Blind Recognition of Touch Inputs on Mobiles

Qinggang Yue, Zhen Ling, Anthony Topper, Xinwen Fu, *Member, IEEE,* Benyuan Liu, *Member, IEEE,* Wei Yu, *Member, IEEE,* Kui Ren, *Senior Member, IEEE,* and Wei Zhao, *Fellow, IEEE*

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 experiments show that 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. If the touch input is meaningful text, we also apply the trigram language model for spelling correction. In our experiments, with the iPhone camera spying on an iPad from about 5 meters away, the basic recognition technique can recover 48% of the words while the trigram model achieves a success rate of 96%. As a countermeasure, we introduce the context aware Privacy Enhancing Keyboard (PEK) which randomizes the keyboard layout during the password/passcode input, but uses the QWERTY keyboard for other inputs. We implemented the first ever Android PEK, which has been downloaded 1600+ times on Google Play.

Index Terms—Computer Vision, Attack, Mobile Devices, Natural Language Processing

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 [1], 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 identifies text on screen or its reflections on objects [2], [3]. The second group of attacks detects visible features of the keys such as light diffusion surrounding pressed keys [4] and popups of pressed keys [5], [6]. The third group of attacks is able to blindly recognize the text input on mobile devices while text or popups are not visible to the attacker [7] .

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, which maps 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 based 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 on various victim target devices, including the iPad, Nexus 7, and iPhone 5. Both login keyboard and QWERTY keyboard are examined. The cameras include a webcam, a phone camera, a smartwatch and Google Glass. The camera is positioned from different distances and angles. Our experiment data shows that we are able to achieve a per-key success rate of over 97% and a 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 in the paper. 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 patterns, 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 fingers. Our experiment data shows that we are able to recognize the touching finger from 10 fingers and achieve a high success rate of more than 95% recognizing passcodes.

In this paper, in addition to passwords, we also demonstrate the use of natural language processing (NLP) techniques to recognize meaningful text such as emails entered on touch-enabled devices.

- *Qinggang Yue, Benyuan Liu and Xinwen Fu are with the Department of Computer Science, University of Massachusetts Lowell, MA 01854. E-mail: {qye, bliu, xinwenfu}@cs.uml.edu.*
- *Zhen Ling is with Southeast University, Nanjing, Jiangsu, China, 210018. E-mail: zhenling@seu.edu.cn.*
- *Anthony Topper is with Advanced Math And Science Academy Charter School, Marlborough, MA 01752. E-mail: anthony@topperstudios.com.*
- *Wei Yu is with Towson University, Towson, MD 21252. E-mail: wyu@towson.edu.*
- *Kui Ren is with University at Buffalo (UB), State University of New York (SUNY), Buffalo, NY 14260. E-mail: kuiren@buffalo.edu.*
- *Wei Zhao is with University at Macau, Taipa, Macau, China. E-mail: weizhao@umac.mo.*

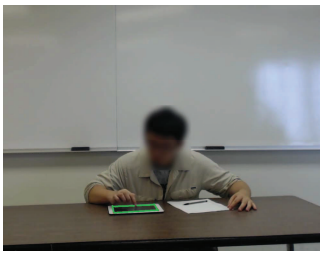


Fig. 1. Touching Frame

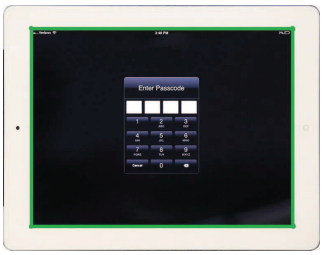


Fig. 2. Soft Keyboard

The problem of retrieving the text can be treated as a spelling correction problem. We have introduced the unigram model in [8]. In this paper, we apply the n-gram language model, particularly the trigram language model, to perform further correction. Words in a sentence are not isolated, follow the grammar rules and have their context. Such information can be used for the correction. We model the sentence as a graph based on the Hidden Markov Model and apply the n-gram language model to derive the most possible sentences. The capability of error correction via NLP techniques increases the distance of spying on touch-enabled devices.

As a countermeasure, we introduce the context aware Privacy Enhancing Keyboard (PEK) which randomizes the keyboard layout during the password/passcode input, but uses the QWERTY keyboard for other inputs. We implemented the first ever Android PEK, which has been downloaded 1600+ times on Google Play. PEK's design and implementation are discussed in [9]. In this paper, we introduce an alternative to PEK, a randomized lock screen keypad app, which is favorable by users who do not want to use a third party keyboard. The performance of the unlock screen keypad is similar to PEK, as reported in [9]. However, we design our unlock screen keypad in such a way that other authenitcation schemes such as novel graphical passwords can be implemented by just changing a couple of classes.

The rest of the paper is organized as follows. In Section 2, we introduce the computer vision based attack against touch inputs. In Section 3, we discuss how to recognize touched points from touching frames. In Section 4, we introduce the tri-gram model to correct errors in a recovered sentence by our computer vision attack. In Section 5, we present experimental design and evaluations. We introduce an alternative to our Privacy Enhancing Keyboard (PEK), a randomized lock screen keypad app in Section 6. In Section 7, we discuss the related work. Finally, we conclude the paper in Section 8.

2 Homography Based Attack against Touch Screen

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

2.1 Basic Idea

Figure 3 shows the flow chart of the automatic and blind recognition of touch inputs on mobile devices. The work flow of the automatic and blind recognition of touch inputs on mobile devices is described as follows:

Step 1 - Taking Videos. An attacker takes a video of a victim tapping on a device. Notice that we do not assume the video records any text or popups, while we assume that the finger movement and the target device's screen surface are recorded.

Step 2 - Preprocessing. In this step, the video will be preprocessed and only the touch screen area with moving fingers will be kept. 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 - Detecting Touching Frames. In this step, the *touching frames*, in which the finger touches the screen surface, will be detected, as shown in Figure 1.

