target device in a video frame. We first generate positive data (touching fingertip) and negative data (non touching fingertip) to train a model for the "touching" fingertip. To get the positive data, we take videos in various scenarios and obtain the touching frames. For each touching frame, we label the touching fingertip with an appropriate bounding box centered at the center of the touched key. We derive the center of a key in a touching frame in the following way. During the training process, we know the touched keys and can derive their position by mapping the area of a key from the reference image to the touching frame with the planar homography. As we know, DPM needs a bounding box that is large enough to perform well although we want a bounding box as small as possible. We evaluated bounding boxes of different size. The optimal bounding box in our context is the one bounding the fingertip, centered at the touched key and has a size of 40×30 pixels. If different bounding box sizes are used for training images, DPM resizes the bounded area to a uniform size. To get the negative data, we use the bounding box around the non-touching fingertip. DPM also generates negative data itself via data mining and treats the bounding box with less than 50 percentage intersection with the positive data as negative data.

After training, DPM produces a multi-component model for the touching fingertip as visualized in Figure 6. The left column visualizes the root filter of a two-component model: the shape of the touching fingertip and the interaction between the fingertip top and the touch screen. Shadow is formed at the fingertip top during touching. The two components actually model the touching fingertip from different viewpoints respectively. The part models in the middle column characterize the six parts of the touching fingertip. The spatial models in the last column characterize the location of each part relative to the root. We apply these models to every touching frame to detect the touching fingertip. Figure 7 shows the detected result as the green large bounding box and its center C.

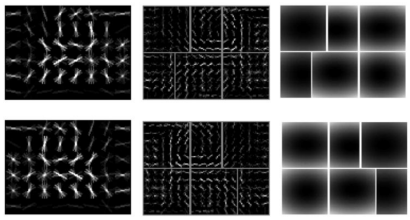


Figure 6: DPM Touching Fingertip Model

Figure 7: Detected Touching Fingertip

Recall that during the training process, the center C of the large bounding box estimates the center of a touched key. Therefore, after DPM is applied to an image, the center of the resultant bounding box is expected to overlap the center of the touched key. Intuitively, we can map the center of the detected bounding box directly to the reference image. The mapped point should fall into the area of the touched key. We denote this method as "baseline method". However, from the experiments, the baseline method does not work well. Evaluation will be provided in Table 5 in Section 5. The poor experiment results demonstrate the limitation of the direct use of machine learning methods to recognize touched keys. The major reason is that DPM is still a very coarse object detector in our context.

3.7 Step 6 - Estimating the Touched Area

Even though the center of the bounding box derived in Step 5 is not exactly the center of the touched key, the touched key should be around the detected fingertip since people tend to touch the center of the key. Our extensive experiments on varieties of subjects have verified this observation. We need to further analyze the image patch in this large bounding box given by DPM in order to derive the accurate touched area, where the fingertip touches the screen.

The complication of lighting and shadowing makes the estimation of accurate touched area a great challenge. We employ two steps to this end. First, within large bounding box, we locate the fingertip and get its contour via k-means clustering of pixels. Second, we derive the fingertip's touching direction and train a tiny bounding box around the top of fingertip as accurate touched area.

Deriving the fingertip contour: We train a *small bounding box* as shown in Figure 8 around C of the large bounding box and use the k-means clustering over this small bounding box to get the fingertip contour. First, we convert the region of this small bounding box into a gray scale image and increase its contrast by normalizing the gray scale image so that its maximum intensity value is 255. The k-means clustering ($K = 2$) is then employed to cluster the pixel values into two categories, dark and bright. This region of interest is then transformed into a binary image. The intuition is that the touching finger is brighter than the area around it. Therefore, we are able to find the contour of the fingertip as the bright area. Figure 9 shows the contour of the fingertip after we process the small bounding box.

Deriving the accurate touched area: Once the fingertip contour is located, we can estimate the top of the fingertip and train a tiny bounding box around the fingertip top as the accurate touched area. To derive the fingertip top, for each horizontal line of pixels in the fingertip contour, we find its central point. We then fit a line over these central points. This line is the central line of the finger in the image, indicating the finger's touching direction and which part of the fingertip is used to touch the screen. The intersection between this line and the fingertip contour produces the top of the fingertip and the center of the touched area. Figure 9 shows the estimated top and bottom of the fingertip and its direction. Figure 10 shows a tiny bounding box we trained around the top of the fingertip.

There are various complications in the two steps above finding the tiny bounding box. For example, when we try to find the contour of the fingertip, the ideal case is: there is only one contour, i.e., the bright fingertip contour, in the small bounding box. However, lighting and shadowing may introduce noise and produce other small contours. We have used erosion and dilation techniques [7] to remove such small contours. Another complication is that the fingertip may have a virtual image on the touch screen, which behaves like a mirror. The virtual image produces a second large contour. Such a contour can be identified by introducing a model of the fingertip's position. For example, the *upper* large contour indicates the actual fingertip. Lighting and shadowing can make the case more complicated. The two large contours corresponding to the fingertip and its virtual image may be connected even if erosion and dilation are applied. In such a case, we locate the convexity defects of the two connected large contours. The large defects indicate the connecting position. We can split the two connected contours at this position. We have applied various computer vision techniques and managed to reduce the impact of complications on automatically recognizing touched keys. However, these complications do affect our recognition results.

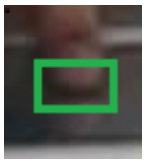


Figure 8: Small Bounding Box

Figure 9: Fingertip Contour and Direction

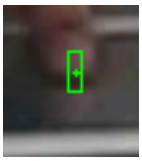


Figure 10: Accurate Touched Area

3.8 Step 7 - Recognizing Touched Keys

Although we have derived the tiny and accurate touching area in Figure 10, such an area is still too large and contains non-touching points. From our analysis in Section 4 and experiments, an actual key area contains only tens of pixels. Our goal of Step 7 is to recognize those actual touched points landed in the key's area. Once the actual touched points are located, we can then map them to the reference image. The corresponding points in the reference image are denoted as mapped points. Such mapped points should land in the corresponding key's area on the software keyboard. Therefore, we can derive the touched keys. This is the basic idea of blindly recognizing the touched keys even if those touched keys are not visible in the video. The key challenge is to accurately locate the touched points in the tiny bounding box. We introduce our model and methodology of addressing this challenge in Section 4.

