The Peeping Eye in the Sky

Qinggang Yue*, Zupei Li*, Chao Gao*, Wei Yu†, Xinwen Fu‡, and Wei Zhao§
*University of Massachusetts Lowell, Email: {qye, zli1,cgao}@cs.uml.edu
†Towson University, Email: wyu@towson.edu
†University of Central Florida, Email: xinwenfu@ucf.edu
‡American University of Sharjah, Email: wzhao@aus.edu

***Abstract*—In this paper, we investigate the threat of drones equipped with recording devices, which capture videos of individuals typing on their mobile devices and extract the touch input such as passcodes from the videos. Deploying this kind of attack from the air is significantly challenging because of camera vibration and movement caused by drone dynamics and the wind. Our algorithms can estimate the motion trajectory of the touching finger, and derive the typing pattern and then touch inputs. Our experiments show that we can achieve a high success rate against both tablets and smartphones with a DJI Phantom drone from a long distance. A $2.5''$ NEUTRON mini drone flies outside a window and also achieves a high success rate against tablets behind the window. To the best of our knowledge, we are the first to systematically study drones revealing user inputs on mobile devices and use the finger motion trajectory alone to recover passcodes typed on mobile devices.**

***Index Terms*—Drone, Security, Privacy, Computer Vision**

I. Introduction

Drones, or Unmanned Aerial Vehicles (UAVs), have primarily been used in military applications until recent commercialization. According to Business Insider, estimated 10 million new consumer drones were shipped in 2017. DJI has been dominating the drone market today. The number of mini drone being sold worldwide demonstrates their increasing popularity.

A powerful technology such as drones often has two ethical sides, split between altruistic intention and subversive potential. People use tablets or smartphones doing all kinds of things such as mobile banking and online shopping outside. Civil drones like DJI drones often have at least four degrees of freedom: moving forward and backward, moving left and right, moving up and down, and rotating left and right (yawing). Equipped with cameras, these drones may easily take videos of people performing touch-inputs and inputting sensitive credentials. Given millions of civil drones flying in the sky, it is necessary and critical to perform a rigid study of such privacy threats from these drones.

In this paper, we perform a systematic study of privacy threats from drones equipped with video recording devices such as lightweight camcorders. In particular, a drone captures videos of individuals typing on their smart devices including tablets and smartphones. The text on the touch screen does not appear in the video because of the distance, lighting, camera angle, and limited camera focal length. Figures 1, 2 and 3 show three example experiment scenarios with mini drones. We apply various computer vision techniques to estimate the fingertip motion trajectory over a reference keyboard. Touched points are where the fingertip touches the touch screen to input. We designed sophisticated recursive moving speed analysis (Section III) to derive touched points from the estimated fingertip motion trajectory. Our algorithms consider various cases of touch inputs, such as repeated keys. By connecting the estimated touched points, we are able to estimate the typing pattern, denoted as *target pattern*, which may be distorted and have a translational displacement from the actual typing pattern formed by the actual touched points. We then use the vector similarity distance and turning function distance to measure the similarity of the target pattern and candidate typing patterns derived from combinations of possible touched keys. We then rank candidate passcodes and choose the ones with high similarity scores as the touched keys.

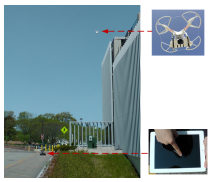

Fig. 1: DJI Drone against iPad

Fig. 2: DJI Drone against iPhone

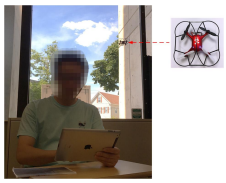

Fig. 3: NEUTRON outside a Window

The major contributions of this paper can be summarized as follows. The key idea of the drone attack in this paper is to estimate the typing pattern from the touching finger's motion trajectory derived from a captured video. The estimated typing pattern can then be used to derive the touch input through our algorithms. To the best of our knowledge, we are the first to use the finger motion trajectory *alone* to derive the passcodes, which can be mobile banking passwords or online shopping passwords, although we use the touch screen unlock pins as the example in the paper. We are also the first to systematically study drones revealing user inputs on mobile devices.

