A Video-side Channel Attack for Pattern Lock

Guixin Ye, Zhanyong Tang, Dingyi Fang, Xiaojiang Chen, Willy Wolff, Adam J. Aviv and Zheng Wang

***Abstract*—Pattern lock is widely used as a mechanism for authentication and authorization on Android devices. This paper presents a novel video-based attack to reconstruct Android lock patterns from video footage filmed using a mobile phone camera. Unlike prior attacks on pattern lock, our approach does not require the video to capture ant content displayed on the screen. Instead, we employ a computer vision algorithm to track the fingertip movements to infer the pattern. Using the geometry information extracted from the tracked fingertip motions, our approach is able to accurately identify a small number of (often one) candidate patterns to be tested by an adversary. We thoroughly evaluated our approach using 120 unique patterns collected from 215 independent users, by applying it to reconstruct patterns from video footage filmed using mobile phone cameras. Experimental results show that our approach can break over 95% of the patterns in five attempts before the device is automatically locked by the Android operating system. We discovered that, in contrast to many people's belief, complex patterns do not offer stronger protection under our attacking scenarios. This is demonstrated by the fact that we are able to break all but one complex patterns as opposed to 60%of the simple patterns in the first attempt. Since our threat model is common in day-to-day life, this paper calls for the community to revisit the risks of using Android pattern lock to protect sensitive information.**

***Index Terms*—Pattern lock, Fingertip movement, Video-based attack, Sensitive information.**

I. INTRODUCTION

Pattern lock is widely used on Android devices to protect sensitive information. It is preferred by some users over PIN- or text-based passwords, as psychology studies show that the human brain remembers and recalls visual information better than numbers and letters [13]. According to a recent study, 40% of the Android users use patterns to protect their devices instead of a PIN [11]. Pattern lock is also used for authentication. For example, `Alipay`, the largest third-party online-payment platform, uses pattern lock for login authentication. Given its pervasive usage, a security breach of the pattern lock could lead to serious consequences.

G. Ye, Z. Tang, D. Fang, and X. Chen are with Northwest University, China
Willy Wolf and Zheng Wang are with Lancaster University, U.K.
Adam J. Aviv is with Naval Academy, U.S.A.

Extension of Conference Paper: a preliminary version of this article entitled "Cracking Android Pattern Lock in Five Attempts" by G. Ye, Z. Tang, D. Fang, X. Chen, K. Kim, B. Taylor and Z. Wang appeared in The Network and Distributed System Security Symposium (NDSS) 2017 [44]. The extended version makes the following several additional contributions over the conference paper: (1) it analyzes the guessing probability of patterns collected from our participants (figure 15 and section V-B); (2) it presents how the screen size and cameras of phones impact on the success rate (table VI, table V, figure 25 and section VI-F); (3) it gives a variant attack method that can crack PIN-base passwords (figure 28, figure 29, figure 30, table VII and section VI-I); (4) it includes new experiment results on relax some requirements to test the effectiveness of our cracking method (figure 31 and section VI-J); (5) it gives a potential countermeasure for defending against video-based attacks (figure 32 and section VII-B); (6) it gives some recent research works on Android pattern lock (section VIII).

Researchers have demonstrated a number of ways to crack Android pattern lock. For example, smudge attacks use the oily residues left on the screen to recover the pattern [3]. However, this approach relies on the persistence of the smudge which can be easily destroyed by subsequent on-screen activities after unlocking. In a recent study, Zhang *et al.* [46] shows that it is possible to infer a locking pattern by analyzing how the WiFi signal is affected by the finger motions when drawing the pattern. Their approach is restricted to a limit set of scenarios due two reasons: (1) the complex setup of the attacking environment and (2) the WiFi signal can be disrupted by any moving objects nearby or body movements.

Recently, video-based side-channel attacks are shown to be effective in reconstructing PIN- or text-based passwords. Some of the early work in this area rely on video footage filmed using a camera directly facing the screen or the keyboard [7, 22]. Latest work shows that this limit can be lifted by exploiting spatial-temporal dynamics of the hands during typing [31]. Despite the success of video-based attacks on PIN- and text-based passwords, no work so far has exploited video-based side-channels to crack pattern lock. To do so, the attack must address several new challenges. These include: How to map the user's fingertip movements to a graphical structure consisting of continuous points instead of discrete keystrokes? How to transform the fingertip movements tracked from the camera's perspective to the user's view point to correctly reconstruct the pattern? How to cancel the camera shake effect that can significantly affect the performance of the attack? How to identify two overlapping line segments of a pattern? The size of the touch-screen or the pattern grid can vary from one device or one application to the other, how can the algorithm adapt to these changes? These new challenges make prior work video-based attacks inapplicable. To overcome these challenges requires creative solutions to be constructed in the new application context of pattern lock.

This paper presents a novel approach to crack Android pattern lock using video footage that captures the user's fingertip motions when drawing the pattern. Unlike smudge attacks [3], our approach does not require the video footage or images to be captured by a camera directly faced the screen. Furthermore, the video can be filmed at a distance of 2 meters from the user in public places. Such a distance is less likely to raise suspicion compared to shoulder surfing [29] that requires a closer observation distance to have a clear sight of the content displayed on the screen.

Our attack employs a computer vision algorithm to track the fingertip motions from the video. Using the geometry information extracted from the fingertip motions, it then maps the tracked fingertip locations to a small number of (often just one) candidate patterns to be tested on the target device.

We thoroughly evaluate our approach using 120 unique

(a) A common attacking scenario.
(b) The device screen seen from the video filmed in (a).
(c) The video was recorded from a distance of 2.5 meters.
(d) The device screen seen from the video filmed in (c).
(e) An outdoor filming scenario.
(f) The device screen seen from the video filmed in (e).

Figure 1. Examples of scenarios in which a mobile phone camera is used to film the unlocking process. In these scenarios, the camera does not need to have a clear sight of the screen.

patterns collected from independent users. We show that our approach is effective in inferring candidate patterns and as a result, an attacker can unlock the target device with a success rate of over 95% (up to 97.5%) in five attempts. We demonstrate that, in contrast to many people's belief, complex patterns do not provide stronger protection over simple patterns under our attack. According to a recent study [25], people tend to use complex patterns for critical applications such as online banking and shopping applications. Our finding suggests that using pattern lock to protect sensitive information could be risky.

Contributions This work is the first attack to reconstruct graphical patterns from video footage without requiring the video to capture any contents displayed on the screen. Our attack exploits techniques developed in the computer vision domain to address the key challenges highlighted above. The key contribution of this paper is a new attack for Android pattern lock that has not seen in prior work. This paper makes the following specific contributions:

- *A New Attack.*This is the first work to reconstruct locking patterns without relying on the content shown on the screen (Section II-B). Experimental results show that our method can break over 95% of the lock patterns in five attempts (Section VI-A). Given that the Android operating system (OS) allows five tries before locking the device, our attack represents a real threat for pattern lock.
- *Identifying New Vulnerabilities.* According to a recent study [12], direct observation techniques, e.g. shoulder surfing, are considered to be a low risk due to the close distance between the attacker and the user (in order to gain a clear sight of the device screen). As a result, many users may underestimate the dangers from using pattern lock in public places. Under our attack, filming can be carried out at a distance of 2 meters from the user and the camera does not need to directly face the target device. Such a camera setting makes our attack less likely to raise suspicion and more likely to success when compared to direct observation techniques. For instance, the video can be filmed by an adversary who pretends to interact with his phone, sitting next to the user in a public place (see Figure 1). In many similar scenarios, users will not be suspicious of the attacker's behavior.
- *New Findings.* Our study suggests that complex patterns are more vulnerable under video-based attacks (Section VI-A). This finding debunks many people's conception that more complex patterns give stronger protection. Therefore, our work sheds new insights on the practical use of pattern lock.