Step 4 - Deriving the Homography Matrix. In this step, features of the touch screen surface will be identified and the planar homography matrix between the touching frames and the reference image will be derived.

Step 5 - Locating the Touching Fingertip. In this step, DPM and various computer vision techniques are used 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 locate the touched point that can be mapped to the reference image and to recognize the touched key. We denote the direct use of DPM finding the touched point as the baseline method.

Step 6 - Estimating the Touched Area. In this step, the fingertip contour in the large bounding box will be identified and a tiny bounding box around the fingertip top as the accurate touched area will be trained.

Step 7 - Recognizing Touched Keys. In this step, a model of the touching process will be established, the touched points from the estimated tiny touched area will be identified, and mapping those identified points to the reference image will be performed via the homography. If the touched points can be correctly located, we can disclose the corresponding touched keys.

Step 8 - Error Correction. In this step, the errors will be corrected by applying language models, if the retrieved text is comprised of meaningful English text. Spelling correction techniques based on the Hidden Markov Model are applied to the text in order to correct the errors in the retrieved text.

In the next few subsections, we present these steps in detail.

2.2 Step 1 - Taking Videos

In this step, the attacker takes a video of a victim tapping on a device. Such scenarios include students taking classes, researchers attending conferences, tourists gathering and resting in a square, and many others. 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 also sells a webcam-like plugable 2MP USB 2.0 digital microscope with a 10x-50x optical zoom lens [10].

Three factors in taking videos could 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 the 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

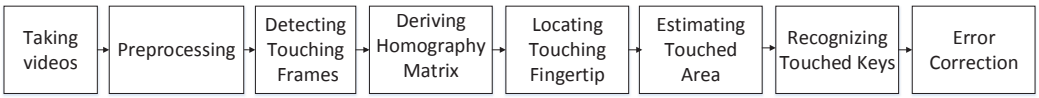


Fig. 3. Work flow of Blind Recognition of Touch Inputs

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.

2.3 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 and detect 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 obtain the positive data, we take 700 images of the target device, such as iPad, from different angles, and manually label the device with a bounding box. To obtain 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 upright bounding rectangle of the four points accurately delimits the device in the training image. To derive the negative data, we use 900 background images from the SUN database [11] and label objects with 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 the iPad. The first row models the iPad viewed from the right, and the second row models the iPad viewed from the left, and the third and fourth rows model the iPad viewed from the right front and left front of the iPad, respectively. The first column shows the root model (the coarse model characterizing the 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 the iPad. After training, we apply the learned model to the video frames, and the device is accurately localized, as shown in Figure 5.

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 the 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.

2.4 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 **??**.

During the touching process, the fingertip first moves downward toward 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 [12] 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 use the KLT algorithm [13], 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 [14] 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

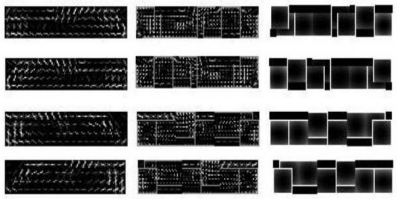


Fig. 4. Trained iPad DPM Model

Fig. 5. Detected iPad (Magnified)

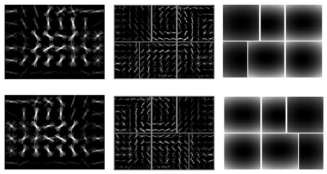


Fig. 6. DPM Model of the Touching Fingertip

Fig. 7. Detected Touching Fingertip

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.

2.5 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 [15]. First, a feature detector such as SIFT (Scale-Invariant Feature Transform) [16] or SURF (Speeded Up Robust Features) [17] is used to detect feature points. Matching methods such as FLANN (Fast Library for Approximate Nearest Neighbors) [18] 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) [19].

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 $\mathbf{H}$ 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 [15]. In our case, we try to use the corners of the touch screen as shown in Figures 1 and 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 [20] to extract the edges and use the Hough line detector [21] to derive candidate lines in the image. We manually choose the lines aligned to the edges. Notice that this is the only manual procedure in our entire system of blindly recognizing touched keys. After edges are derived, we can now compute 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 [15]. 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.

2.6 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 using the homography matrix in order to obtain 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 obtain 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 manner. 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 while we want a bounding box as small as possible. We evaluated bounding boxes of different sizes. The optimal bounding box in our context is the one bounding the fingertip, centered at the touched key and has a size of 40 $\times$ 30 pixels. If different bounding box sizes are used for training images, DPM resizes the bounded area to a uniform size. To obtain 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 in order to detect the touching fingertip. Figure 7 shows the detected result as the green large bounding box and its center C.

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 the "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 main reason is that DPM is still a very coarse object detector in our context.

2.7 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 a variety 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 the accurate touched area a great challenge. We employ two steps to this end. First, within the 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. These two steps are further described below.

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 used to cluster the pixel values into two categories (i.e., 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 [22] 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.

Fig. 8. Small Bounding Box

Fig. 9. Fingertip Contour and Direction

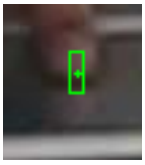


Fig. 10. Accurate Touched Area

2.8 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 3 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.

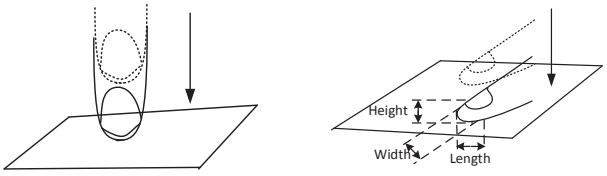
Once the actual touched points are located, we can then map them to the reference image. The corresponding points in the reference image are denoted as mapped points. Such mapped points should land in the corresponding key's area on the software keyboard. Therefore, we can derive the touched keys. This 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 3.