To validate the drone attack, we tested two types of drones: a DJI Phantom 2 drone with a Sony camcorder and a NEUTRON mini drone with a built-in camera. Overall, we recruited 14 volunteers for the experiments. At approximately 15 meters, the DJI drone has a success rate of 100% recovering a 4-digit passcode against iPad. The success rate is 85% even if the distance is 20 meters. We also perform experiments with the DJI drone against smaller mobile devices such as iPhone from 10 meters away and achieve a success rate of over 90%. A $2.5''$ NEUTRON drone were flown outside a window to attack

an iPad behind the window from about 3 meters. It achieves a success rate of over 86%. These experiment results clearly demonstrate the severity of the drone privacy issue.

The rest of this paper is organized as follows. In Section II, we introduce the threat model and the drone attack, in which a drone equipped with a camcorder records a video of an individual typing on her smart device and an adversary can analyze the video to derive the individual's touch inputs. In Section III, we present the fingertip motion analysis in detail. The fingertip motion analysis plays a critical role of identifying touched points. In Section IV, we evaluate the performance of the drone attack. We review related work in Section V and conclude the paper in Section VI.

II. Drone Attack

In this section, we first present the threat model and the basic idea of our drone attack. We then introduce the attack process in detail.

A. Threat Model

In the investigated attack, we assume the drone is equipped with a camcorder, and flies as high as possible to stealthily take a video of a user tapping on the touch screen of her mobile device. We assume that the displayed text on the touch screen does not appear in the video. This is a reasonable assumption, because in a video taken far away, the text on the touch screen may not be visible due to the distance, lighting, camera angle, and limited camera focal length. This is also what we observe in the experiments. We also assume that the adversary has the ability to adjust the height and angle of the drone and camera to capture the device's screen and the fingertip movement in the video. The Sony camcorder in our experiments can be controlled via WiFi by smartphones or tablets with the control app. Therefore, the adversary can see the real-time video with the app and adjust the drone to capture videos with the expected quality.

B. Attack Process

Figure 4 shows the workflow of the investigated attack. We use the four-digit passcode on an iPad as an example and explain these eight steps in detail below.

1) Step 1: Capturing Videos: Compared with video recording on the ground, a drone has the unique viewpoint recording videos from the air. A civil drone often has the four degrees of freedom: moving forward and backward, moving left and right, moving up and down, and rotating left and right (yawing). The adversary can choose an angle for a good view of the target and adjust the height of the drone for stealthy video recording. A civil drone may also rotate forward and backward (pitch) or rotate side to side (roll). Pitching and rolling are the other two degrees of freedom. However, performing pitching and rolling often requires significant training. The stunts of pitching and rolling by civil drones are often performed with high velocity and are not useful for taking quality videos in our context.

Hovering in the sky also introduces the camera instability issue. Motors make a drone and the camera attached to it vibrate. Wind may push the drone out of the view of the target. For a quadcopter, the position-hold function tries to drag the drone back to the original position, but the thrust applied to the drone will tilt the drone and camera. The target can be lost because of such tilting. The vibration may also incur the camera focusing issue and generate blurry videos. In our experiments, when the wind speed is less than 3 miles per hour (mph), we can easily keep the target within the camera view and have expected quality videos. When the wind speed is more than 10mph, it will be very hard to maneuver the drone successfully. When the wind speed is between 3mph and 10mph, it requires significant effort to control the drone successfully. In all conditions, if the drone is at a reasonable height, the camera can capture the whole typing process and the video is not very blurry, we can have a good chance to recover the touch input. Please also note that an adversary may choose calm days without much wind for the drone attack.

2) Step 2: Deriving the Keyboard Layout: In this step, we will draw the reference device and software keyboard of the same size as the ones on a victim device. For example, the iPad display has a resolution of 1024×768 pixels. We can physically measure the device, the coordinates and size of the keys and the margins with a vernier caliper, and then draw the device and keyboard layouts. Here we assume we know the make and model of the victim mobile device. The salient features of most mobile devices may disclose such information from recorded videos.

3) Step 3: Tracking the Device: As discussed in Step 1, the drone and the camera vibrate. The motion of the drone and camera moves the position of the victim device in the video even if the victim device is static. In addition, the appearance of the device in the video may change continuously due to the change of camera angle and lighting. Therefore, effectively tracking the target device in the video is essential for the accuracy of further analysis.

We use DPM [1] to detect the device in the first frame of the video and then track it in all subsequent video frames. DPM models the object as a whole with several (often six) deformable parts. To detect iPad, we train iPad's DPM model and apply it to the first frame. In order to train a DPM model, we need to generate positive data (iPad) and negative data (background). To obtain positive data, we take images of the target device from different viewpoints with the drone, and manually label the device with a bounding box. To collect negative data, we use background images in our attack scenarios taken by the drone. To extend the varieties of scenarios, we also use background images from the SUN dataset [2].