4. RECOGNIZING TOUCHED KEYS

In this section, we model how people use their finger tapping on the touch screen and the image formation process of a tapping finger. We then propose a clustering-based strategy to identify touched points and map these points in the touching frames to keys in the reference image.

4.1 Formation of Touching Frames

To model how touching frames are formed, we first analyze how people tap on the screen, denoted as *touching gestures*. According to [5, 12, 40], there are two types of touching gestures: vertical touch and oblique touch. In the case of vertical touch, the finger moves downward vertically to the touch screen as shown in Figure 11 (a). People may also choose to touch the screen from an oblique angle as shown in Figure 11 (b), which is the most common touching gesture, particularly for people with long fingernails. The terms of vertical and oblique touch refer to the "steepness" (also called "pitch") difference of the finger [12]. From Figure 11, the finger orientation (i.e. "yaw") relative to the touch screen may also be different [41]. The shape and size of a person's finger and key size also affect the touching gestures and where a key will be touched.

Now we analyze the image formation of a fingertip touching a key. Videos may be taken from different angles. Without loss of generality, we study the case in which the camera faces the touching finger and the front edge of the key is in parallel to the image plane. Figure 12 shows the geometry of the image formation of the touching finger in the 3D world when the fingertip falls inside a key's area. The key's height is w and its length is l. The point F

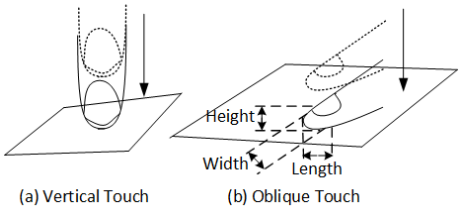


Figure 11: Touching Gestures

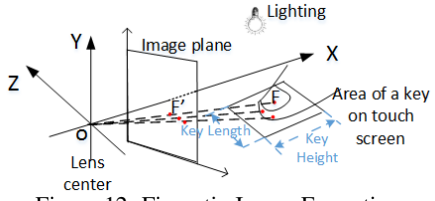


Figure 12: Fingertip Image Formation

on the fingertip will project to the point F' on the image plane. Its brightness in the image will be determined by lighting and the fingertip shape. Because of the lighting difference, points on the side of the finger facing the touch screen are dark in the image. Adjacent to the dark area is the gray area where lighting is weak. There is also the bright area on the finger that is well illuminated.

Figure 13 shows the longitudinal view of a finger touching the surface. We use Figure 13 to discuss our basic principle of inferring a touched key. K_f and K_b are the front and back of the touched key $\mathcal{K}$ respectively. T is the touched point. Apparently T is on the line segment $\overline{K_f K_b}$. T and $\overline{K_f K_b}$ are projected onto the image plane as T' and $\overline{K'_f K'_b}$. If we can identify T' in the image, our problem is solved. However, as we can see from Figure 13, since the human finger has a curved surface, the camera may not be able to see the touched point. $\overline{OT_o}$ is the tangent line to the curved finger surface and it intersects with the touch screen surface at T_o. The camera can see T_o, which is the closest point to the touched point on the touch screen surface. T_o is projected as T'_o on the image plane. If T'_o is on the line segment $\overline{K'_f K'_b}$, then we just need to find T'_o in the image and T'_o can be used to determine the touched key.

We argue that T'_o generally lands in the area of a key. Table 1 shows the key size of the *unlock screen software keypad* for iPad, iPhone and Nexus 7 tablet. Figure 12 gives the definition of key height and length. Table 2 gives the average size of the fingertip for index and middle fingers of 14 students of around 27 years old, including 4 females and 10 males. The fingertip height is the distance from the fingertip pulp to the fingernail. The fingertip length is the distance between the fingertip pulp to the far front of the finger. When people touch the screen, they generally use the upper half of the fingertip to touch the middle of the key so that the key can be effectively pressed. We can see that half of the fingertip is around 6.5mm, less than the key height for all devices in Table 1. Moreover, according to Tables 1 and 2, the fingertip width is smaller than the key length. Therefore, the fingertip generally lands inside the key area, as shown in Figure 13. That is, the far front of the fingertip F in Figure 13 is in the range of the key and the touched point is inside the key area. Based on the perspective projection, T_o is on the segment of $\overline{K_f K_b}$ so that T'_o is on the segment of $\overline{K'_f K'_b}$ whenever the fingertip is in the view of the camera.

On a QWERTY keyboard of iPhone and other small smartphones, keys are very small. In these scenarios, people often use vertical touching or touch with the fingertip side in order not to touch wrong keys. That is, the fingertip top lands in the key area. The analysis above is still valid. Our experiments on the QWERTY keyboard also validate this analysis.

Table 1: Unlock Screen Keypad Size - Height × Length (mm)

	iPad	iPhone 5	Nexus 7
Height (mm) × Length (mm)	9 × 17	8 × 16	10 × 16

Table 2: Fingertip Size (σ - Standard Deviation)

	Index Finger		Middle Finger	
	Average	σ	Average	σ
Height (mm)	9.6	1.2	10.4	1.3
Length (mm)	12.9	1.6	13.1	1.7
Width (mm)	13.1	1.9	13.7	1.7

There are cases that T'_o is not on the line segment $\overline{K'_f K'_r}$, corresponding to the touched key $\mathcal{K}$. Figure 14 illustrates such a case. Please note we intentionally draw a large finger for clarity. In this case, the key, such as one on a keyboard for a non-unlock screen, is so small. The camera is too close to the finger and takes such a wrong angle that T_o lands outside $\overline{K_f K_r}$. Therefore, T'_o is not on the line segment $\overline{K'_f K'_r}$. In such cases, our observation is that T'_o generally lands into the far rear part of the key $\mathcal{K}'$ in front of $\mathcal{K}$. We define a percentage α. If an estimated touched point lands in the rear α of $\mathcal{K}'$, the touched key is $\mathcal{K}$.