2.9 Step 8 - Error Correction

If the input from the user is meaningful text, it should follow some spelling and syntactic rules. We can apply natural language models to correct the errors. We first introduce the noisy channel model, and show how the non-word errors are corrected. After getting candidates for each possible word, we apply the unigram and trigram language model for the correction. The unigram language model corrects each word independently from neighboring words. With the trigram language model, we model the whole sentence as a graph with the Hidden Markov Model, and then derive the intended input text. In Section 4, we investigate error correction with the language models in detail.

3 Recognizing Touched Keys

In this section, we first model how people use their fingers 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.



(a) Vertical Touching (b) Oblique Touching

Fig. 11. Touching Gestures

3.1 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 [23], [24], [25], there are two types of touching gestures: (i) vertical touch, and (ii) 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 [24]. From Figure 11, the finger orientation (i.e., "yaw") relative to the touch screen may also be different [26]. The shape and size of a person's finger and key size also affect the touching gestures and the location where a key will be touched.

We now 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 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 $\overline{K_fK_b}$. T and $\overline{K_fK_b}$ are projected onto the image plane as T' and $\overline{K'_fK'_b}$, respectively. 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 $\overline{K'_fK'_b}$, 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 $\overline{K_fK_b}$ so that T'_o is on the segment of $\overline{K'_fK'_b}$ whenever the fingertip is in the view of the camera.

On a QWERTY keyboard of an 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 the wrong keys. That is, the fingertip top lands in the key area. The analysis above is still valid. Our experiments on the QWERTY keyboard validate this analysis as well.

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 $\overline{K'_fK'_r}$, 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 $\overline{K_fK_r}$. Therefore, T'_o is not on the line segment $\overline{K'_fK'_r}$. 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 $|\overline{K_fK_b}| = w$. The distance between the key front K_f and the lens center is d. By geometry operations, we have

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

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

$$l' = \frac{fl}{d}. \quad (2)$$

From Equations (1) and (2), 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 the key in an image.

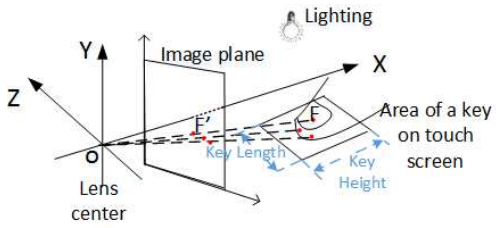


Fig. 12. Fingertip Projection

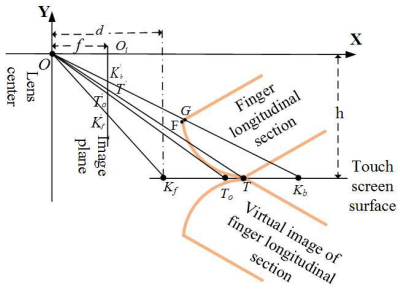


Fig. 13. Touched Point inside a Key

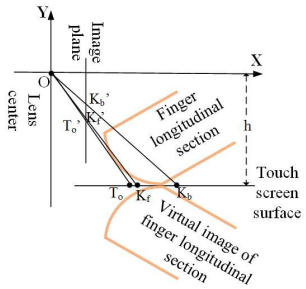


Fig. 14. Touched Point outside a Key

Table 3 gives the camera specifications of the cameras used in our experiments: Logitech HD Pro Webcam C920 [27], 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 Equation (1) and our experiments, the key height is only a few pixels. Therefore, in our experiments, we often need to zoom the fingertip image for the 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.

3.2 Clustering-based Recognition of Touched Points

Based on the model of the touching finger in an image, we now introduce the clustering-based strategy to recognize 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[1]. 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.

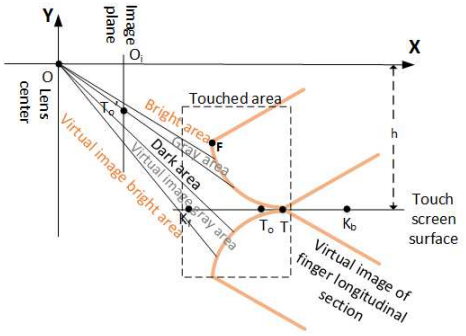


Fig. 15. Five Pixel Groups at Fingertip

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 reflection conforms to the Lambert's Cosine Law [12]: 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 portion 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 in order to *accurately* identify the touched point. 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: (i) The coordinate of a pixel p is (x, y), where x is the column number and y is the row number. Therefore,

$$\mathcal{S} = \{p | p \in \mathcal{C},\ p.y < \text{median of } y \text{ of all pixels in } \mathcal{C}\}. \quad (3)$$

(ii) We 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 3.1, 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 $\mathcal{K}$, we choose $\mathcal{K}_b$ as the real mapped key. Our experiments show that the optimal α is 1/5.

If we examine Figure 13 carefully, we can see that in addition to touched points, points on the fingertip arc may also be projected

1. Touched points actually form an area under the fingertip.

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 onto 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.

4 TEXT ANALYSIS

We now introduce how to conduct context-sensitive analysis using n-gram language model given a reconstructed sentence by the computer vision based attack introduced above.

4.1 N-gram Language Model

N-gram language models (LMs) in NLP produce a probability for a word sequence in the language. The probability of the word sequence indicates how likely the sequence appears in the language. Formally, such a language model is defined as follows [28]: A language model consists of a finite vocabulary set V, and a function $p(w_1, w_2, \ldots, w_n)$ such that: For any $< w_1, \ldots, w_n > \in V\dagger$, where $V\dagger$ is an infinite set of all possible sentences that can be generated using vocabulary V, we have

$$p(w_1, w_2, \ldots, w_n) \geq 0. \tag{4}$$

In addition,

$$\sum_{<w_1, \ldots w_n> \in V\dagger} p(w_1, w_2, \ldots, w_n) = 1, \tag{5}$$

where $p(w_1, w_2, \ldots, w_n)$ is a probability distribution over the sentences in $V\dagger$.