For subsequent frames in the video, we will track the device in each video frame with tracking-based algorithms, which reduce computing time compared with detection-based algorithms. Due to the camera's angle and lighting, the appearance of the device in the video may change. Adaptive tracking algorithms can address this issue [3]. In this paper, we apply the *online tracking algorithm based on boosting* [4]. It is a classification-based tracking algorithm that updates the classifier while tracking the object by selecting the most

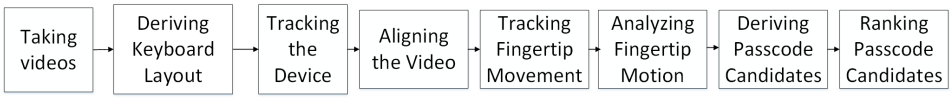

Fig. 4: Workflow of the attack

discriminating features.

4) Step 4: Aligning the Video: In this step, we will stabilize the video by aligning the device in the video to the reference device derived in Step 2. In Step 3, we keep the device area and remove much of the background in the original video. However, each frame may be taken from a different angle and location by the vibrating and moving drone. Even if we can control the drone static in the sky, the victim user may move the device or even walk around on the ground. Therefore, the location of the device in the video would vary in each video frame and it is hard to analyze the movement of the fingertip directly from the raw video. By the alignment performed in this step, we have a common coordinate system for the touch screen surface in *all* video frames.

The alignment is based on the planar homography [5] between the device in video frames and the drawn reference device. Planar homography is a 2D projective transformation that relates two images of the same planar surface. Assume $p = (s, t, 1)$ is a 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 the homography matrix,

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

According to the theory of planar homography [5], we need at least four pairs of corresponding points to obtain the homography matrix. We can use the four corners of the display. These corners are the intersections of the four edge lines. To obtain the edge lines, we first apply the bilateral smoothing filter [6] to sharpen the edges while removing noise in images. After smoothing, we apply the LSD (Line Segment Detection) algorithm [7] to detect the line segments as shown in Figure 5. We now derive the four edges of the display area. Applying the same procedure to the reference device, we obtain the corresponding four points.

The homgoraphy matrix can create a warped image, i.e. aligned image, of the device image in the video frame. Figure 6 shows an example of the warped image blended with the reference keyboard, and demonstrates the relationship between the fingertip and the keyboard.

5) Step 5: Tracking Fingertip Movement: To track the fingertip, we employ the DPM object detection algorithm. We train the DPM fingertip model and apply it to each frame. To obtain the fingertip data for the DPM training, we manually select the fingertip in video frames. The negative data (non fingertip area) is randomly selected from the background. Applying the fingertip model to the device area derived in Step 3, we obtain the detected fingertip in the red bounding box in Figure 7. Since the searching space for the fingertip is limited by the device area, the detecting speed is much faster and the false positive rate is much smaller compared with searching the entire video frame.

To accurately locate the fingertip top, we first train a smaller bounding box around the center of the detected fingertip bounding box. For pixels in this bounding box, the screen area (background) is much darker than the fingertip. Therefore, we can obtain the fingertip contour by clustering pixels in this small bounding box into two groups: the dark screen and the bright fingertip. To derive the *fingertip top*, we first find the central point for each horizontal line of pixels within the fingertip contour. We then fit a line over these central points. This line is the estimated central line of the finger in the video frame, indicating the orientation of the fingertip that touches the screen. The intersection between this line and the fingertip contour is the estimated fingertip top.

Once the fingertip top point is derived for each video frame, we map it to the reference device. By connecting the rectified fingertip points in the reference device image for all video frames, we obtain the fingertip motion trajectory as shown in Figure 8.

6) Step 6: Analyzing Fingertip Motion: In this step, we analyze the fingertip motion trajectory to estimate touched points, where the fingertip touches the screen. The motion trajectory in Figure 8 shows the fingertip moving on the screen, or hanging over the screen during the touching process. The trajectory can be treated as a sequence of 3-tuples (x_t, y_t, t), where x_t and y_t are coordinates of the fingertip in the reference device image, and t is the frame number indicating the time.