We now derive the size of a key in an image and investigate its impact. The camera focus length is f. The height from the camera to the touch screen surface is h. The physical key height $|\overline{K_f K_b}| = w$. The distance between the key front K_f and the lens center is d. By geometry operations, we have

$$|K'_f K'_b| = \frac{fh}{d(1 + d/w)}. \quad (4)$$

If the physical key length is l, the key length l' in the image is,

$$l' = \frac{fl}{d}. \quad (5)$$

From Formulas (4) and (5), the farther the touch screen from the camera, the smaller the size of the key in the image. The smaller the physical key size, the smaller of the key in an image. Table 3 gives the camera specifications of the cameras used in our experiments: Logitech HD Pro Webcam C920 [26], the iPhone 5 camera and the Google glass camera. If the camera is around 2 meters away and half a meter away from the target, according to Formula (4) and our experiments, the key height is only a few pixels. Therefore, in our experiments, we often need to zoom the fingertip image for accurate localization of touched points. We can also derive the key size in the touching frames practically by using the homography from the reference image to the touching frames. The key area in the reference image is known, thus the key size in the touching frames can be derived.

4.2 Clustering-based Recognition of Touched Points

Based on the model of the touching finger in an image, we now introduce the clustering-based strategy recognizing touched keys. If we can derive the position of the touched point T'_o in Figure 15, we can infer the corresponding key by applying the homography. The problem is how to identify this touched point[2]. Intuitively, since T'_o is far below the fingertip, which blocks light rays, T'_o should be in the darkest area around the fingertip in the image.

We now analyze the brightness of the area around the fingertip. The fingertip is a very rough surface at the microscopic level and can be treated as an ideal diffuse reflector. The incoming ray of light is reflected equally in all directions by the fingertip skin. The

[2]Touched points actually form an area under the fingertip.

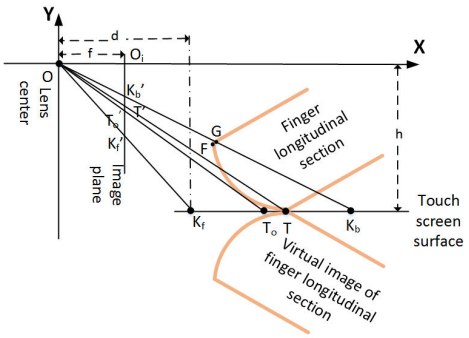


Figure 13: Touched Point inside the Key

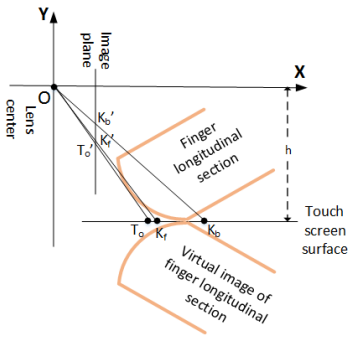


Figure 14: Touched Point outside the Key

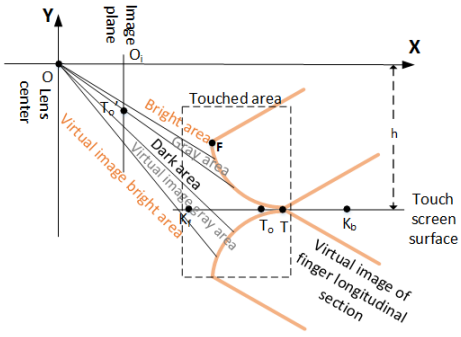


Figure 15: Five Pixel Groups at Fingertip

reflection conforms to the Lambert's Cosine Law [38]: the reflected energy from a small surface area in a particular direction is proportional to cosine of the angle between the particular direction and the surface normal. Therefore, for the lower part of the fingertip arc facing the touch screen, denoted as the inner side of the fingertip, the angle is large and less energy will be reflected so that the pixels are darker. Particularly, the area around $T_o^{'}$ is the darkest, i.e., touched points are the darkest. The area around the fingertip top F is the brightest. From the bright area to the dark area, there exists the gray area between F and $T_o^{'}$ in Figure 15. Since the touch screen is basically a mirror, the camera may also capture the virtual image of the inner side of the fingertip, which also has a dark area, gray area and bright area.

Table 3: Camera Specifications

Camera	Focal Length (mm)	Pixel Width (μm)
Logitech C920	3.67	3.98
iPhone 5	4.10	1.40
Google glass	2.80	0.18

Therefore, around the fingertip and its virtual image, we can have five areas with five different brightness: bright fingertip top, gray fingertip middle area, dark fingertip bottom and its virtual image (dark fingertip bottom, dark fingertip bottom of the virtual image), gray fingertip middle area of the virtual image and bright fingertip top of the virtual image. $T_o^{'}$ lands in the upper half part of the dark area since the other half of the dark area is the virtual image of the dark fingertip bottom.

We can use clustering algorithms to cluster these five areas of pixels of different brightness. The k-means clustering is applied to pixels in the tiny bounding box in Figure 10. The number of clusters is set as 5. The darkest cluster $\mathcal{C}$ indicates the area where the finger touches the screen surface. We automatically select a pixel in the upper half $\mathcal{S}$ of $\mathcal{C}$ as the touched point in the following way: a. the coordinate of a pixel p is (x, y), where x is the column number and y is the row number. Therefore,

$$\mathcal{S} = \{p | p \in \mathcal{C},\ p.y < \text{median of } y \text{ of all pixels in } \mathcal{C}\}. \quad (6)$$

b. Derive the minimal upright bounding rectangle $\mathcal{R}$ for pixels in $\mathcal{S}$. *The touched point is chosen from $\mathcal{S}$ and is the closest one to the center of the bounding rectangle $\mathcal{R}$.* This touched point is then mapped to the reference image, and the mapped point shall fall onto the correct key, denoted as *mapped key* $\mathcal{K}$. Denote the key behind $\mathcal{K}$ as $\mathcal{K}_b$, seen through the camera's "perspective". As discussed in Section 4.1, we also need to check the distance between the mapped point and the back edge of $\mathcal{K}$. If the touched points lands in the back α portion of $\mathcal{K}$, then we choose $\mathcal{K}_b$ as the real mapped key. Our experiments show that the optimal α is 1/5.