The probability of the word sequence $w_1, w_2, \ldots, w_n$ is derived by

$$p(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_1 \ldots w_{i-1}). \tag{6}$$

Applying the $(N-1)^{th}$ Markov assumption, only the prior $N-1$ words in the history are kept in the conditional probability of Equation (6) and this forms the N-gram Language Model as follows:

$$p(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_{i-N+1} \ldots w_{i-1}), \tag{7}$$

where N usually ranges from 2 to 4.

4.2 Applying N-gram Language Models With Hidden Markov Model (HMM)

Given a reconstructed sentence, we can model the sentence as a state transition system migrating from one state (word) to another (word) along the timeline. The values of the states (the original words) are unknown, and we have observations of the states (observed words). Figure 16 is the Hidden Markov Model [29] for the reconstructed sentence 'shem it ix q lony one you tzkd z cab', with the original input 'when it is a long one you take a cab'. Our goal is to find the most possible underlying state sequence from the observed words as the most possible intended input. In the following, we give the necessary mathematics to capture the essence of using HMM and n-gram model for error correction in our context.

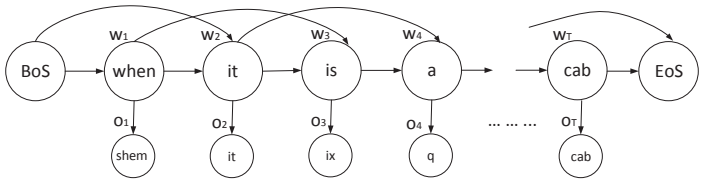


Fig. 16. Trigram Language Model with Hidden Markov Model for a Sentence

Formally, given the reconstructed word sequence $o_1, o_2, \ldots, o_T$, we want to get the state sequence that would most possibly generate the observed word sequence as the intended input. The candidates of the observed words form a directed graph, and all the paths of length T form the candidate sentence set S. We need to find the sentence candidate $\hat{W}$ that maximizes the probability $P(W|O)$, where O is the observed sentence, and W is the sentence candidate in S. $\hat{W} = w_1, \ldots, w_T$ are the derived underlying states. This is formalized into Equation (8),

$$\hat{W} = \underset{W \in S}{\operatorname{argmax}} P(W|O). \tag{8}$$

Applying Bayes rule, we obtain Equation (9),

$$\hat{W} = \underset{W \in S}{\operatorname{argmax}} \frac{P(O|W)P(W)}{P(O)}. \tag{9}$$

Since $P(O)$ is the same for all the candidate sentence W, we just ignore this denominator and obtain Equation (10),

$$\hat{W} = \underset{W \in S}{\operatorname{argmax}} P(O|W)P(W). \tag{10}$$

We set the sentence as the word sequence $W = w_1 w_2 \ldots w_T$, $O = o_1 o_2 \ldots o_T$. The output independence assumption states that the probability of an output observation o_i produced by state w_i depends only on the state itself, not on any other states or observations, that is,

$$P(o_i \mid w_1 \ldots w_i \ldots w_T, o_1 \ldots o_i \ldots o_T) = P(o_i \mid w_i). \tag{11}$$

To derive $P(O \mid W)$, we apply the chain rule and the output

independence assumption, and obtain Equation (19),

$$P(O \mid W) = P(o_1 \dots o_T \mid w_1 \dots w_T), \quad (12)$$
$$= \frac{P(o_1 \dots o_T, w_1 \dots w_T)}{P(w_1 \dots w_T)}, \quad (13)$$
$$= P(o_T \mid o_1 \dots o_{T-1}, w_1 \dots w_T) P(o_1 \dots o_{T-1}, w_1 \dots w_T)/P(w_1 \dots w_T), \quad (14)$$
$$= P(o_T \mid o_1 \dots o_{T-1}, w_1 \dots w_T) P(o_{T-1} \mid o_1 \dots o_{T-2}, w_1 \dots w_T) P(o_1 \dots o_{T-2}, w_1 \dots w_T)/P(w_1 \dots w_T), \quad (15)$$
$$= P(o_T \mid o_1 \dots o_{T-1}, w_1 \dots w_T) P(o_{T-1} \mid o_1 \dots o_{T-2}, w_1 \dots w_T) \dots P(o_1, w_1 \dots w_T)/P(w_1 \dots w_T), \quad (16)$$
$$= P(o_T \mid o_1 \dots o_{T-1}, w_1 \dots w_T) P(o_{T-1} \mid o_1 \dots o_{T-2}, w_1 \dots w_T) \dots P(o_1 \mid w_1 \dots w_T), \quad (17)$$
$$= \prod_{i=1}^{T} P(o_i \mid o_1 \dots o_{i-1}, w_1 \dots w_T), \quad (18)$$
$$= \prod_{i=1}^{T} P(o_i \mid w_i). \quad (19)$$

In [8], we introduced how to compute $P(o_i \mid w_i)$ in Equation (19). The basic idea is as follows. The computer vision based attack may not be able to reconstruct the exact touched letter. However, we can always derive a few candidates for that letter since we can identify the touched fingertip area, which contains the candidate letters. Therefore, combinations of candidates of all the letters in a reconstructed word form the word candidates. By computing distance between each word candidate to the reconstructed word, we can estimate $P(o_i \mid w_i)$.