At the moment the fingertip touches the key, the fingertip would stay on the key for a short period of time. Assume the fingertip touches the key at frame s and leaves the key at frame e. That is, there may be a few frames at which the fingertip touches the key of interest. These few frames form the so-called *touching action*. For robustness, when our algorithm identifies a touching action, we use the center of the fingertip top points retrieved from these few frames to estimate the touched point. Therefore, **a touching action is defined as a 3-tuple (P, s, e), where P is the center of the fingertip points from frame s to frame e.**

We use motion analysis to derive touched points. When a key is touched, the fingertip stays at the key for a short period and its velocity reduces toward zero. Therefore, we can analyze the moving speed of the fingertip to estimate touched points. We denote touched points estimated through the speed analysis as *hovering points*. However, sometimes, the fingertip moves so fast that a touching action identified by our algorithm includes frames where the fingertip moves


Fig. 5: Detected Lines

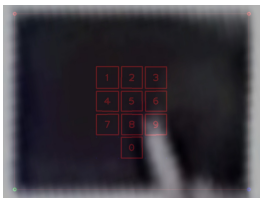
Fig. 6: Warped Image


Fig. 7: Detected Fingertip

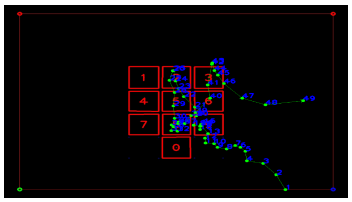
Fig. 8: Fingertip Motion Trajectory

toward the key, and/or frames where the fingertip leaves the key. The fingertip velocity will not be zero in such a case. Moreover, the imperfect computer vision algorithm may not recognize a touched point over a key even if the finger stays still on the key from frame s to frame e. This also introduces a non-zero velocity for the touching action. We will present the detailed analysis in Section III.

7) Step 7: Deriving the Passcode Candidates: In a lucky case, the touched points derived above land in the actually touched areas and the passcode is recovered. However, if the video is of low resolution, we may not be able to derive the touched points directly. The keys holding the touched points are the possible keys, and the keys surrounding the touched points can also be the touched keys. Therefore, we propose an algorithm to generate the key candidates for each touched point. To derive the candidate keys for each estimated touched point, we compute the distance between the estimated touched point and the center of each key. The key with the smaller distance will be the most probable candidate. Given a touched point with coordinate (x_i, y_i), the distance between a key with coordinate (x_k, y_k) and the touched point is defined as the Euclidean distance $dist = \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2}$. For each touched point, we will derive two candidate keys, corresponding to two smallest distances.

Given key candidates for each touched point, we obtain the passcode candidates by combining the key candidates. For a four digit passcode with two key candidates for each touched point, we will have $2^4 = 16$ passcode candidates. Because individuals tend to target the key center when touching keys, connecting the passcode key centers will generate the corresponding *candidate typing pattern* or *candidate pattern* in short.

8) Step 8: Ranking the Passcode Candidates: In this step, we rank the passcode candidates, derived in Step 7, in order to find the most probable ones. Note that the estimated touched points may not be the actual ones, and the key holding the estimated touched point may not be the actual touched key although they should be close to each other. The estimated touched points are critical points on the fingertip motion trajectory. This is because at these points, the fingertip may be performing the touching actions. We connect the four estimated touched points and form the so-called target pattern. Recall that the centers of keys of a candidate passcode derived in Step 7 form the so-called candidate pattern. By measuring the similarity between the target pattern and a candidate pattern, we can rank the passcode candidates so that the one with the highest score will be considered to be the most likely passcode.

We measure the similarity between a candidate pattern and the target pattern to rank passcode candidates. The comparison between the two patterns can be formalized as a polygon similarity estimation problem, which estimates the similarity between the polygons in terms of shape (angle) similarity, scale similarity, location similarity, etc. The classic polygon similarity estimation problem needs to first match the polygon vertices by iterating through all possibilities. The best match is then chosen. Nonetheless, in our case, the polygon is formed by an ordered sequence of keys. The vertex correspondence is already known in Step 7, since we assume that estimated touched points are close to actual touched points. We do not need to iterate over all possible combinations to obtain the vertex correspondence. Since vertices of our two polygons (patterns) refer to the locations of the fingertip on the keyboard and are under the same coordinate system, we do not need to consider the translation, rotation, or scaling.