II. Background

A. Android Pattern Lock

Pattern lock is widely used to protect sensitive information and perform authentication on Android touch-screen devices. The motivation for user to prefer to pattern lock is that graphical passwords are easily to be recalled comparing to alphanumeric characters [34, 41]. To unlock a device protected with pattern lock, the user is asked to draw a predefined sequence of connected dots on a pattern grid[1]. Figure 2 (e) shows a pattern which consists of seven dots on a 3×3 grid. To form a pattern, the user starts by selecting one dot as the starting point and then swiping over multiple dots of the grid until the fingertip is lifted from the screen. There are several rules for creating an Android pattern: (1) a pattern must consist of at least four dots; (2) each dot can only be visited once; and (3) a previously unvisited dot will become visited if it is part of a horizontal, vertical or diagonal line segment of the pattern. Taking into account these constraints, the total number of possible patterns on a 3×3 grid is 389,112 [39]. Given the large number of possible patterns, performing brute-force attacks on Android pattern lock is ineffective, because by default the device will be automatically locked after five failed tries. Further, the patterns with complex structures are hard to guess compared to text-based passwords [21, 27].

B. Threat Model

In our threat model, we assume an adversary wants to access some sensitive information from or to install malware on a target device that is protected by pattern lock. This type of attacks is mostly likely to be performed by an attacker who can physically access to the target device for a short period of time

[1]In this paper we use the Android default pattern grid with 3×3 dots, unless otherwise stated.

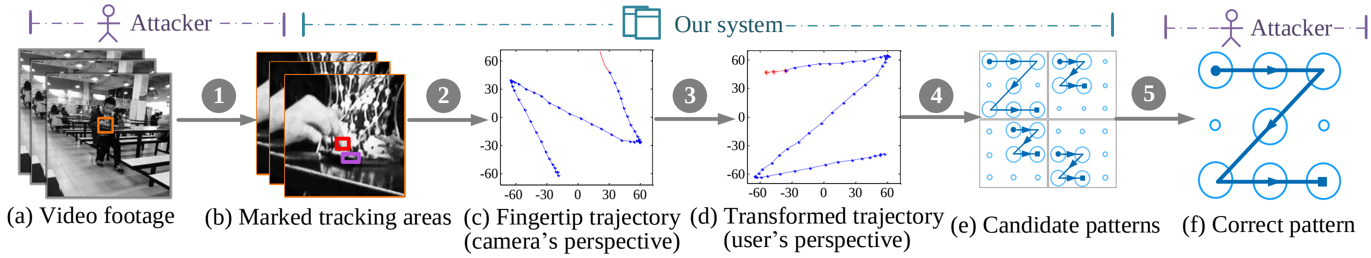


Figure 2. Overview of the attack. Our system takes in a video segment that records the unlocking process (a). The adversary first marks two areas of interest on the first video frame (b): one contains the fingertip involved in pattern drawing, and the other contains part of the device. Our system then tries to track the fingertip's location w.r.t. to the device. The tracking algorithm produces a fingertip movement trajectory from the camera's perspective (c) which is then transformed to the user's perspective (d). Finally, the resulted trajectory in (d) is mapped to several candidate patterns (e) to be tested on the target device (f).

(e.g. via attending a meeting or party where the user presents). To quickly gain access to the device without raising suspicion, the attacker would like to obtain the user's locking pattern in advance. The attack starts from filming how the user unlocks the device. Video recording can be done on-site or ahead of time (probably with the help of someone hired by the attack). The video will then be processed to identify a small number of patterns to be tested on the target device. Because filming can be carried out in from a distance of as far as 2.5 meters using a mobile phone camera and the camera does not need to directly face the target device, this activity often will not be noticed by the user. Moreover, given that many users use the same pattern across devices and applications, the pattern obtained from one device could also be used to break the user's other devices. *The goal of this paper is to demonstrate the feasibility of a new attack and its implications to the use of pattern lock.*

Examples of Filming Scenarios. Figure 1 illustrates three scenarios where filming can be performed without raising suspicion to many users. For all the examples presented in Figure 1, the filming camera had a left- or right-front view angle from the target device and did not directly face the screen of the target device. Due to the filming distance (2-3 meters), the recoded video typically does not have a clear vision of the content displayed on the screen. This observation can be confirmed by the video snapshot placing alongside each scenario, where it is impossible to identify the content shown on the screen. The examples given in Figure 1 are some of the day-to-day scenarios where security of the user's device can be compromised under our attack.

Assumptions. Our attack requires the video footage to have a vision of the user's fingertip involved in pattern drawing and part of the device (e.g. an edge of a phone). We believe this is a reasonable assumption because in practice many users often do not fully cover their fingers and the entire device when drawing a pattern. This is particularly true when holding a large-screen device by hands. To launch the attack, the attacker needs to know the layout of the grid, e.g. whether it is a 3×3 or a 6×6 grid. Our approach is to generate a set of candidate patterns for each of the Android pattern grids and the attacker can simply decide which set of candidate patterns to use after seeing the target device (at the time the layout of the grid will become available). However, unlike prior work on video-based attacks on keystroke based authentication [31], our approach does not require having knowledge of the console's geometry. In other words, the size of the screen or the position of the pattern grid on the screen does not affect the accuracy of our attack. We also assume the video does not need to capture any content displayed on the screen. This assumption makes previous video-based attacks on pattern lock [3] inapplicable.

III. Overview of The Attack

In this section, we give an overview of our attacking system which analyzes the user's fingertip movement to infer the locking pattern. The system takes in a video segment that records the entire unlocking process. It produces a small number of candidate patterns to be tested on the target device. Figure 2 depicts the five steps of our attack:

① **Filming and Video Preprocessing:** The attack begins from filming how the pattern is drawn. The video footage can be filmed at a distance of about 2 meters from the user using a mobile phone camera (or 9 meters using a digital single reflex camera). After recording, the attacker needs to cut out a video segment that contains the entire unlocking process. We have shown that it is possible to automatically identify this video segments in some scenarios (Section IV-A). After cutting out the video segment, the attacker is then asked to mark two areas of interest from a video frame: one area consists of the fingertip used to draw the pattern, and the other consists of part of the device (see Figure 2 (b)).

② **Track Fingertip Locations:** Once the areas of interest are highlighted, a computer vision algorithm will be applied to locate the fingertip from each video frame (Section IV-B2). The algorithm aggregates the successfully tracked fingertip locations to produce a fingertip movement trajectory. This is illustrated in Figure 2 (c). Keep in mind that at this stage the tracked trajectory is presented from the camera's perspective.

③ **Filming Angle Transformation:** This step transforms the tracked fingertip locations from the camera's perspective to the user's. We use an edge detection algorithm to automatically calculate the filming angle which is then used to perform the transformation (Section IV-C). For example, Figure 2 (c) will be transformed to Figure 2 (d) to obtain a fingertip movement trajectory from the user's perspective.

④ **Identify and Rank Candidate Patterns:** In this step, our software automatically maps the tracked fingertip movement trajectory to a number of candidate patterns (Section IV-D).