In this paper, we use the trigram language model. Therefore, to derive $P(W)$, we apply the chain rule and the second order Markov assumption that the probability of a particular state depends on only two previous states, and obtain Equation (23)

$$P(W) = P(w_1, w_2 \dots w_T), \quad (20)$$
$$= P(w_1)P(w_2 \mid w_1) \dots P(w_T \mid w_1 \dots w_{T-1}), \quad (21)$$
$$= P(w_1) \prod_{i=2}^{T} p(w_i \mid w_1 \dots w_{i-1}), \quad (22)$$
$$= \prod_{i=1}^{T} P(w_i \mid w_{i-2}, w_{i-1}). \quad (23)$$

To model the sentence by the hidden Markov model in Figure 16, we need to add the start (BoS) and end states (EOS) to the sentence. That is, $w_{T+1} = o_{T+1} = EoS$, $w_0 = w_{-1} = BoS$. We assume that the EoS of the sentence can be correctly detected, that is, $P(EoS \mid EoS) = 1$. Substitute Equations (19) and (23) into Equation (10), we obtain Equation (24),

$$\hat{W} = \underset{W \in S}{\arg\max} \prod_{i=1}^{T+1} p(w_i \mid w_{i-2}, w_{i-1}) p(o_i \mid w_i). \quad (24)$$

To avoid computing all the possible combinations with a complexity of exponential $O(\mid Q \mid^T)$, where T is the length of the sentence and $\mid Q \mid$ is the number of possible candidates for each word, we apply the Viterbi algorithm [30], which is a dynamic programming algorithm with computation complexity $O(TM^2)$, where T is the length of the sentence and M is the number of possible word candidates for each word w_i. These M candidates have the largest probabilities among all the word candidates for the word w_i. In this paper, we set M as 15, and find a larger M does not significantly improve the result.

We use the Maximum Likelihood Estimation(MLE) to evaluate the probability of a given trigram. MLE is a generic technique to estimate such probabilities. MLE is defined as follows:

$$p(w \mid u, v) = \frac{c(u, v, w)}{c(u, v)}, \quad (25)$$

where $c(x)$ is the count of the appearances of the tuple x in the training corpus. We use the British National Corpus [31] to train the language model.

In practice, many of the trigram counts will be 0 due to the data sparsity. Given that any training corpus is limited and correct trigrams may not appear in the corpus, some trigrams will have probability 0. To solve this problem, we deploy the discounting and back-off smoothing technique, using the bigram or unigram model to derive the trigram probability [32]:

$$p(w_i \mid w_{i-k+1}^{i-1}) = \begin{cases} \dfrac{count(w_{i-k+1}^{i})}{count(w_{i-k+1}^{i-1})}, & \text{if } count(w_{i-k+1}^{i}) > 0, \\ \alpha p(w_i \mid w_{i-k+2}^{i-1}), & \text{otherwise,} \end{cases} \quad (26)$$

where

$$p(w_i) = \frac{count(w_i)}{N}. \quad (27)$$

Here, α is set to be 0.4, $count(x)$ is the number of appearances of the tuple x in the corpus, and N is the number of words in the training corpus.

5 Evaluation

In this section, we present the experiment design and results to demonstrate the impact of the blind recognition of touch inputs, including both random passcodes and meaningful text. We first introduce experimental design and results of recognizing passcodes and then show the results of text recognition.

5.1 Experiment Design

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

Users: Different people have different finger shapes, fingernails and touching gestures. Five females and six males with the experience of using tablets and smartphones participated in the

experiments. They were divided into three groups: 3 people in the first group, and 7 people in the second group. These two groups performed experiments with iPad. The 3rd group evaluated 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 scenarios in our experiments.

5.2 Detecting Touching Frames via Optical Flow

As discussed in Section 2.4, 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 Approach (AA), introduced in Section 2. 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 3, 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. Thus, 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 defined as 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 relatively 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 should help. However, our threat model does not allow the use of those high zoom cameras.

To test whether humans 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 obtain the whole 4-digit passcode right. And, it is almost impossible for humans to recognize keys on a QWERTY keyboard given so many keys and small key size in a video.

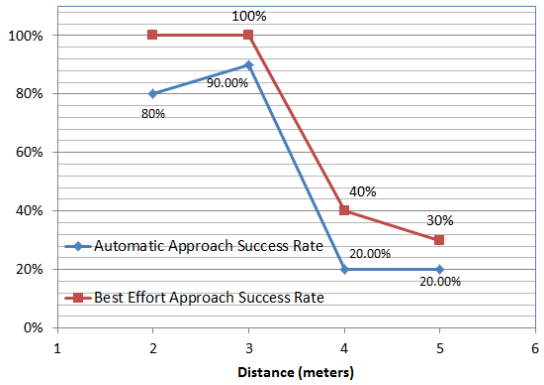


Fig. 17. Success Rate v.s. Distance

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 two meters away from and about 0.65m above the device. To investigate the impact of different cameras, we conducted 30 experiments using the iPhone 5 to record passcode inputs on an 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. 30 experiments with the Samsung Galaxy Gear Smartwatch recording password input on iPad were performed from about two meters away and the attacker sitting in front of the victim with the smartwatch on the right wrist. Figure 18 presents the results. We can observe that 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 investigated attack.

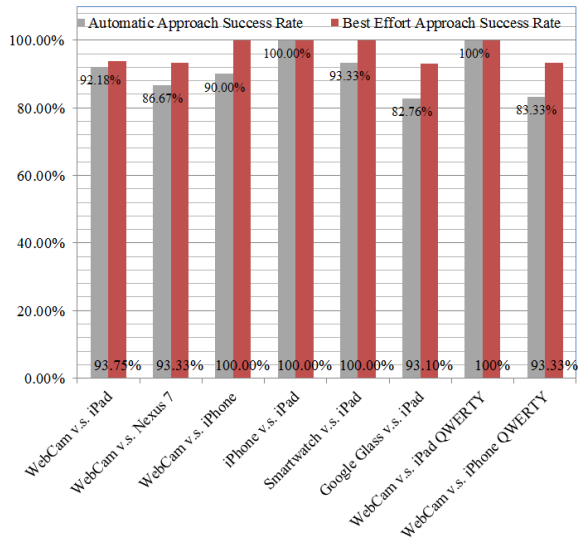


Fig. 18. Success Rate Comparison

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 conducted 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%!