The scale and location similarities play an important role in our similarity estimation since they are related to the touched keys. The traditional polygon similarity problem requires two polygons have exactly the same shape, while the polygon size and location can be different. For example, the polygon formed by the key sequence 1-2-5-4 and the one formed by the key sequence 5-6-9-8 are similar under the traditional definition, but differ much in our context.

We use the vector similarity distance [8] and the turning function distance [9] as similarity metrics. These two distances measure the difference between two polygons from different perspectives.

Vector Similarity Distance

A polygon with n vertices can be represented by a ordered sequence of vertices, $P(n) = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$. All x-coordinates of P form a vector $\vec{X} = \{x_1, \cdots, x_n\}$ and all y-coordinates of P form a vector $\vec{Y} = \{y_1, \cdots, y_n\}$. Assume we have two polygons P' and P'', $P' = \{(x'_1, y'_1), \cdots, (x'_n, y'_n)\}$ and $P'' = \{(x''_1, y''_1), \cdots, (x''_n, y''_n)\}$. Therefore, $\vec{X'} = \{x'_1, \cdots, x'_n\}$, $\vec{Y'} = \{y'_1, \cdots, y'_n\}$, $\vec{X''} = \{x''_1, \cdots, x''_n\}$, and $\vec{Y''} = \{y''_1, \cdots, y''_n\}$. The vector similarity distance (VSF) $d_{vec}(P', P'')$ is defined as follows

$$d_{vec}(P', P'') = \frac{\vec{X'} \cdot \vec{X''}}{\mid \vec{X'} \mid\mid \vec{X''} \mid} \frac{\vec{Y'} \cdot \vec{Y''}}{\mid \vec{Y'} \mid\mid \vec{Y''} \mid}, \quad (2)$$

where $\mid \vec{X} \mid$ is the norm of the vector $\vec{X}$ and $\vec{X'} \cdot \vec{X''}$ is the dot product of two vectors $\vec{X'}$ and $\vec{X''}$. Cosine similarity measures the similarity of two non-zero vectors as the cosine

of the angle between the two vectors. We can see that vector similarity is the product of cosine similarity of $\vec{X'}$ and $\vec{X''}$ and cosine similarity of $\vec{Y'}$ and $\vec{Y''}$.

Turning Function Distance

The turning function $\Theta(s)$ is a cumulative angle function. The measurement starts with a reference point on an edge of the polygon of interest. s is the total arc length of the polygon starting from this reference point. The initial value of $\Theta(s)$ is the angle between the counterclockwise tangent along the edge at the reference point and the x-axis, where $s = 0$. When we move froward counterclockwise, s increases along the edge of the polygon. Whenever there is an angle change, we update $\Theta(s)$ with the change, which is the counterclockwise angle between the previous tangent (edge) and the next tangent (edge). For a left hand turn with an angle change of δ°, $\Theta(s) += \delta$. For a right turn with an angle change of δ°, $\Theta(s) -= \delta$. When the measurement is finished, s is the perimeter of the polygon.

For two polygons P' and P'' with turning functions $\Theta_{P'}(s)$ and $\Theta_{P''}(s)$, their turning function distance $d_{tf}(P', P'')$ is defined as follows

$$d_{tf}(P', P'') = \| \Theta_{P'}(s) - \Theta_{P''}(s) \|_p, \quad (3)$$

$$= (\int_0^1 | \Theta_{P'}(s) - \Theta_{P''}(s) |^p \, ds)^{\frac{1}{p}}, \quad (4)$$

where $\| \cdot \|_p$ is the L_p norm and p is 2 in our case. For two polygons, their perimeters may be different. Therefore, before computing $d_{tf}(P', P'')$, we normalize the length s by dividing it by the perimeter of each polygon.

In our context, we want to compute the distance/similarity between the target pattern and a candidate pattern, i.e., two polygons. We know the correspondence of vertices on the two polygons. Therefore, we select a pair of corresponding vertices as the reference points and then compute $d_{tf}(P', P'')$.

III. Fingertip Motion Analysis

In this section, we introduce the detailed fingertip motion analysis to derive accurate touched points.

A. Speed Analysis

We first introduce how to compute the speed at an identified fingertip top point. For two fingertip top points $p_i = (x_i, y_i)$, $p_{i+1} = (x_{i+1}, y_{i+1})$, the raw speed VP_i at the point p_i is defined as follows:

$$VP_i = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}. \quad (5)$$

The speed models how fast the fingertip moves from the current point to the next point. Because of the limitation of computer vision algorithms, there exists noise while retrieving the location of the fingertip top. We use the moving average to smooth the speed. The moving average smooths the speed at one point by computing the average of the speed at the point and its neighboring points.