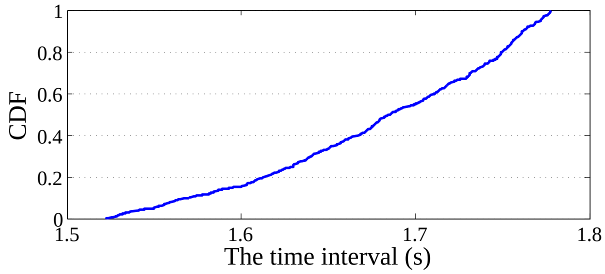
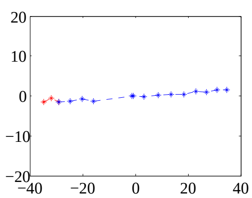
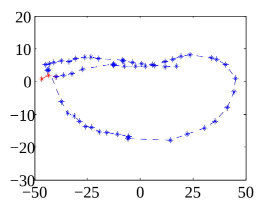


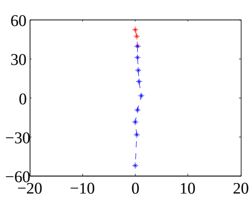
Figure 3. The cumulative distribution function (CDF) of the time interval between pattern drawing and other on-screen activities.



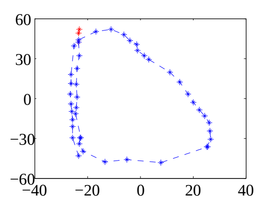
(a) a horizontal-swiping gesture



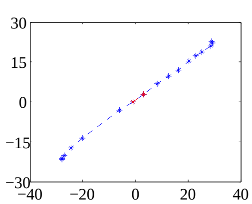
(b) two consecutively horizontal-swiping gestures



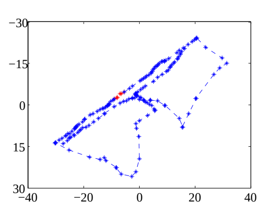
(c) a vertical-swiping gesture



(d) two consecutively vertical-swiping gestures



(e) a zooming gesture



(f)two consecutive zooming gestures

Figure 4. Spatial-temporal characteristics for performing an on-screen gesture once (a, c, e) and twice (b, d, f).

We rank the candidate patterns based on a heuristic described in Section IV-D2. For instance, the fingertip movement trajectory in Figure 2 (d) could be mapped to a number of candidate patterns shown in Figure 11. Our approach is able to reject most patterns to leave no more than five candidate patterns to be tried out on the target device.

⑤ **Test Candidate Patterns:** In this final step, the attacker tests the candidate patterns on the target device.

Algorithm 1 Unlocking process identification heuristic

Input:

IV: Video footage

$frameCount$: Pause threshold before or after unlocking

Output:

<start,end>: Start and end of the unlocking video segment

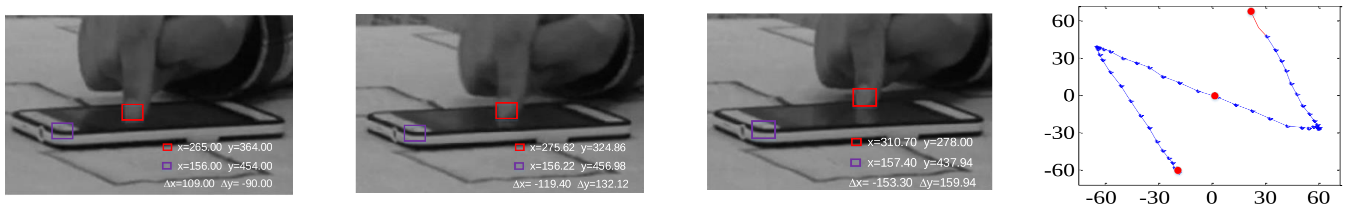
```
frames[] ← getVideoFrames(IV)
LEN ← getFramesLen(frames[])
for i = 1 : LEN − frameCount do
    sL ← hasFingertipChanged(frames[i : i + frameCount])
    if !sL then
        sNo = i + frameCount
        for j = sNo : LEN do
            if checkLoop(frames[j : LEN]) then
                eNo = i
                break;
            else if !hasFingertipChanged(frames[j : j + frameCount]) then
                eNo = i
                break;
            end if
        end for
        break;
    end if
end for
< start, end >← getTargetVideo(frames[], sNo, eNo)
```

IV. Implementation Details

A. Video preprocessing

The first step of our attack is to identify the unlocking process from the video footage. While all our participants (see Section V-A) consider this as a straightforward manual task, we developed a *simple yet effective* heuristic to automatically detect the video segment in some typical scenarios. Our heuristic is based on the following observations: (1) before or after unlocking, users often pause for a few seconds; (2) two consecutive on-screen operations (e.g. swiping, zooming etc.) typically expose some spatial-temporal motion characteristics. To test our hypothesis, we have recorded 50 video streams (where each video stream lasts around 2 minutes) of how ten of our participants drew patterns. During video recording, our participants firstly performed some on-screen activities such as web browsing and gaming as they wished; they then randomly opened up a pattern lock screen to draw a pattern and continued to perform other on-screen operations afterwards. We then analyzed frames that are associated with pattern drawing and those are not.

Figure 3 shows that all our participants paused at least 1.5 seconds before or after pattern drawing due to delay of the user or the device. We also found that identical on-screen activities often follow closely. For example, on several occasions our participants had to swipe several times to locate a program from the application list. These consecutive on-screen operations have some spatial-temporal motion characteristics that are different from pattern drawing. Figure 4 shows the



(a) The first video frame (b) A middle video frame (c) The last video frame (d) Fingertip movement trajectory

Figure 5. Tracking the fingertip movement trajectory. For each video frame, the system tracks two areas: one surrounds the fingertip and the other covers the edge of the device. The fingertip position is determined by computing the relative coordinates of the central points of the two areas. The red points highlighted in the final results (d) are the touching points tracked from the three video frames.

spatial-temporal motion structure for two gestures, swiping and zooming, when they are performed once (a, c, e) and twice (b, d, f). This diagram indicates that the spatial-temporal motion of two identical on-sreen activities contains one or more loop structures for which pattern drawing does not have.

Our heuristic is described in Algorithm 1. The input to the algorithm is a video capturing the unlocking process, and the output of the algorithm is a time-stamp tuple, <start, end>, which marks the start and the end of a video segment. To locate the video segment of pattern drawing, we first filter out on-screen activities where the fingertip location does not change within a timeframe of 1.5 seconds (lines 4 and 11). This allows us to exclude some basic on-screen activities, such as clicking. We use the number of video frames, *frameCount*, as a proxy to estimate the time interval. Here, a time interval of 1.5s translates to 45 frames or 90 frames when the video was shot at 30 or 60 frames per second (FPS) respectively. We also use the spatial-temporal characteristics described above to exclude two consecutive swiping or zooming gestures (line 8). Finally, we exploit the observation that users typically paused at least 1.5s before or after unlocking to locate the start and end points of pattern drawing (line 19).

Limitations Our heuristic is likely to fail if the user was typing using a Swype-like method (i.e. entering words by sliding a finger from the first letter of a word to its last letter) during video recording. In this case, our method will identify multiple video segments of which one may contain the pattern unlock process. If multiple segments are detected, the algorithm will ask the user to confirm which video segment to use. In this scenario, the first identified segment is likely to be the correct one. In practice, an experienced attacker would wait patiently to avoid this complicated situation by finding the right time for filming (e.g. for a screen lock, the time is just after the device is retrieved). The attacker could also watch the video to manually cut it to ensure the obtain the correct video segment. It is worthwhile to mention that automatically identifying the pattern unlocking process is not central to our attack because an experienced attacker can watch the video to manually cut it to ensure the tracking algorithm (described in the section) receives a quality input. Despite its limitations, our algorithm can reduce the efforts involved in some common scenarios.