Basically, the clustering algorithm helps *accurately* identify the touched point. As an example, Figure 16 (a) shows the clustered result of the area in the green and tiny bounding box. The green point is the touched point in the *upper half of the darkest area*. The Figure 16 (b) shows the mapped point (in green) that falls into the quite front part of key 5. Therefore, 5 is the touched key.

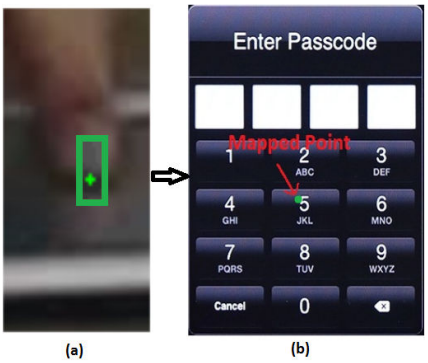


Figure 16: Clustered Result and Mapped Point

If we examine Figure 13 carefully, we can see that in addition to touched points, points on the fingertip arc may also be projected into the area of a key $\overline{K_f' F_b'}$ on the image plane. In Figure 13, line $\overline{OK_b}$ and the finger intersect at the point G. We can see that all points on the fingertip arc $\overline{TG}$ visible to the camera are projected into the area of the key in the image. Therefore, in this case, both touched points and points on this fingertip arc can be used to deduce the key even if the points on the fingertip arc are in the bright or gray area from our clustering above. However, due to the size of the fingertip, touched position, touching gestures, and distance, height and angle of the camera, G's position changes too and can be any point on the fingertip arc. It is not reliable to use these points on the fingertip arc to infer the touched key. We still use touched points in the darkest area around the fingertip top, but the fact that points in the gray or bright area may be projected into the key's area lends us some robustness to use touched points in the darkest area to infer the touched key.

5. EVALUATION

In this section, we present the experiment design and results to demonstrate the impact of the blind recognition of touched keys.

5.1 Experiment Design

We have performed extensive experiments on various target devices with different key size, including iPad, iPhone and Nexus 7 tablet. Three cameras are used: Logitech HD Pro Webcam C920, iPhone 5 camera and Google Glass. Table 3 summarizes their specifications. Most experiments are performed with Logitech HD Pro Webcam C920. The last group of experiments are designed for comparing web camera, iPhone camera and Google glass against d-

ifferent devices as well as the impact of different kinds of keyboards. In all experiments, we try to recognize 4-digit or 4-character passcodes, which are randomly generated. The success rate is defined as the probability that the passcodes are correctly recognized.

In addition to different cameras and target devices, we also consider the impact from the following factors: users, the distance between the camera and target device, and the camera angle.

Users: Different people have different finger shape, fingernail and touching gestures. Five females and six males with the experience of using tablets and smartphones participated in the experiments. They were separated to three groups: 3 people in the first group, and 7 people in the second group. These two groups performed experiments with iPad. The last group helped us evaluate the success rate versus the distance between the camera and the target, different cameras versus different devices, and the web camera versus different kinds of keyboards. For the first group, we took 10 videos for every person at each angle (front, left and right of the target device). For the second group, five videos were taken for every person per angle. Discarding 3 videos not recording the whole touching process, we obtain 192 videos totally. During the experiments, users tap in their own way without any intervention.

Angles and Distance: To measure the impact of the angle, we placed the target in front, on the left (3 o'clock) and on the right (9 o'clock) of the camera. In the first two groups of experiments, the camera was 2.1 meters (m) to 2.4m away from and around 0.5m above the device. To test how the distance affects the recognition results, we also positioned the camera, the Logitech HD Pro Webcam C920, in front of the target device, an iPad, at a distance of 2m, 3m, 4m and 5m, and approximately one meter above the target.

Lighting: The lighting affects the brightness and contrast of the image. The experiments are performed in a classroom with dimmable lamps on the ceiling. The first group of videos were taken under normal lighting and the second group of experiments were taken under strong lighting. All other experiments were performed under normal lighting. Darkness actually helps the attack since the touch screen is brighter in the dark. We did not consider these easy dark scenes in our experiments.

5.2 Detecting Touching Frames via Optical Flow

As discussed in Section 3.4, we track a hand's feature points and use their velocity change to detect touching frames. Our experiments show that 5 or more feature points are stable for tracking touching frames with a true positive of 100%, as shown in Table 4. The optical flow algorithm may also produce false positives, falsely recognizing frames in which a finger does not touch the screen surface as touching frames. The false positive rate is very low, less than 1% as shown in Table 4. One way to reduce the false positive is to use DPM. Our experiments show that DPM is able to detect touching frames since no fingers touch the screen in non-touching frames and DPM only recognizes touching fingers in touching frames. We exclude the non-touching frames by DPM if the number of detected touching frames is more than 4 by optical flow.

Table 4: Performance of Detecting Touching Frames

	Front	Left	Right	Average
True Positive	100%	100%	100%	100%
False Positive	0.91%	0.88%	0.88%	0.89%

5.3 Recognizing Touched Keys on iPad via Webcam

Table 5 shows the result of the baseline method for videos taken from different viewpoints. Its overall success rate is less than 30%. Therefore, the baseline method is not very effective since DPM cannot accurately locate the touched points.