The basic idea of speed analysis is to find low-speed points, which may correspond to a touching action (P, s, e) with the touched point P. To identify points with low speed automatically, we apply the K-means [10] algorithm to cluster the speed into two groups, low and high. We use the median of the low speed group as a threshold to obtain points corresponding to touching actions. Note that one touching action corresponds to a segment of consecutive points.

There is a possibility that within the same touching action, a point may have a large speed in the middle of the touching action because of noise. In such a case, the single touching action is divided into two or more touching actions. To correct this error, we check whether touching actions can be merged based on the time gap between the two actions, the distance between points and the number of frames in the two touching actions. The details of the merging algorithm will be discussed at the end of this subsection.

It is also possible that there may be repeated keys in the passcode. The speed analysis can be still valid for this situation to identify touched points. Our experiments show that since the drone camera always has an angle relative to the direction of touching by the fingertip, the speed of the fingertip is still slow when it touches the key and is relatively fast when it moves towards and leaves the key in a recorded video. Therefore, we may still detect repeated keys with speed analysis. Moreover, the number of frames in each of those touching actions corresponding to repeated keys is large enough to avoid being merged as one touching action.

In some cases, when a person touches different keys, her fingertip speed varies. For example, the fingertip may stay on the last key of a passcode for a longer time. This behavior produces extremely low fingertip speed and affects the K-means algorithm. That is, points with extremely low speed form one group and all other points land in the other group. We may miss other touching actions because of points with the extremely low speed. Therefore, we propose the *recursive speed analysis*: we apply the K-means algorithm to cluster

Algorithm 1 Recursive Speed Analysis

Input:
Raw fingertip motion trajectory $Traj = \{p_1, p_2 \cdots, p_m\}$
k: expected number of touched points
Output: Hovering point set CP_1
for $i = 1$ to $m - 1$ **do**
  $VP_i = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$
end for
$VP_m = 0$
Smoothing VP_i with moving average
$CP_1 = \emptyset$
while $| CP_1 | < k$ **do**
  Clustering the moving average into two groups
  $sThred$ = median of the low cluster
  Filter points with speed larger than $sThred$
  Cluster the filtered results into groups
  Derive the center of points in each group as a touched point
  Determine if newly detected touched points can be merged with the existing detected touched points
  If true, merge newly detected touched points with the corresponding existing touched points and update CP_1. Otherwise, save the touched point into CP_1
end while
return CP_1

points into two groups. If the number of touching actions in the low speed group is less than the expected number of keys in a passcode, we exclude the low speed points and apply the clustering/grouping procedure again to the remaining points until we find the expected number of touching actions. Algorithm 1 gives the recursive algorithm that estimates the touched (hovering) points based on speed analysis.

B. Merging Algorithm

We now discuss how to merge two touched points which are close to each other in Algorithm 1. Given two touched point candidates: $cp_1 = (p_1, s_1, e_1)$ (where point p_1 is the center of touched points from frame s_1 to frame e_1) and $cp_2 = (p_2, s_2, e_2)$ (where the point p_2 is the center of touched points starting frame s_2 to frame e_2), they belong to the same touching action if Equations (6), (7) and (8) hold,

$$dist(p_1, p_2) < DistThred, \quad (6)$$

$$s_2 - e_1 < GapThred, \quad (7)$$

$$e_2 - s_1 < TimeThred, \quad (8)$$

where $DistThred$ is the distance threshold and is defined as the radius of the key. $GapThred$ is defined as the threshold of timing distance between two consecutive touching actions. $TimeThred$ is defined as the threshold of the length of one touching action. $dist(p_1, p_2)$ is the Euclidean distance between points p_1 and p_2. When we merge the two corresponding touching actions, we derive the center of the points in the merged touching actions as the new touched point.

IV. Evaluation

We have performed extensive experiments to evaluate the drone attack introduced in this paper. In this section, we first introduce the metrics and then give the evaluation results.

A. Metrics

To evaluate the effectiveness of the investigated attack and the factors that affect the success of the attack, we use the following metrics and measures to evaluate the attack.