B. *Track fingertip locations*

After cutting out the video segment of pattern drawing, we need to track the finger motions from the video segment. We achieve this by employing a video tracking algorithm called *Tracking-Learning-Detection (TLD)* [20]. This algorithm automatically detects objects defined by a boundary box. In our case, the objects to be tracked are the user's fingertip and an area of the device. These are supplied to the algorithm by simply highlighting two areas on the first frame of the video segment (see Figure 2 b). The algorithm tries to localize the fingertip from each video frame and aggregates the successfully tracked locations to produce a fingertip movement trajectory as an output (see Figure 2 c).

1) Generate The Fingertip Movement Trajectory: The TLD algorithm automatically detects objects based on the examples seen from the first frame. For each tracked object, the algorithm generates a confidence between 0 and 1. A tracking is considered to be successfully if the confidence is greater than a threshold. In our experiments, this threshold is set to 0.5 since this value is found to give good performance in our initial design experiments using 20 patterns[2]. TLD has three modules: (1) a tracker that follows objects across consecutive frames under the assumption that the frame-to-frame motion is limited and objects are visible; (2) a detector to fully scan each individual frame to localize all appearances of the objects that have been seen in previous frames; and (3) a learner that estimates errors of the detector and updates the detector to avoid these errors in future frames.

The TLD learner automatically extracts features from the area of interest to build a K-Nearest Neighbor classifier [17] which is a part of the detector. In the following frames, the learner estimates the detection errors and generates new training examples (i.e. new appearances of the object) arose from object motion to re-train the classifier to avoid these errors. For each video frame, TLD calculates the tracking confidence and if the confidence is lower than the predefined threshold, the result of this particular frame will be discarded. This allows the algorithm to tolerate a certain degree of detection errors. Finally, the successfully detected object locations will be put onto a single image as the output. Detailed discussion of TLD can be found at [20]. Sometimes the algorithm may fail to

[2]To provide a fair evaluation, the patterns used in our initial test runs in the design phase are different from the ones used later in evaluation.

detect the objects in many video frames due to poor selections of interesting areas. If this happens, our system will ask the user to re-select the areas to track. We have also extended TLD to report when a fingertip position is seen on the footage. This temporal information is recorded as the number of video frames seen so far. This is used to separate two possibly overlapping line segments described in Section IV-D.

2) Camera Shake Calibration: By default, the TLD algorithm reports the position of a tracked object with respect to the top-left pixel of the video frame. However, videos recorded by a hand-held device is not always perfectly steady due to camera shake. As a result, the top-left pixel of a video frame may appear in a different location in later frames. This can drastically affect the precision of fingertip localization, leading to misidentification of patterns.

Our approach to cancel camera shake is to record the fingertip location with respect to a fixed point of the target device. To do so, we track two areas from each video frame. One area is an edge of the device and the other is the fingertip. Both areas are highlighted on the first frame by the user. The location of a successfully tracked fingertip is reported as the the relative coordinates of the two center points of the marked areas. This approach can also be used to calibrate the slight movements of the target device when drawing the pattern.

Example: To illustrate how our camera-shake calibration method works, considering Figure 5 where two areas are firstly marked by two bounding boxes in subfigure (a). Both areas will then be automatically detected by the TLD algorithm in following video frames as shown in subfigures (b) and (c). The coordinates of the two center points of each box are the values of x and y, and their relative positions are represented by $\triangle X$ and $\triangle Y$. For each frame where both areas are successfully tracked, we compute the relative coordinates, $(\triangle X,\ \triangle Y)$, which are reported as the location of the tracked fingertip.

Figure 6 shows the results when using TLD to process a video that was filmed with some camera shake effects. Here we show the results without (Figure 6 a) and with (Figure 6 b) camera-shake calibration. To aid clarity, we have converted the trajectories into the user's perspective. Without camera-shake calibration, the resulted trajectory is significantly different from the actual pattern shown in Figure 6 (c). Because of this great difference, using Figure 6 (a) will lead to misidentification of candidate patterns. By contrast, processing the same video with camera-shake calibration produces Figure 6 (b). Clearly, this figure is more alike the correct pattern.

3) Noisy Points Calibration: During tracking process, the TLD algorithm may report mistaken position of a tracked object as rapid deformation of the tracked object. This can affect the shape of fingertip movement, leading to extract uncorrect geometric information of fingertip movement.

C. Filming angle transformation

In practice, the filming camera will not directly face the target device to avoid the filming activity to be noticed by the user. As a result, the fingertip movement trajectory generated by the tracking algorithm will look differently from the actual pattern. For example, for the pattern presented in Figure 2 (a), if the video is filmed from the attacker's front-left to the target device (i.e. with a filming angle of approximate 45 degrees), we get the trajectory shown in Figure 2 (c). Using this trajectory without any postprocessing will lead to misidentification of candidate patterns. Therefore, we must transform the resulted trajectory to the user's view point. To do so, we need to estimate the angle between the filming camera and the target device. Our approach is described as follows.

We use an edge detection algorithm called Line Segment Detector (LSD) [16] to detect the longer edge of the device. The filming angle is the angle between the detected edge line and a vertical line. This is illustrated in Figure 7. In Section VI-E, we show that a slight estimation error of the filming angle has little impact on the success of the attack. By default, we assume that the pattern grid is presented in the portrait mode[3]. If this is not the case, i.e. the pattern grid is shown in the landscape mode, we need to use the shorter edge of the device to calculate the filming angle. We believe that an attacker interested in a particular target would have some knowledge of how the pattern grid of the target device is presented under different orientation modes and be able to identify the device orientation by watching the video. There are also other methods to be used to identify the filming angle [38].

Based on the estimated filming angle, θ, we use the following formula to transform the tracked fingertip movement trajectory from the camera's view point to the user's:

$$S = TS^{'} \quad , \quad T = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{1}$$

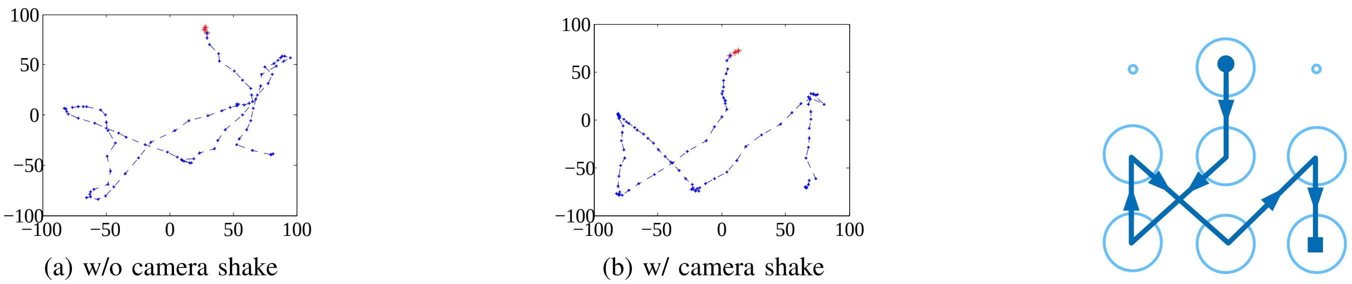
where T is a Transformation Matrix, $S^{'}$ is the coordinate of a point of the tracked trajectory, and S is the resulted coordinate after the transformation. For each video frame, our algorithm individually calculates the filming angle and perform the transformation, because the filming angle may change across video frames.