Table 5: Success Rate by Baseline Method

	Front	Left	Right	Average
Success Rate	26.66%	29.03%	22.22%	26.13%

From now on, we present experiment results using the seven-step recognition method, referred to as Automatic Apprpach (AA), introduced in Section 3. We also use a metric called the Best Effort Approach (BEA) success rate, which is derived by giving a second attempt for correcting a wrong recognition with some human intervention. The BEA is performed in the following way. We often see one or two wrong keys in the failed experiments. Some of these wrong keys are caused by DPM that fails to detect the touching fingertip. Sometimes, even if the touching fingertip is detected, the image can be so blurry that pixels around the touching fingertip have almost the same color and it is difficult to derive the fingertip contour in Figure 9. Other fingers may also block the touching finger and incur wrong recognition of the touching fingertip top. Therefore, we often know which key might be wrong and give them a second try. We manually select the small bounding box of the fingertip in Figure 8 or the touched area in Figure 10 to correct such errors. As analyzed in Section 4, for each touch we may also produce two candidates. Using one of the two choices, we may correct the wrong keys in the second time try. Therefore, the BEA success rate is higher than the AA success rate.

Table 6 gives the success rate of recognizing touched keys from videos taken at different angles. Recall that the success rate is the ratio of the number of the correctly recognized passcodes (all four digits or characters) over the number of passcodes. For the wrong results, we give a second attempt by applying the Best Effort Approach. It can be observed that the overall AA success rate reaches more than 80%. The success rate for videos taken from the left and right is a little lower because there are some quite blurry videos, which are difficult to analyze for Step 6. The BEA success rate is higher than the AA success rate and reaches over 90%. The per digit success rate is defined as the ratio between the number of correctly retrieved digits and the number of all the digits. The per digit success rate for BEA is more than 97%.

Table 6: Success Rate by Clustering Based Matching

	Front	Left	Right	Average
Automatic Apprpach	92.18%	75.75%	79.03 %	82.29%
Best Effort Approach	93.75%	89.39%	90.32%	91.14%
Per Digit for BEA	98.04%	96.59%	97.58%	97.39%

Figure 17 presents the results of measuring the impact of the distance between the camera and the target on the success rate. It can be observed that the trend is: as the distance increases, the success rate decreases. At the distance of 4m or 5m, the AA success rate is as low as 20%. This is because at such a distance the keys in the image are so small that they are only 1 or 2 pixel wide. It is much more difficult to distinguish a touched key at such a distance. A camera with a high optical zoom shall help. However, our threat model does not allow the use of those high zoom cameras.

To test whether human can retrieve the passcodes easily, we asked all people involved in the experiments for human based recovery. Given the tiny software keyboard and no text or popup in the recorded video, nobody could get the whole 4-digit passcode right. And, it is almost impossible for human to recognize keys on a QWERTY keyboard given so many keys and small key size in a video.

5.4 Comparing Different Targets and Cameras

To compare the success rate of recognizing touched keys on different devices, we performed 30 experiments on Nexus 7 and iPhone 5 respectively with the Logitech HD Pro Webcam C920 from

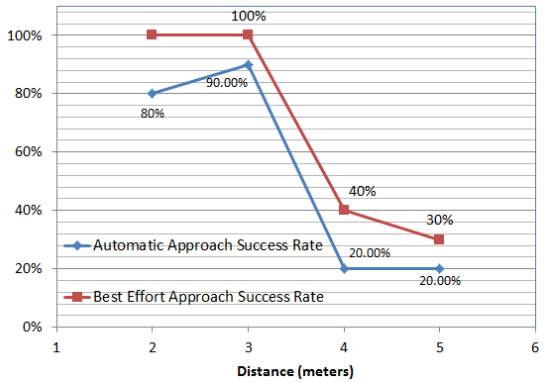


Figure 17: Success Rate v.s. Distance

two meters away from and about 0.65m above the device. To investigate the impact of different cameras, we conducted 30 experiments using iPhone 5 to record passcode inputs on iPad, from a similar distance and at a similar height. 30 experiments with the Google glass recording passcode inputs on iPad were performed two meters away and at a human height. Figure 18 presents the results. The AA success rate is more than 80%, and the BEA success rate is more than 90% in all cases. The high success rate for all the cases demonstrates the severity of our attack.

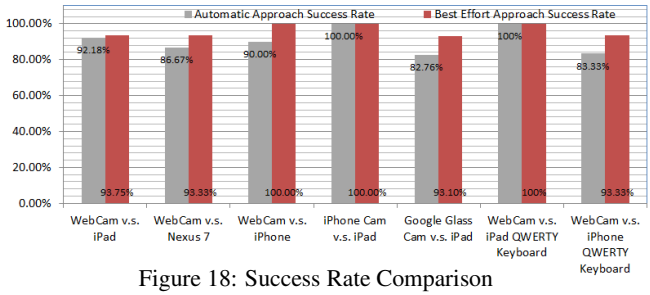


Figure 18: Success Rate Comparison

We also tested the effect of our attack on different kinds of keyboard: the iPad QWERTY keyboard and iPhone QWERTY keyboard. The iPad QWERTY keyboard key is larger than the iPhone QWERTY keyboard key. 30 experiments were done respectively with the web camera from the front of the target from about 2.2 meters away and at a height of 0.6 meters. Figure 18 presents the results. It can be observed that the AA success rate is over 80% and the BEA success rate is over 90%!

6. DISCUSSION: TOUCHING WITH MULTIPLE FINGERS AND TWO HANDS

As shown in Figure 19, people may type the iPad passcodes, or their bank account passwords on a large keyboard (in Figure 21) with multiple fingers and two hands. In this section, we extend our work in the previous sections and discuss the recognition of touch input by two hands and multiple fingers.

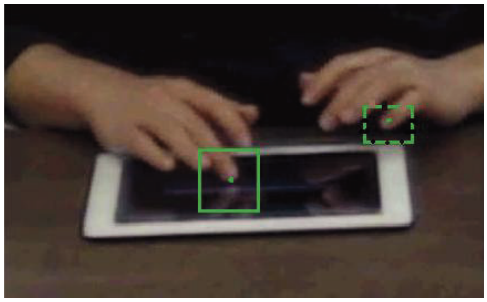


Figure 19: Touching Frame with Multiple Fingers (Magnified)

In the case of multiple fingers, the challenge is to recognize which finger is the touching finger. In [43], touching is defined as a fingertip hovering at a certain position or suddenly changing the moving direction for another key. In the case of touching with multiple fingers, this is not true any more. As shown in Figure 19, the non-touching fingertips also follow a similar pattern during the touching process if both the little finger of the right hand and middle finger of the left hand are used for typing. The strategy in [43] for deriving the fingertip's moving trajectory cannot work directly if multiple fingertips are involved.