Accuracy: The accuracy is the ratio of the number of successfully recognized passcodes over the total number of passcodes. In an attack, an attacker may try the first passcode candidate or try the top three passcode candidates. For the *first one* strategy, if the first candidate is correct, we consider it to be successful. Otherwise, it is a failure. Under the *top three* strategy, if any of the three passcode candidates is correct, it is a successful attack. This *top three* strategy is reasonable, given that most of the smart mobile devices allow three passcode tries before the device is locked. The account locking policy is similar for other mobile applications such as online shopping and mobile banking.

Human guess by volunteers: During the experiments, we also ask volunteers to guess the passcode from videos recorded by the drone in order to demonstrate the necessity of our computer vision based attack. The vibrating camera makes it impossible for humans to guess the whole passcode right.

Distance effects: To obtain the information about how the height (the distance from the drone to the user) affects the attack accuracy, we fly the drone at different distances from the target with the camera at the same focal length.

B. Results

We now show the evaluation results of the drone attack. To validate the feasibility and effectiveness of the investigated attack, we performed three groups of experiments with the DJI drone from different distances to the user with an iPad 2 tablet, and also performed another group of experiments with iPhone 6s to test how the attack works on smaller devices. In the first group of experiments, we employed 10 people and obtained 30 videos at a distance of approximately 10 meters. We took two more groups of videos with five volunteers and obtained 20 videos with the drone from around 15 meters and 20 meters respectively to test how the height affects the attack accuracy. 5 people were involved in the experiments by the DJI drone attacking the iPhone from around 10 meters, and 15 videos are taken with each person inputting three passcodes. For the NEUTRON mini drone, we involved 5 volunteers tapping the passcodes on iPad and recorded 15 videos.

Table I gives the results of the two passcode selection strategies - (*first one* and *top three*) - for the two ranking algorithms in Step 8 in Section II. It can be observed that the vector similarity distance (VSD) based ranking algorithm works better than the turning function distance (TFD) analysis. The reason can be that normalization used in TFD discards the size difference between polygons so that TFD may incorrectly rank a wrong candidate pattern higher. Moreover, the TFD analysis does not consider the absolute location of a polygon while the absolute location of a polygon is important in our polygon similarity estimation problem.

TABLE I: Accuracy at Around 10 Meters Attacking iPad

	Vector Similarity	Turning Function
First One	100%	53.33%
Top Three	100%	63.33%

Figure 9 shows the accuracy for the two algorithms with the drone at different distances from the target. From the results, we can see the VSD analysis works perfectly even when the distance is 15 meters. At around 20 meters, VSD still has an accuracy of 85% for the *first one* strategy while its accuracy of the *top three* strategy is 87.5%. Compared with the VSD algorithm, the TFD analysis does not work so well as the VSD analysis while the accuracy is not bad.

Figure 9 shows that when the distance increases, the accuracy has a decreasing trend. In the experiments, we set the cameracoder focal length to about one third of its maximum. The resolution of the device at 10 meters is about 133 $\times$ 83, the device size at around 15 meters is about 71 $\times$ 57 and the device at 20 meters is about 57 $\times$ 49. We can see the device size in the video affects the attack accuracy.

Figure 10 gives the summary of the experiment results for different target devices with the VSD ranking algorithm under

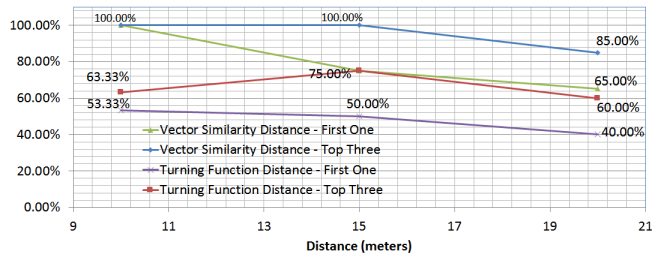

Fig. 9: Accuracy v.s. Distance

different distances and drones. In the case of the DJI drone attacking iPhone, the accuracy is 93%, less than 100% in the case of DJI attacking iPad. The smaller device and keys of iPhone play its role here. Recall in the NEUTRON mini drone attack, the mini drone is outside a building and attacks an iPad behind a window. The distance is around 3 meters and the attack accuracy for the *top three* strategy is over 85%! Figure 10 demonstrates the severity of attacks from various drones on market.