D. Identify and rank candidate patterns

In this step, the fingertip movement trajectory will be mapped to a number of candidate patterns to be tested on the target device. Our goal in this step is to exclude as many patterns as possible and only leave the most-likely patterns to be tried out on the target device. Our approach is to use the geometry information of the fingertip movement trajectory, i.e. the length and direction of line segments and the number of turning points, to reject patterns that do not satisfy certain criteria. In this section, we first describe how to identify overlapping line segments and extract length and direction information before presenting how to use the extracted information to identify and rank candidate patterns.

1) Extracting Structure Information: A pattern can be defined as a collection of line segments where each line segment has two properties: the length of the line, l, and the direction of the line, d. Figure 10 shows all the possible

[3]The pattern grid of the Android native pattern lock is always presented in the portrait mode regardless of the orientation of the device.



(a) w/o camera shake calibration (b) w/ camera shake calibration (c) correct pattern

Figure 6. The resulted fingertip movement trajectories without (a) and with (b) camera-shake calibration. The correct pattern is shown in (c). To aid clarity we have transformed (a) and (b) to the user's perspective.

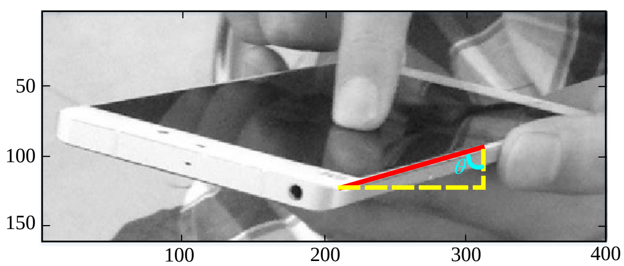
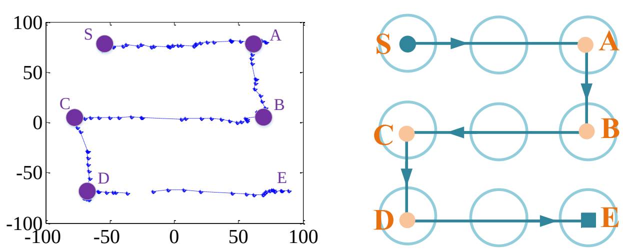


Figure 7. Filming angle calculation. The filming angle, θ, is the angle between the edge line of the device and a vertical line.



(a) tracked fingertips (b) pattern example

Figure 8. This figure shows the tracked fingertip movement trajectory (a) of a pattern (b). Point S on (a) is the starting point and points A, B, C, and D on (b) represent four turning points.

directions of a 3×3 pattern grid. We define a pattern, P, as a collection of line segment prosperities, $P = \{L, D\}$. Here $L = \{l_1, l_2, \cdots, l_n\}$ is a collection of the lengths of all line segments (that are numbered from 1 to n) of the pattern, and $D = \{d_1, d_2, \cdots, d_n\}$ is the collection of directions for all line segments in L. Algorithm 3 describes how P is extracted. We extract the length and the direction of each line segment from the tracked fingertip movement trajectory and store them into arrays $L[]$ and $D[]$ respectively.

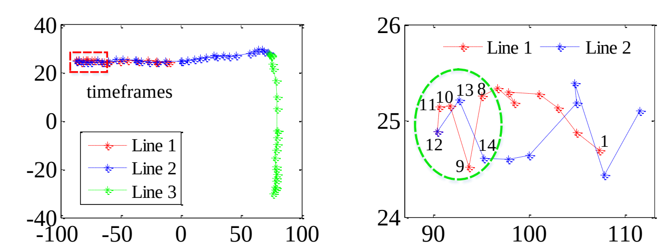
Identify Line Segments. The first step of geometry information extraction is to identify individual line segments from the trajectory. This can be achieved by finding turning points, the start and the end points of the pattern, because two points define a line segment. For example, turning points, A and B, in Figure 8 defines a line segment, AB. In Algorithm 2, we use a linear fitting method [23] to discover turning points (line 3). A specific challenge here is how to separate two overlapping line segments (see Figure 12 c for an example). It is to note

Algorithm 2 Line Segment Identification

Input:
struct $T[]$: Temporal information of each tracked location
$timeTh$: Threshold of whether two line segments are overlapping

Output:
$tp[]$ Turning points of fingertip movement.

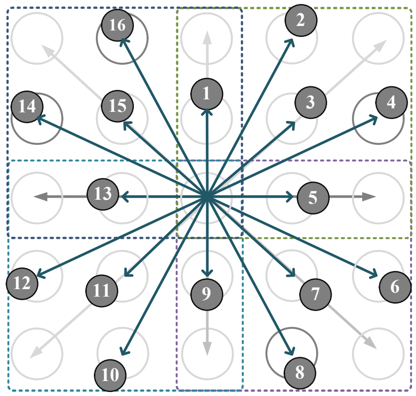
1: **for each** fingertip movement with temporal sequences $T[]$ **do**
2: $tpNum = 0$;
3: **struct** $lines[] \leftarrow getLines(T[])$
4: $lNum \leftarrow getLinesNumber(lines[])$
5: **for** $i = 1 : lNum$ **do**
6: **if** $checkOverlap(lines[i], timeTh)$ **then**
7: $p[tpNum++] \leftarrow getOverlapPoints(line[i])$
8: **end if**
9: $p[tpNum++] \leftarrow getTurningPoints(line[i])$
10: **end for**
11: **end for**
12: $tp[] = p[0 : end - 1]$



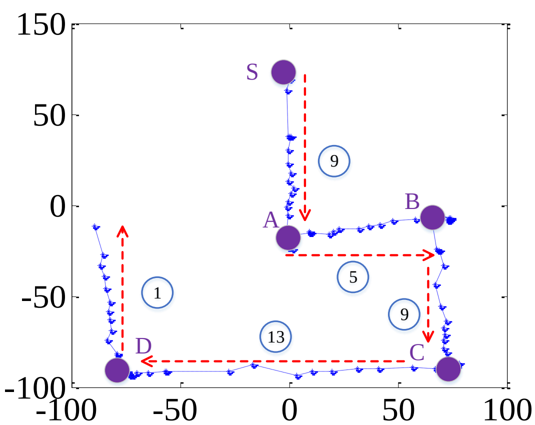
(a) overlapping lines (b) zoom-in view

Figure 9. Separating two overlapping line segments by checking the number of overlapping points within a timeframe.

that up to two lines can be overlapped on a pattern grid. The naive linear fitting algorithm would consider two overlapping segments to be a single line as their points stay close to each other. We overcome this problem by using the temporal information (that is recorded by the tracking algorithm) to separate two overlapping points. To do so, we visit all tracked points of each line segment given by the linear fitting algorithm (line 5) within a timeframe (*timeTh*) of 20 video frames for a video of 30 FPS (40 for a video of 60 FPS). For each point, we calculate its Euclidean distances to all other points within



(a) line direction number



(b) numbering line segment of the tracked trajectory

Figure 10. All possible line directions on a 3×3 Android pattern grid (a) and an example trajectory (b).