To validate that our attack can be deployed remotely, we carried out 30 experiments with the camcorder. The attacker was on the third floor of a building and the victim is over 40 meters away on the ground. We have a 100% success rate.

5.5 Text Analysis

We have demonstrated the threats from our computer vision based attack against passwords entered by more than 10 people. We now demonstrate the threats of the attack against meaningful messages. Since the basic idea of the attacks against passwords and messages are the same, we only employ three persons in this set of experiments. iPhone was used as the attack camera to spy on the iPad and 12 messages (181 words) were selected from Wall Street Journal, email messages and the example sentences from [7] as intended sentences, as shown in Table 7.

In our experiments, the reconstruction process can recover the messages perfectly from 3 or 4 meters. As the distance reaches 5 meters, the reconstruction makes errors and can correctly recover 48.62% of the words. By applying NLP techniques, we can correctly recover 94.45% of the words with the unigram language model and 96.13% of the words with the HMM based trigram language model. Table 7 gives example results. The bold words in the last two columns refer to words that are not correctly recognized. From Table 7, it is clear that some obvious errors by the unigram language model can be corrected by the trigram language model although the trigram language model is not perfect either.

6 Privacy Enhancing Keyboard

As a countermeasure, we introduce the context aware Privacy Enhancing Keyboard (PEK) which randomizes the keyboard layout during the password/passcode input, but uses the QWERTY keyboard for other inputs. We implemented the first ever Android PEK, which has been downloaded 1600+ times on Google Play. PEK's design and implementation are discussed in [9]. In this paper, we introduce an alternative to PEK, a randomized lock screen keypad app, which is favorable by users who do not want to use a third party keyboard. The performance of the unlock screen keypad is similar to PEK, as reported in [9]. However, we design our unlock screen keypad in such a way that other authenitcation schemes such as novel graphical passwords can be implemented by just changing a couple of classes.

6.1 Overview

The unlock app overrides the Android lock screen to provide a custom layer of authentication. The app consists of two major interface components: a settings screen and a lock screen. The former is displayed whenever the user opens the app from the launch screen to change the password, or to enabled/disable the app. If enabled, the lock screen will be displayed whenever the phone is woken up.

Figure 19 illustrates the overall architecture of the lock screen app. It has three layers. In the layer of *Custom Approach*, a developer just need to edit the two classes *DefaultUnlocker* and *DefaultSettingsActivity* to implement their own authentication strategy while we implement the randomized keypad in this paper. *DefaultUnlocker* is the lock screen and *DefaultSettingsActivity* is

TABLE 7
Example Recognition Results

Intended Sentence	Reconstruction	Unigram	Trigram
when your round is a short one you take a walk	whdn your rounc ix z wnpft ljd you tzkd z szl,	when your round is a short one you take a walk	when your round is a short one you take a walk
when it is a long one you take a cab	shem it ix q lony one you tzkd z cab	when it is a long one you take a **can**	when it is a long one you take a cab
if you know your enemy and you know yourself you need not fear the results of a hundred battles	if uou ojkw your emeky amd yoi kmow yourself uou need not fear the redultd ot a jundred battles	**of** you **like** your enemy and you know yourself you need not fear the results of a hundred battles	if you **like** your enemy and you know yourself you need not fear the results of a hundred battles
if you know neither the enemy nor yourself you will succumb in every battle	if you knoa heither thd ensmy nor ykurself you aill xuccumb in dgery battle	**of** you know neither the enemy **not** yourself you will succumb in every battle	if you know **brother** the enemy **not** yourself you will succumb in every battle
i plan to stay at home	i p.an to ztay zt home	i plan to stay at home	i plan to stay at home
i am busy tonight	k zm buxy tonight	i **an** busy tonight	i am busy tonight

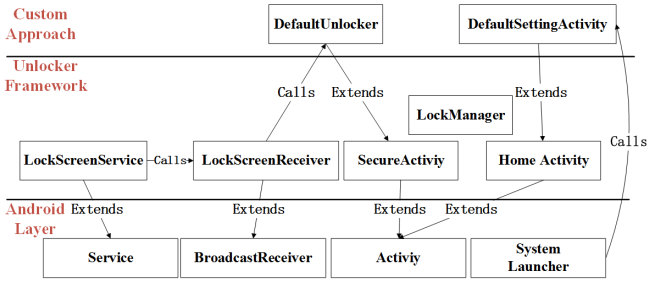


Fig. 19. Architecture of the Lock Screen

the settings activity shown when the app is launched. The layer of *Unlocker Framework* takes care of Android programming specifics of implementing a secure lock screen app. *SecureActivity* contains a set of mechanisms to prevent an attacker from bypassing the lock screen by for example, pressing the *Home* button. Using *LockManager*, the phone can also be natively locked for further security. The unlock app registers an Android service *LockScreenService* to lock the screen. The *Android Layer* lists Android classes that the app extends.

6.2 Immersive Lock Screen

The lock screen displayed by our app is actually overlaid on the native Android lock screen. It is necessary for the application to be fully immersive and block system components of the native lock screen. A new feature in Android API level 19 is the immersive system UI flag, View.SYSTEM_UI_FLAG_IMMERSIVE. Along with other system flags, the navigation bar and status bar can be hidden. The navigation bar is especially important to hide, as it provides user capability to exit the app. While other security measures are taken to avoid such an exit, it makes a cleaner user-interface to hide the navigation bar and to properly imply that navigation is not allowed within the lock screen. The system flags including View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION, View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN, View.SYSTEM_UI_FLAG_HIDE_NAVIGATION, View.SYSTEM_UI_FLAG_FULLSCREEN, and View.SYSTEM_UI_FLAG_IMMERSIVE can be applied to the window's decor view. The *Home* button may be used to bypass the immersive lock screen. We address this issue by registering the app as the default device launcher to intercept all presses of the home button.