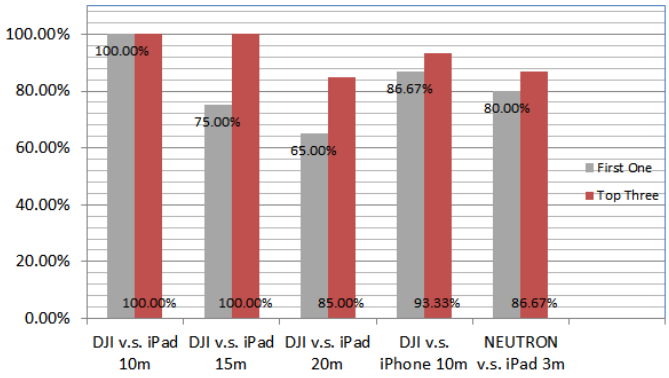

Fig. 10: Accuracy On Different Smart Devices, Drones and Distances

V. Related Work

Because of space limit, we briefly review most related work. In computer vision-based attacks against mobile devices [11]–[15] , an adversary can take videos of people typing on their mobile devices and retrieve the inputs by analyzing the fingertip movement, or the device movement. By applying the Natural Language Processing techniques to correct errors while recovering meaningful messages, an adversary can further increase accuracy. The difference between this paper and these related papers is that previous attacks do not consider the use of the fingertip motion trajectory and typing pattern. We demonstrate that use of such patterns alone can severely breach user privacy.

VI. Conclusion

In this paper, we systematically study the privacy threats from the increasingly popular civil drones. In particular, we investigate the attack in which a drone equipped with a video recording device spies on people inputting sensitive information such as passcodes on the touch screen of their mobile devices. The novel drone attack can recover the passcodes from a recorded video by analyzing the fingertip motion trajectory, even if there is no text visible in the video. Our extensive experiments demonstrate the severity of the privacy issues introduced by drones on market and flying in the key.

Although various privacy enhancing techniques have been proposed to defeat computer vision based attacks, the passcode based authentication with a fixed software keyboard is still the mainstream authentication strategy on various mobile devices. The novel drone attack studied in this paper poses a severe threat against such authentication strategy, given the flexibility of drones and its increasing popularity. More usable and effective mobile authentication strategies are in demand to counter the drone threats.

References

[1] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 32, pp. 1627–1645, 2010.

[2] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba, "Sun database: Large-scale scene recognition from abbey to zoo," in *Proceedings of 2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010, pp. 3485–3492.

[3] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *ACM Computing Surveys*, vol. 38, no. 4, December 2006.

[4] H. Grabner, M. Grabner, and H. Bischof, "Real-time tracking via on-line boosting," in *Proceedings of the British Machine Vision Conference*. BMVA Press, 2006, pp. 6.1–6.10.

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

[6] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proceedings of the Sixth IEEE International Conference on Computer Vision (ICCV)*, 1998.

[7] R. G. von Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall, "Lsd: A fast line segment detector with a false detection control," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 32, no. 4, pp. 722–732, 2010.

[8] D. Cakmakov, V. Arnautovski, and D. Davcev, "A model for polygon similarity estimation," in *Proceedings of Computer Systems and Software Engineering (CompEuro)*, May 1992, pp. 701–705.

[9] E. M. Arkin, L. P. Chew, D. P. Huttenlocher, K. Kedem, and J. S. B. Mitchell, "An efficiently computable metric for comparing polygonal shapes," in *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1990, pp. 129–137.

[10] D. Arthur and S. Vassilvitskii, "k-means++: the advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, 2007, pp. 1027–1035.

[11] Z. Li, Q. Yue, C. Sano, W. Yu, and X. Fu, "3D vision attack against authentication," in *IEEE International Conference on Communications (ICC)*, May 2017.

[12] J. Sun, X. Jin, Y. Chen, J. Zhang, R. Zhang, and Y. Zhang, "VISIBLE: Video-assisted keystroke inference from tablet backside motion," in *Network and Distributed System Security Symposium (NDSS)*, February 2016.

[13] 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 (CCS)*, 2014, pp. 1403–1414.

[14] Q. Yue, Z. Ling, W. Yu, B. Liu, and X. Fu, "Blind recognition of text input on mobile devices via natural language processing," in *Proceedings of the 2015 Workshop on Privacy-Aware Mobile Computing (PAMCO)*, 2015, pp. 19–24.

[15] Q. Yue, Z. Ling, X. Fu, B. Liu, W. Yu, and W. Zhao, "My google glass sees your passwords!" in *Black Hat USA*, 2014.