Table I

MAPPINGS FROM LINE SLOPES AND FINGERTIP-HORIZONTAL MOVEMENTS TO DIRECTION NUMBERS

Direction No.	1	2	3	4	5	6	7	8
slope (L $\rightarrow$ R)	$+\infty$	2	1	$\frac{1}{2}$	0	$-\frac{1}{2}$	-1	-2
Direction No.	9	10	11	12	13	14	15	16
slope (R $\rightarrow$ L)	$-\infty$	2	1	$\frac{1}{2}$	0	$-\frac{1}{2}$	-1	-2

Algorithm 3 Candidate Pattern Identification Algorithm

Input:

$L[]$: Relative line length
$D[]$: Direction number (see Figure 10)
tn: Number of turning points of fingertip trajectory
$lengthTh$: Threshold of considering two lines to have the same length
$directionTh$: Threshold of considering two lines to be in the same direction

Output:

$P[]$: Candidate patterns

for each possible pattern p with tn turning points **do**
  $n \leftarrow getLineNumber(P[])$
  $pL[] \leftarrow getRelativeLength(p)$
  /*Relatvie line length for pattern p*/
  $pD[] \leftarrow getDirection(p)$
  if match($pL[]$, $L[]$, $lengthTh$) **then**
    if match($pD[]$, $D[]$, $directionTh$) **then**
      $P[] \leftarrow p$
    end if
  end if
end for
$P[] \leftarrow sort(P[])$

the timeframe. We consider two points to be overlapping if their distance is less than 5 pixels. For a video shot at 30 FPS, we consider there exist two overlapping line segments if 5 (10 for a 60 FPS video) or more overlapping points in the timeframe. Again, these threshold values were determined through our initial design experiments. Finally, we consider the center of all points as the turning point of the two overlapping line segments and use turning point to separate the two lines.

Example: As an example, consider a fingertip movement trajectory shown in Figure 9 (a). The red rectangle on the figure is a timeframe consisting of 20 tracked points. If we zoom in on the timeframe, we get Figure 9 (b) where a point is labelled with a frame number according to when the point was seen, starting from 1 for the earliest point. In this example, there are more than 6 overlapping points in the timeframe, which are marked by a green circle. We use the center point (No.10) of the overlapping points as the turning point to separate the two line segments.

Extract the Line Length. The physical length of a line segment depends on a number of device-specific parameters: the sizes of the screen and the pattern grid, and the space between two touch dots. To ensure our approach is independent of the device, we normalize the physical length of a line segment to the shortest line found on the tracked trajectory. For the example shown in Figure 8 (a), the line lengths for segments, SA, AB, BC, CD, and DE, are $2l_s, l_s, 2l_s, l, 2l_s$, respectively. Here segments AB and CD have the shortest length, l_s. The physical length of a line segment is calculated by computing the Euclidean distance between the start and the end points of a segment.

Extract Direction Information. In addition to the line length, we also want to know to which direction the finger moves. This information is useful for inferring which dots are selected to unlock the pattern. Figure 10 (a) shows all possible 16 directions on a 3×3 pattern grid. The directions are numbered from 1 to 16 in clockwise. For each line segment of the tracked trajectory, we calculate its line slope and the horizontal movement of the finger (i.e. left $\rightarrow$ right or vice versa). This information will then be checked against Table I to determine the direction number of the line segment. The horizontal movement of the fingertip is determined by first using the temporal information to find out the start and the end points of the line and then comparing the horizontal coordinates of the two points. The line slope is also computed based on the coordinates of the start and the end points of the line segment. Figure 10 (b) gives the direction number of each tracked line segment of a fingertip movement trajectory.

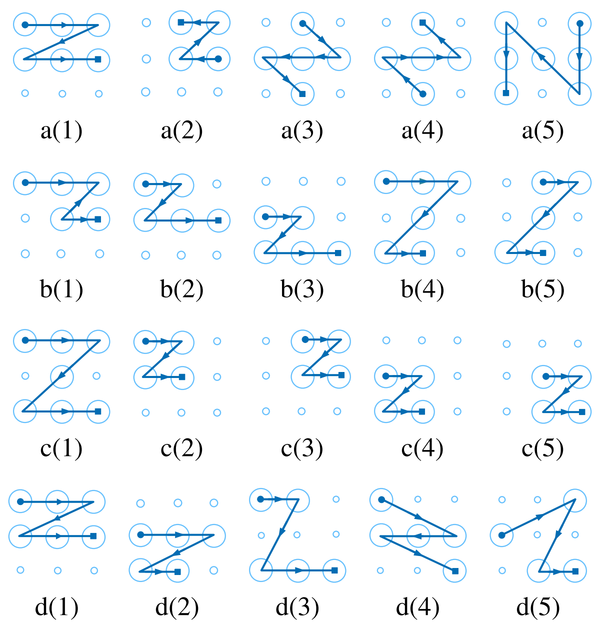


Figure 11. Possible mappings for the tracked fingertip movement trajectory presented in Figure 2 (d).

2) Map the Tracked Trajectory to Candidate Patterns: In this step, we use the extracted geometry information to map the fingertip movement trajectory to a small number of candidate patterns which will then be ranked using a heuristic. This process is described in Algorithm 3.

Identify Candidate Patterns. Our implementation simply enumerates all possible patterns for a given pattern grid to identify candidate patterns, starting from the top-left touch point. We reject patterns that do not meet the requirements that the correct pattern is expected to have. The requirements are the number of line segments (this is checked by counting the number of turning points), and the length and the direction for each line segment. This is an *automatic* process performed by our software system without any user involvement. We consider two line segments having the same length and slope if the difference between them is less than a threshold. Specifically, the relative length threshold, $lengthTh$, is set to 1.12 and the slope threshold, $directionTh$, is set to 0.25. To determine the thresholds, we have evaluated a range of possible values in our initial design experiments to chose the best performing values.

Example: We use the pattern depicted in Figure 2 as an example to describe our algorithm. Figure 11 gives several possible mappings for the fingertip movement trajectory shown in Figure 2 (d). For this particular trajectory, the collections of lengths and directions are $L = \{l, \sqrt{2}l, l\}$ and $D = \{5, 11, 5\}$ respectively. Any pattern that does not meet L or D should not be considered as a candidate pattern for this trajectory. For this reason, Figure 11 a(1)–a(5) will be rejected. Take Figure 11 a(1) as an example, the line lengths and directions for all four line segments of this pattern are $\{l, \frac{\sqrt{5}}{2}l, l\}$ and $\{5, 12, 5\}$ respectively. It does not meet the expected L or D and should be rejected. The patterns presented in b(1)–b(5) and c(1)–c(5) of Figure 11) will also be rejected for the same reason.

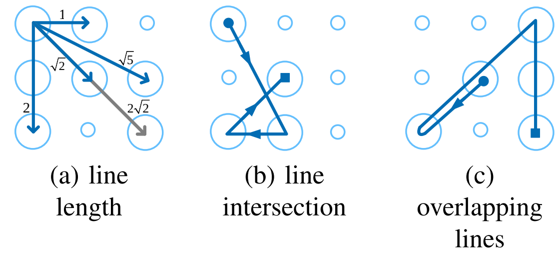


Figure 12. Illustrations of the terminologies used in Equation 2.

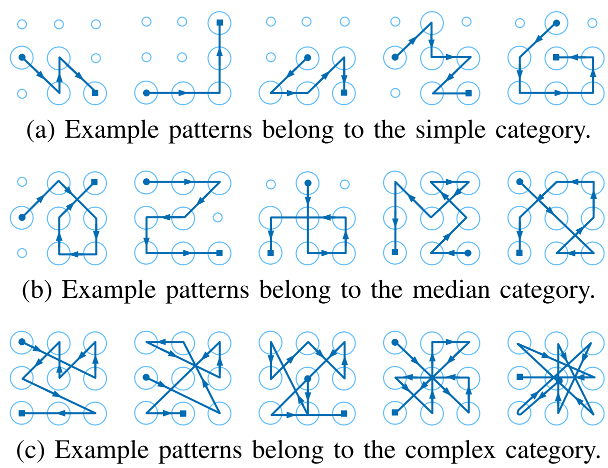


Figure 13. Examples of patterns collected from our participants. Patterns are grouped into *simple*, *median* and *complex* categories, according to their complexity scores.