6.3 Administrator Privileges

Our lock screen application runs with an immersive interface so that system UI elements are hidden. This configuration, however, is not sufficient means for securing the phone. It is still possible to open the system drop-down pane and gain access to the phone, and there are other ways to bypass a simple interface overlay. It is therefore necessary to lock the phone on the system level.

In order to perform such a lock operation, the application should have administrator privileges. These can be obtained from the user using a prompt screen and a result receiver. The prompt screen is displayed when the user attempts to save their password from the settings screen. The prompt screen can be launched using an intent with the action ACTION_ADD_DEVICE_ADMIN. It then should specify a receiver class, using a *ComponentName* object, to listen for the result of the user's choice. It also should specify an explanation string, which will be displayed on the prompt screen, and whenever the user views a list of current device administrators. On the prompt screen, the user can either choose to grant or not grant the application administrator privileges. Once they have selected their choice, the receiver class is invoked.

The *AdminReceiver* class extends *DeviceAdminReceiver*. The onDisableRequested of this receiver class returns a string to be displayed if the user requests that administrator privileges be revoked. It is then necessary to detect if the application has administrator privileges. If it does, the lock screen can proceed to lock the phone. If not, the lock screen is not secure and will simply terminate. For performing most administrative operations, a *DevicePolicyManager* is used. It can be obtained from the system service DEVICE_POLICY_SERVICE.

6.4 Device Locking

Now probably the most important part of the application, the system lock performed by the lock screen. If the application has administrator privileges, it will set a device password. A *DevicePolicyManager* is used to set the device password's value. In Android, restrictions on passwords are sometimes in place, such as a minimum length or password quality. It is thus necessary to reset these restrictions when setting the password, to ensure that it is not rejected by the system. The quality is set to PASSWORD_QUALITY_UNSPECIFIED, which means any form of string is acceptable, and the minimum length is set to 0. While it could be set to any arbitrary value, the password is set to be the same as the user's password entered through the App. In the event of the App crashing, the user should still be able to access their phone.

Similarly, once the user successfully enters their password on the App's lock screen, the system password is removed to allow entry. The system password can be simply removed by setting its value to a blank string, supplying no additional flags. To then dismiss the Keyguard, the Android system lock screen, the parameter flag FLAG_DISMISS_KEYGUARD can be added to the application window.

7 Related work

Because of the space limit, we only review the most related work. Xu *et al*. extended the work in [6] and tracked the finger movement to infer input text [7]. Their approach consists of six stages: in Stage 1, they use a tracking framework based on AdaBoost [33] 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 the unigram 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 [7] on recognizing passwords, we can achieve a much higher success rate. We also extend our work to the scenario of touching with both hands and multiple fingers while such a scenario is not addressed in [7]. As to recognizing texts, we apply both trigram language model and the Hidden Markov Model to perform the correction.

Shukla *et al*. [34] relaxed the requirement that the video should capture the fingertip movement and focus on the analysis of PINs. Their work assumes that they are able to capture parts of the hand during the PIN tapping process and the back side of the phone. In their investigated attack, to derive when the fingertip touches the screen, they obtain the frame where two feature points on the hand and the device that has the nearest distance. They analyzed the hand movement and the relation between the hand and the device to derive the touched key through a geometry mapping process. However, their work only focused on the PIN retrieving, and no analysis on the QWERTY keyboard was performed. They did not analyze the case where more fingers are used interchangeably during the input process.

8 Conclusion

In this paper, we presented 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 and the reference image of a software keyboard. Our extensive experiments show that the the success rate of the attack is more than 90% against keypad and QWERTY keyboards by various cameras on mobile devices such as Google Glass, smartwatch and smartphone. We also explored the n-gram language models to recognize text inputs on touch-enabled devices, given the input is meaningful English sentences, with grammars and semantics. We then built trigram language models from the British National Corpus and applied them to analyze sentences via the hidden markov model. Our experiment results show that while the basic computer vision based reconstruction technique has a success rate of 48%, the trigram language model improves the success rate to 96%. As a countermeasure, we introduce an alternative to our Privacy Enhancing Keyboard (PEK), a randomized lock screen keypad app. Its software architecture allows easy extension for various authentication strategies for the Android lock screen.

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, 1318948, 1262275, 1350145, 1117175, and 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.

References

[1] 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.

[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 (S&P)*, 2008, pp. 158–169.

[3] 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 (S&P)*, 2009, pp. 315–327.

[4] 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 (S&P)*, 2008, pp. 170–183.

[5] F. Maggi, S. Gasparini, and G. Boracchi, "A fast eavesdropping attack against touchscreens," in *IAS*. IEEE, 2011, pp. 320–325.

[6] 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*, ser. CCS '11, 2011, pp. 527–536.

[7] 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.

[8] Q. Yue, Z. Ling, W. Yu, B. Liu, and X. Fu, "Blind recognition of touched keys on mobile devices," in *Proceedings of the 2015 Workshop on Privacy-Aware Mobile Computing*. ACM, 2015.

[9] 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 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 1403–1414.

[10] 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.

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

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

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

[14] J. Shi and C. Tomasi, "Good features to track," Tech. Rep., 1993.

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

[16] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[17] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Comput. Vis. Image Underst.*, vol. 110, no. 3, pp. 346–359, Jun. 2008.

[18] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *Proceedings of VISAPP International Conference on Computer Vision Theory and Applications*, 2009, pp. 331–340.

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

[20] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, no. 6, pp. 679–698, 1986.