In Section 3.4 when we address the case of touching with one finger, we use the finger's velocity direction change to detect the touching frames. The touching finger and other fingers follow a similar pattern of velocity direction change. The touching frame is the one in which the majority of those fingers change the velocity from positive to negative. In the case of multiple fingers, in order to use the similar strategy, we have to differentiate and track the two hands. The complicated background involving the two hands is a great challenge for a general solution.

We address the challenges above by looking more closely at what is touching. The finger movement registers a touch input only when the fingertip actually touches the screen. Therefore, we may detect touching frames by detecting touching actions from the perspective of action detection [45] (also termed action localization [24], or event detection [20]). A video can be modeled as a sequence of frames captured along the time, as shown in Figure 20. We can detect each touch action with SDPM (Spatiotemporal Deformable Part Model [39]). One touch event usually involves several touching frames. Therefore, we can treat the touching action detection problem as the problem of detecting a set of touching frames.

We use DPM to detect touching frames and localize the touching finger in the case of typing with multiple fingers. As discussed in Section 3.6, DPM can localize the touching fingertip in touching frames effectively. Therefore, by applying DPM to all video frames, we can detect and localize all the touching fingertips in the touching frames, excluding non touching fingertips. The bounding box in Figure 20 shows the detected touching fingertip in the case of typing with multiple fingers. It is also observed that the touch event involves a few consecutive touching frames along the t-axis as shown in Figure 20.

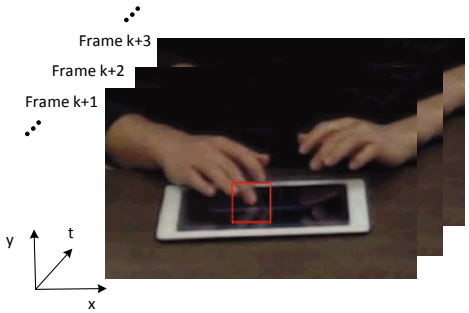


Figure 20: One Touch Event Involving Multiple Frames in a Video

For every touching action, we use the touching frame in the middle as the actual touching frame. Given the touching frame, we adopt the same steps introduced in Section 4 to derive the touched key. Figure 21 shows the mapped result (green dot) for the touch event in Figure 20, where "U" is the touched key. From the discussion above, we can also see that touching with one finger (or hand) is a special case of touching with two hands and multiple fingers.

To validate the attack against typing with multiple fingers, we performed 21 experiments with the web camera spying on the iPad character keyboard, from distance about 2.2 meters away and

Figure 21: Multi Touching Mapped Result

0.65 meters above the device. We achieve the AA success rate of 95.24%. In the experiments, only one touch event was not correctly detected. If we manually retrieve the touching frame for that touch event as introduced in previous sections, we get the BEA success rate of 100% to retrieve the character passcode by two hands and multiple fingers. This demonstrates the correctness and severity of the attack introduced in this paper.

7. RELATED WORK

In this paper, we exploit the movement of the touching finger to infer the input on a touch screen. It is one type of side channel attack. There are various similar attacks on touch-enabled devices exploiting different hidden venues. Because of the space limit, we discuss the most related work on side channels using computer vision knowledge. Backes *et al*. [2, 1] exploit the reflections of a computer monitor on glasses, tea pots, spoons, plastic bottles, and eyes of the user to recover what is displayed on the computer monitor. Their tools include a SLR digital camera Canon EOS 400D, a refractor telescope and a Newtonian reflector telescope, which can successfully spy from 30 meters away.

Balzarotti *et al*. propose an attack retrieving text typed on a physical keyboard from a video of the typing process [3]. When keys are pressed on a physical keyboard, the light diffusing surrounding the key's area changes. Contour analysis is able to to detect such a key press. They employ a language model to remove noise. They assume the camera can see fingers typing on the physical keyboard. Maggi *et al*. [28] implement an automatic shoulder-surfing attack against touch-enabled mobile devices. The attacker employs a camera to record the victim tapping on a touch screen. Then the stream of images are processed frame by frame to detect the touch screen, rectify and magnify the screen images, and ultimately identify the popping up keys. Raguram *et al*. exploit refections of a device's screen on a victim's glasses or other objects to automatically infer text typed on a virtual keyboard [33]. They use inexpensive cameras (such as those in smartphones), utilize the popup of keys when pressed and adopt computer vision techniques processing the recorded video in order to infer the corresponding key although the text in the video is illegible.

Xu *et al*. extend the work in [33] and track the finger movement to infer input text [43]. Their approach has six stages: in Stage 1, they use a tracking framework based on AdaBoost [13] to track the location of the victim device in an image. In Stage 2, they detect the device's lines, use Hough transform to determine the device's orientation and align a virtual keyboard to the image. In Stage 3, they use Gaussian modeling to identify the "fingertip" (not touched points as in our work) by training the pixel intensity. In Stage 4, RANSAC is used to track the fingertip trajectory, which is a set of line segments. If a line segment is nearly perpendicular to the touch screen surface, it implicates the stopping position. In Stage 5, they apply image recognition techniques to determine which keys are most likely pressed given the stopping positions. In Stage 6, they apply a language model to optimize the result given the candidate keys and associated confidence values from the previous stage. They use two cameras: Canon VIXIA HG21 camcorder with 12x optical zoom and Canon 60D DSLR with 400mm lens.

In comparison with [43] on recognizing passwords, we can achieve a much higher success rate. We extend our work to the scenario of touching with both hands and multiple fingers while such a scenario is not addressed in [43].