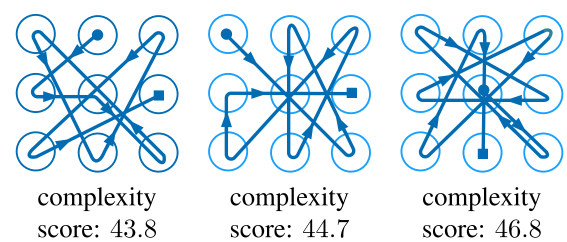


Figure 14. Three most complex patterns on a 3×3 grid based on Equation 2.

Rank Patterns. Candidates patterns are then ranked using a simple heuristic. The heuristic assumes a pattern starting from left dot of the grid is more likely to be the correct pattern over a pattern starting from a right dot. This assumption is supported by recent studies which show that people tend to select a left dot as the starting point to construct a pattern [25, 39]. If two candidate patterns start from the same dot, we consider the pattern with a longer total line length is more likely to be the correct pattern. Using these criteria, the five candidate patterns are ranked in order from subfigures d(1) to d(5) in Figure 11. Therefore, an attacker would first try the candidate pattern presented in Figure 11 d(1). This attempt will lead to a successful attack for the example presented in Figure 2. Our experimental results show that this heuristic is effective.

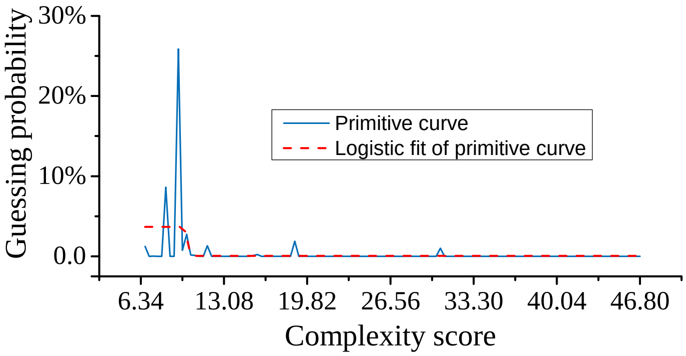


Figure 15. How likely a pattern can be guessed in five attempts as the complexity score increases. Patterns with a complexity score that is greater than 13 are unlikely to be guessed by an attacker within five attempts.

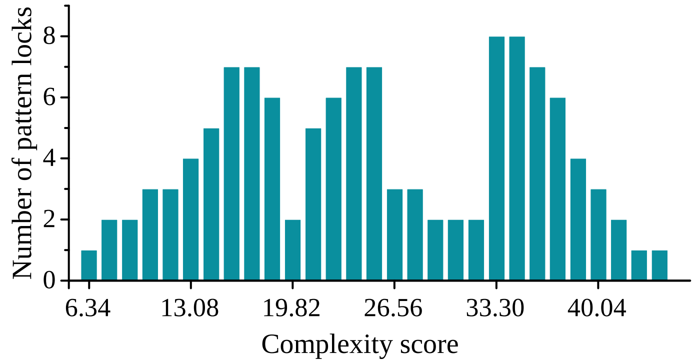


Figure 16. The distribution of complexity scores for the patterns given by our participants.

V. Experimental Setup

A. Data Collection

The patterns used in our evaluation were collected from users who use at least one Android device (a smartphone or a tablet) on a daily basis. To collect the patterns, we have distributed over 1,000 survey forms and collected back 215 valid forms, resulting in 120 unique patterns[4]. Our participants include 95 females and 120 males who were undergraduate or postgraduate students in our institution. The majority of our participants are in an age group of under 30.

To collect the patterns, we have conducted a "pen-and-paper" survey by asking participants to fill in an anonymized questionnaire. The questionnaire and survey were approved by the research ethics board (REB) of our institution. We have made sure that our survey complied with strict privacy regulations. For example, we did not collect any personally identifiable information other than the gender and age group of the participant. Our participants were well informed on the purpose of the study and how the data will be managed and used. The survey forms were distributed as voluntary homework so that the participants can take the survey form away to fill in. Users were invited to return the survey form anonymously within three weeks to a dedicated, locked mailbox, if they wish to participate in the study. To avoid a user submits multiple copies of the same form, each survey form is given a unique, randomly generated 32-digital number.

Overall, 37.6% of our participants confirmed that they use pattern lock as the screen lock to protect their Android devices on a daily basis; and 33% of those who do not use a pattern as their screen lock said that they are often required to use a pattern for authentication by an application like `Alipay`. Furthermore, 60% of our participants also indicated that the pattern they provided is currently being used or have been used in the past by themselves. Other participants (often those did not use a locking pattern on a daily basis) indicated that they have provided a pattern which they would like to use if a locking pattern is required. Based on this information, we are confident that our patterns represent some of the real world patterns. Finally, all participants believe that a complex pattern provides stronger protection than a simple one.

Table II
Screen sizes for the test phones

Screen size	MI4	Honor7	Note4
Height(cm)×Width(cm)	13.9×6.9	14.3×7.2	15.4×7.9

B. Pattern Complexity Classification

We quantify the complexity of a pattern using the complexity (strength) score proposed in [37]. The complexity score, CS_P, of a pattern, P, is defined as:

$$CS_P = S_P \times \log_2(L_P + I_P + O_P) \qquad (2)$$

where S_P is the number of connected dots, L_P is the the total length of all line segments that form the pattern (see Figure 12 a), I_P are the number of intersections (which are also termed as "knight moves" in some prior work [40], see Figure 12 b) and O_P are the number of overlapping linear segments (see Figure 12 c). To calculate the line length, we assume the length between two horizontally or vertically adjunct dots is one. Thus, our method is independent of the size of the screen and the grid.

Intuitively, the more connected dots (S_P), line segments (L_P), intersections (I_P) and overlapping line segments (O_P) that a pattern has, the more complex it is. For example, the patterns shown in Figure 13 (c) use all the nine dots of the grid, and have at least seven line segments and three intersections. There are other methods to quantify the complexity score of pattern locks, including the methods proposed by Song *et al.* [33] and Andriotis *et al.* [1]. These methods in general suggest that patterns with more connected dots and intersections are considered to provide stronger security strengths [18].

Figure 15 illustrates how the guessing probability[5] changes as the complexity score (given by Equation 2) of the pattern lock changes. The probability is calculated on the set of patterns collected from our participants, by using the pattern strength measurement method proposed in [18]. As can be

[4]Available to be download at: https://dx.doi.org/10.17635/lancaster/researchdata/113

[5]A guessing probability is how likely an attacker can successful guess a given pattern within five attempts.

seen from this diagram, FIXED: the primitive curve presents the relevance between the guessing probability and the complexity scores. It directly reveals how likely a pattern can be guessed within five attempts as the complexity score increases. The trend of the primitive curve is shown by the logistic fit curve (the red dash curve), which indicates that a pattern with higher complexity score (typically more than 13) is hard to be guessed within five attempts.

Pattern Grouping. Base on the complexity score, we divide the collected patterns into three complexity categories: *simple*, *median* and *complex*. A simple pattern has a score of less than 19, a median complex pattern has a score between 19 and 33, and a complex pattern must have a score greater than 33. This classification gives us roughly 40 patterns per category. Figure 13 gives some examples for each category while Figure 16 shows the distribution of these patterns according to their complexity scores. Based on this definition, the most complex pattern on a 3×3 grid has a score of 46.8 (see Figure 14). The complex scores of the patterns we collected range from 6.4 to 46.8.