[21] J. Matas, C. Galambos, and J. Kittler, "Robust detection of lines using the progressive probabilistic hough transform," *Comput. Vis. Image Underst.*, vol. 78, no. 1, pp. 119–137, 2000.

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

[23] 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*, ser. CHI '06, 2006, pp. 1263–1272.

[24] 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*, ser. CHI '07, 2007, pp. 647–656.

[25] 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*, ser. UIST '09, 2009, pp. 23–32.

[26] 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*, ser. CHI '09, 2009, pp. 1063–1072.

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

[28] R. C. Berwick. (2012) Natural language processing notes for lectures 2 and 3, fall 2012. [Online]. Available: http://web.mit.edu/6.863/www/fall2012/lectures/lecture2&3-notes12.pdf

[29] L. R. Rabiner and B. H. Juang, "An introduction to hidden markov models," *IEEE ASSp Magazine*, 1986.

[30] A. J. Viterbi, "A personal history of the Viterbi algorithm," *Signal Processing Magazine, IEEE*, vol. 23, pp. 120–142, 2006.

[31] L. Burnard, Ed., *Reference Guide for the British National Corpus*. Research Technologies Service at Oxford University Computing Services, 2007.

[32] T. Brants, A. C. Popat, P. Xu, F. J. Och, J. Dean, and G. Inc, "Large language models in machine translation," in *In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2007, pp. 858–867.

[33] H. Grabner, M. Grabner, and H. Bischof, "Real-time tracking via online boosting," in *Proceedings of the British Machine Vision Conference*, 2006.

[34] D. Shukla, R. Kumar, A. Serwadda, and V. V. Phoha, "Beware, your hands reveal your secrets!" in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. ACM, 2014, pp. 904–917.

Qinggang Yue Qinggang Yue is a PhD student at University of Massachusetts Lowell. He received a master's degree from the Institute of Information Engineering, Chinese Academy of Sciences. His research interests are mobile security and privacy. He gave a talk at Black Hat USA 2014. He loves hiking.

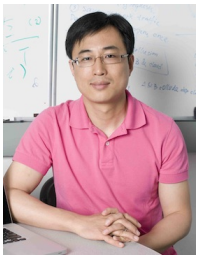
Zhen Ling is a lecturer at the School of Computer Science and Engineering at the Southeast University, Nanjing, China. He received the BS degree (2005) and Ph.D degree (2014) in Computer Science from Nanjing Institute of Technology, China and Southeast University, China, respectively. He also studied and worked at the City University of Hong Kong and the University of Victoria during his Ph.D study. His research interests include network security, privacy, and forensics.

Anthony Topper is a high school student at the Advanced Math and Science Academy in Massachusetts. He has been writing software ever since he was 12 years old, when he first published an iOS app to the Apple App Store. Ever since, he has worked on projects, including websites, mobile apps, and backend solutions. He also enjoys music, especially playing piano and composing pieces of music.

Wei Yu received the B.S. degree in electrical engineering from Nanjing University of Technology, Nanjing, China, in 1992, the M.S. degree in electrical engineering from Tongji University, Shanghai, China in 1995, and the Ph.D. degree in computer engineering from Texas A&M University in 2008. He is currently an associate professor with the Department of Computer and Information Sciences, Towson University. He received U.S. National Science Foundation (NSF) CAREER Award in 2014 and the University System of Maryland (USM) Regents Faculty Award for Excellence in Scholarship, Research, or Creative Activity in 2015. His research interests include cyber security, computer networks, and cyber-physical systems.

Xinwen Fu is an associate professor in the Department of Computer Science, University of Massachusetts Lowell. He received B.S. (1995) and M.S. (1998) in Electrical Engineering from Xi'an Jiaotong University, China and University of Science and Technology of China respectively. He obtained Ph.D. (2005) in Computer Engineering from Texas A&M University. Dr. Fu's current research interests are in network security and privacy, network forensics, and computer forensics. He received the Best Paper Award from IEEE ICC 2008, 2013 and WASA 2013. His research was featured on national and international media including CNN and CCTV.

Benyuan Liu has been a faculty member in the Department of Computer Science at the University of Massachusetts Lowell since Fall, 2004. He received his Ph.D. degree at the University of Massachusetts Amherst. Prior to that, he received his B.S. degree in physics from University of Science and Technology of China (USTC) and M.S. degree in physics from Yale University. Dr. Liu's main research interests are in the area of algorithm design and performance analysis of wireless networks. His research has been published in premium networking conferences such as ACM Mobicom. He won the NSF CAREER award in 2010.



Kui Ren is currently an associate professor of computer science and engineering at State University of New York at Buffalo. He received his PhD degree from Worcester Polytechnic Institute and BE and ME degrees from Zhejiang University. Kui's research interests include Cloud Security, Wireless Security, Internet of Things, and Mobile Crowdsourcing Systems. His research has been supported by NSF, DoE, AFRL, Amazon, and MSR. He is a recipient of NSF CAREER Award in 2011 and Sigma Xi/IIT Faculty Research Excellence Award in 2012. Kui also received the Best Paper Award from IEEE ICNP 2011. Kui serves as an associate editor for a number of journals including IEEE Transactions on Information Forensics and Security. Kui is also an Area TPC Chair for IEEE INFOCOM 2015 and a Distinguished Lecturer of IEEE VTS. Kui is a senior member of IEEE and a member of ACM.

Wei Zhao is currently the rector of the University of Macau, China. Before joining the University of Macau, he served as the dean of the School of Science, Rensselaer Polytechnic Institute. Between 2005 and 2006, he served as the director for the Division of Computer and Network Systems in theUSNational Science Foundation when he was on leave from Texas A&M University, where he served as senior associate vice president for research and professor of computer science. As an elected IEEE fellow, he has made significant contributions in distributed computing, real-time systems, computer networks, and cyber space security.