8. CONCLUSION

In this paper, we present a computer vision based attack that blindly recognizes inputs on a touch screen from a distance automatically. The attack exploits the homography relationship between the touching images (in which fingers touch the screen surface) and the reference image of a software keyboard. We use the optical flow algorithm to detect touching frames. The deformable part-based model (DPM) and various computer vision techniques are utilized to track the touching fingertip and identify the accurate touched area. We carefully analyze the image formation of the touching fingertip and design the k-means clustering strategy to recognize the touched points. The homography is then applied to recognize the touched keys. Our extensive experiments show that the AA success rate of recognizing touched keys is more than 80%, while the BEA success rate is more than 90%. We have also extended the attack to the case of typing with two hands and multiple fingers and achieve a high success rate of more than 95%. As a countermeasure, we design a context aware Privacy Enhancing Keyboard (PEK) which pops up a randomized keyboard on Android systems for sensitive information input such as passwords. Our future work includes further refinement of the attack and design of alternative authentication strategies for mobile devices.

9. ACKNOWLEDGEMENTS

This work is supported in part by National Key Basic Research program of China under grant 2010CB328104, Macau FDCT 061-2011-A3, International S&T Cooperation Program of China grant 2013DFA10690, US NSF grants 1116644, CNS-1318948 and 1262275, National Science Foundation of China under grant 61272054. 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.

10. REFERENCES

[1] M. Backes, T. Chen, M. Duermuth, H. Lensch, and M. Welk. Tempest in a teapot: Compromising reflections revisited. In *Proceedings of 30th IEEE Symposium on Security and Privacy*, pages 315–327, 2009.

[2] M. Backes, M. Dürmuth, and D. Unruh. Compromising reflections or how to read lcd monitors around the corner. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 158–169, 2008.

[3] D. Balzarotti, M. Cova, and G. Vigna. Clearshot: Eavesdropping on keyboard input from video. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, SP '08, pages 170–183, 2008.

[4] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[5] H. Benko, A. D. Wilson, and P. Baudisch. Precise selection techniques for multi-touch screens. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, pages 1263–1272, 2006.

[6] R. Biddle, S. Chiasson, and P. van Oorschot. Graphical passwords: Learning from the first twelve years. In *ACM Computing Surveys*, 2012.

[7] G. R. Bradski and A. Kaehler. *Learning opencv, 1st edition*. O'Reilly Media, Inc., first edition, 2008.

[8] A. Bulling, F. Alt, and A. Schmidt. Increasing the security of gaze-based cued-recall graphical passwords using saliency masks. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2012.

[9] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, 1986.

[10] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, pages 886–893. IEEE Computer Society, 2005.

[11] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32:1627–1645, 2010.

[12] C. Forlines, D. Wigdor, C. Shen, and R. Balakrishnan. Direct-touch vs. mouse input for tabletop displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 647–656, 2007.

[13] H. Grabner, M. Grabner, and H. Bischof. Real-time tracking via on-line boosting. In *Proceedings of the British Machine Vision Conference*, 2006.

[14] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2 edition, 2003.

[15] S. B. Hirsch. Secure input system. In *United States Patent No. 4,479,112*, 1982.

[16] S. B. Hirsch. Secure keyboard input terminal. In *United States Patent No. 4,333,090*, 1982.

[17] B. Hoanca and K. Mock. Screen oriented technique for reducing the incidence of shoulder surfing. In *Proceedings of the International Conference on Security and Management (SAM)*, 2005.

[18] P. Huber. *Robust Statistics*. John Wiley & Sons, 1981.

[19] Juniper Networks, Inc. Juniper networks third annual mobile threats report. **http://www.juniper.net/us/en/local/pdf/additional-resources/3rd-jnpr-mobile-threats-report-exec-summary.pdf**, 2013.

[20] Y. Ke, R. Sukthankar, and M. Hebert. Efficient visual event detection using volumetric features. In *Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1 - Volume 01*, ICCV '05, pages 166–173, 2005.

[21] D. Kim, P. Dunphy, P. Briggs, J. Hook, J. W. Nicholson, J. Nicholson, and P. Olivier. Multi-touch authentication on tabletops. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2010.

[22] I. Kim. Keypad against brute force attacks on smartphones. In *IET Information Security*, 2012.

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

[24] T. Lan, Y. Wang, and G. Mori. Discriminative figure-centric models for joint action localization and recognition. In *International Conference on Computer Vision (ICCV)*, 2011.

[25] C. Lee. System and method for secure data entry. In *United States Patent Application Publication*, 2011.

[26] Logitech. Logitech hd pro webcam c920. **http://www.logitech.com/en-us/product/hd-pro-webcam-c920**, 2013.

[27] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004.

[28] F. Maggi, S. Gasparini, and G. Boracchi. A fast eavesdropping attack against touchscreens. In *IAS*, pages 320–325. IEEE, 2011.

[29] J. Matas, C. Galambos, and J. Kittler. Robust detection of lines using the progressive probabilistic hough transform. *Comput. Vis. Image Underst.*, 78(1):119–137, 2000.

[30] K. E. McIntyre, J. F. Sheets, D. A. J. Gougeon, C. W. Watson, K. P. Morlang, and D. Faoro. Method for secure pin entry on touch screen display. In *United States Patent No. 6,549,194*, 2003.

[31] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *In VISAPP International Conference on Computer Vision Theory and Applications*, pages 331–340, 2009.

[32] Plugable. Plugable usb 2.0 digital microscope for windows, mac, linux (2mp, 10x-50x optical zoom, 200x digital magnification). **http://www.amazon.com/Plugable-Digital-Microscope-Windows-Magnification/dp/B00AFH3IN4/ref=sr_1_1?ie=UTF8&qid=1382796731&sr=8-1&keywords=optical+zoom+webcam**, 2013.

[33] R. Raguram, A. M. White, D. Goswami, F. Monrose, and J.-M. Frahm. ispy: automatic reconstruction of typed input from compromising reflections. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 527–536, 2011.

[34] 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 30th ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2012.

[35] J. Shi and C. Tomasi. Good features to track. Technical report, 1993.

[36] H.-S. Shin. Device and method for inputting password using random keypad. In *United States Patent No. 7,698,563*, 2010.

[37] X. Suo, Y. Zhu, and G. S. Owen. Graphical passwords: A survey. In *Proceedings of Annual Computer Security Applications Conference (ACSAC)*, 2005.

[38] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag New York, Inc., 1st edition, 2010.

[39] Y. Tian, R. Sukthankar, and M. Shah. Spatiotemporal deformable part models for action detection. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '13, pages 2642–2649, 2013.

[40] F. Wang, X. Cao, X. Ren, and P. Irani. Detecting and leveraging finger orientation for interaction with direct-touch surfaces. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, UIST '09, pages 23–32, 2009.

[41] F. Wang and X. Ren. Empirical evaluation for finger input properties in multi-touch interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 1063–1072, 2009.

[42] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. pages 3485–3492. IEEE, 2010.

[43] 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 Conference on Computer and Communications Security (CCS)*, 2013.

[44] 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.

[45] Y.Hu, L. Cao, F.Lv, S.Yan, Y.Gong, and T.S.Huang. Action detection in complex scenes with spatial and temporal ambiguities. *ICCV*, 2009.

[46] J. yves Bouguet. Pyramidal implementation of the lucas kanade feature tracker. *Intel Corporation, Microprocessor Research Labs*, 2000.

[47] Y. Zhang, P. Xia, J. Luo, Z. Ling, B. Liu, and X. Fu. Fingerprint attack against touch-enabled devices. In *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*, SPSM '12, pages 57–68, 2012.

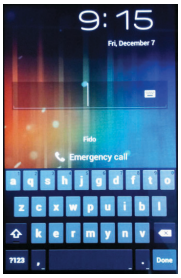
Appendix

In this appendix, we discuss countermeasures to computer vision based attacks introduced in this paper and related work. There are many other authentication approaches immune to these attacks to some extent, including biometric-rich gesture-based authentication [34, 44, 21] and graphic password schemes [6, 37, 8]. The idea of randomized keyboard has been proposed for legacy keypad and touch-enabled devices [16, 15, 30, 17, 36, 25, 22]. We have designed and developed the context aware Privacy Enhancing Keyboards (PEK) for Android for the first time. We have implemented the PEK as a third party app and are also able to change the internal system keyboard to implement the PEK.

Design and Implementation of PEK

A software keyboard may contain three sub-keyboards. The primary sub-keyboard is the QWERTY keyboard, which is the most common keyboard layout. The second sub-keyboard is the numerical keyboard that may also contain some symbols. The last sub-keyboard is a symbol keyboard that contains special symbols. The layout for these three sub-keyboards is stored in a XML file, which records the positions of keys and corresponding key codes. The system generates its keyboard in this way: the keys will be read from the XML file and put in a right position.

PEK changes the key layout to implement randomized keys. When a keyboard is needed, we first generate a random sequence of key labels for each of the three different keyboards. When a key is read from the XML, we randomly choose an integer number between one and the size of the key sequence. We use this number to pick the specific key label from the randomized key sequence and also remove this label from the key sequence. This randomly selected key replaces the current key. In this way, we can shuffle the key positions on a normal keyboard. Another version of PEK is to make each key move within the keyboard region in a Brownian motion fashion by updating each key's position repeatedly according to the Brownian motion. In this way, the keys are moving all the time. Even if the same key is pressed a few times, their positions are different. This is an improvement compared with PEK with shuffled keys, in which the keyboard does not change in one session of password input. Figure 22 shows PEK with shuffled keys and Figure 23 shows PEK with the Brownian motion of keys.



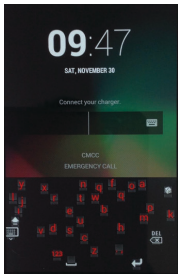


Figure 22: PEK-Shuffled Keys Figure 23: PEK-Brownian Motion

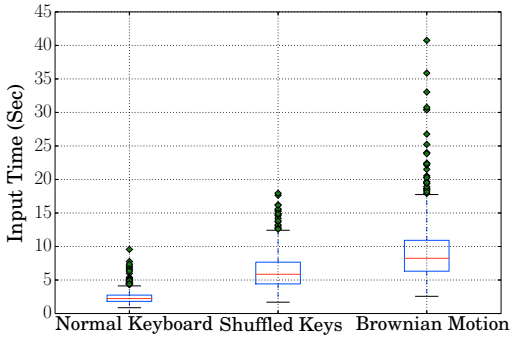


Figure 24: Input Time of Three Distinct Keyboards

PEK is aware of the context and can pop up the randomized keyboard only if the input box is for sensitive information. The Android class "EditorInfo" can be used to detect the input box type. In our case, **`TYPE_NUMBER_VARIATION_PASSWORD`**, **`TYPE_TEXT_VARIATION_PASSWORD`**, **`TYPE_TEXT_VARIATION_VISIBLE_PASSWORD`** and **`TYPE_TEXT_VARIATION_WEB_PASSWORD`** are used to identify the password input box. The first type is the variation of **`TYPE_CLASS_NUMBER`**, while the other three types are the variations of **`TYPE_CLASS_TEXT`**. Once the password input box is triggered by the user, a new randomized keyboard will be constructed. As a result, the user can have different key layouts every time she presses the password input box.

Evaluation of PEK

To measure the usability of the PEK, we recruit 20 students, 5 female students and 15 male students, whose average age is 25 years old. We implemented a test password input box and generated 30 random four-letter passwords. The students are required to input these 30 passwords by using a shuffled keyboard and a Brownian motion keyboard, and the test app records the user input time. Table 7 shows the results of our evaluation and Figure 24 gives a box plot of the input time of three different keyboards. The median input time is around 2.2 seconds on the normal keyboard, 5.9 seconds on the shuffled keyboard and 8.2 seconds on the Brownian motion keyboard. The success rate is the probability that a user correctly inputs four-letter passwords. The success rate of all three keyboards are high while it is a little bit lower for the Brownian motion keyboard. The participants in our experiments feel PEK is acceptable if PEK only pops up the randomized keyboard for sensitive information input.

Table 7: Usability Test

	Normal Keyboard	Shuffled Keys	Brownian Motion
Median Input Time (Second)	2.235	5.859	8.24
Success Rate	98.50%	98.83%	94.17%