C. Video Recording and Preprocessing

Recording Devices. We used three smartphones for video recording: an Apple iPhone4S, a Xiaomi MI4 and a Meizu2 phones. Each mobile phone was used to record 40 patterns with a 1080p HD resolution of 30 FPS under different settings.

User Participation. We recruited ten postgraduate students (five male and five female students) from our institution to reproduce the 120 patterns (collected from the users) and the 60 most complex patterns (see Section VI-A) on three target mobile phones: a Xiaomi MI4, a Huawei Honor7 and a Samsung Note4 smartphones. Table II lists the screen size for each target mobile phone.

Video Recording Setup. By default, we used the Android 3×3 native pattern grid, but we evaluated our approach using other pattern grids with different sizes in Section VI-H. We recorded each pattern under three filming angles, 45, 90 and 135 degrees, by placing the camera on the left-front, front, and right-front of the target device respectively. By default, the video was recorded indoor during daytime under a natural lighting condition. In Section VI-D we evaluated our approach under different lighting conditions both indoor and outdoor. By default, videos were recorded at a distance of 2 meters from the target device and we evaluated the impact of the filming distance in Section VI-H.

Video Filming. Before recording, our participants were given the opportunity to practice a pattern several times, so that they can draw the pattern at their natural speed. On average, this practice session took 10 trails per user per pattern. When drawing the pattern, some participants sat, while others stood, some hold the device by hands, while others placed it on a table. Each pattern was drawn on three target devices and recorded under three filming angles. Thus, for the 120 patterns collected from users, we recorded 1,080 videos in total.

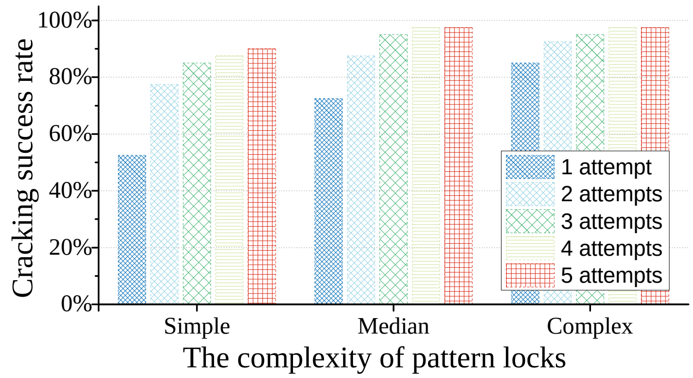


Figure 17. For each pattern category, the figure shows the success rate using no more than 1, 2, 3, 4 and 5 attempts.

Video Preprocessing. For each video stream, we used the algorithm described in Section IV-A to cut out the video segment of the unlocking process. We left around 200 to 300 milliseconds of the video segment before and after the pattern unlocking process. To track the fingertip locations, we used Windows Movie Make to highlight two areas of interest on the first frame of the video segment: one area surrounds the fingertip, and the other contains an edge of the phone (see Section IV-B2).

Implementation. Our prototyped attacking system built upon a TLD library [19]. The developed software ran on an Intel Core i5 PC with 8GB RAM. The operating system is Windows 10. Our implementation can be ported onto Android or Apple iOS systems, which is our future work. On our evaluation platform, our software takes less than 30 seconds to process a video to produce candidate patterns.

VI. Experimental Results

In this section, we first present the overall success rate for cracking the 120 patterns collected from our participants plus the top 60 most complex patterns on a 3×3 pattern grid. Our results show that our approach can successfully crack over 95% of the patterns using no more than five attempts. We then analyze how the success rate is affected by the filming distance, filming angles and camera shake. Finally, we demonstrate that direct observations lead to poor performance before evaluating our approach on alternative pattern grids.

A. Overall Success Rate

Result 1: *We can successfully crack over 95% of the patterns in five attempts and complex patterns are less secure compared to simple patterns under our attack.*

In this experiment, videos were recorded from a distance of 2 meters away from the target device. This mimics a scenario where the adversary sits at the next table to the user in a public space (e.g. a restaurant). The smartphones used for filming in this experiment were hand-held. Figure 17 shows the success rate for cracking different types of patterns within 1, 2, 3, 4 and 5 attempts. For all the patterns used in this evaluation, our approach does not generate more than five

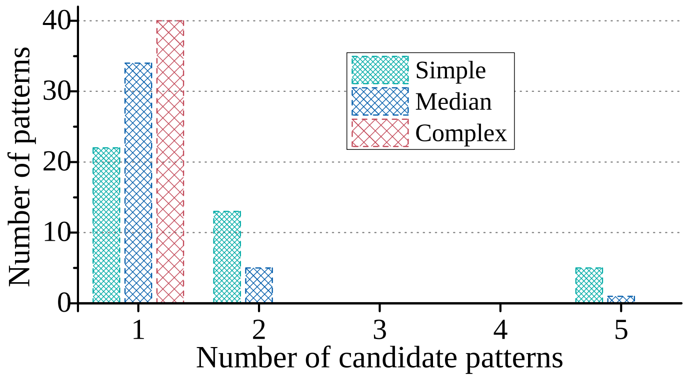


Figure 18. The distribution of candidate patterns for each category. No more than 5 candidate patterns were generated by our algorithm.

candidate patterns. For complex patterns, we are able to crack all except one in the first attempt. For simple and median patterns, the success rate increases with more tries. In one attempt, we are able to successfully crack 60% and 87.5% of the simple and median patterns respectively. With two attempts, the success rate increases to 87.5%, and 95% for simple and median patterns respectively. Using five attempts, we are able to crack all simple patterns and all but one median patterns. The reason that we failed on one median and one complex patterns is because of some blur motions of the video footage (probably caused by the video compressing algorithm), which leads to many tracking failures. But we are able to crack the same pattern using a video filmed by a different device. It is important to note that the Android OS will not lock the device unless seeing more than five failed tries [15]. This means, in practice, our approach is able to successfully crack most locking patterns.

Another interesting observation is that in contrast to many people's intuition, complex patterns do not provide stronger protection under our attack because most of these patterns can be cracked in one attempt. This is because although complex patterns can better protect the user against direct observation techniques like shoulder surfing [29], their unique graphical structures often allow our system to reject most patterns to produce a single candidate pattern. This is confirmed by Figure 18. It shows that for most median and all complex patterns, our system produces one candidate pattern – the correct one for most of our test cases.

We also evaluated our approach using the top 60 most complex patterns (according to Equation 2) on a 3×3 grid. To evaluate our approach on a wide range of patterns, we exclude patterns that are simply a rotation to an already chosen pattern. Figure 14 illustrates three highly complex patterns which have a complexity score between 43.8 and 46.8. The three patterns use all the nine dots of the grid and have a larger number of line segments, intersections and overlapping lines when compared to simpler patterns. Because of their complex graphical structures, it would be hard to remember these patterns using direct observation techniques. In this experiment, we can crack all the complex patterns in one attempt. This result reinforces our claim that complex patterns are less security under video-based attacks.

Table III
TRACKING PRECISION VS FILMING DISTANCE

Distance	1 m	2 m	3 m	3.5 m
fingertip	100%	98.7%	80.9%	68%
device edge	100%	99.4%	90.6%	69%

B. Impact of Filming Distances

Result 2: *We can crack over 80% of the patterns in five attempts, if the video was filmed using a smartphone within a distance of 2.5 meters away from the target.*

We would like to know how the filming distance affects the success rate of the attack. To do so, we have selected all 120 pattern locks and we varied the filming distance from 1 meter to 3.5 meters. Figure 20 shows how the cracking success rate changes as the filming distance increases. There is slight differences in the success rate between this diagram and Figure 17 because we used less patterns in this experiment. When the filming distance is less than 2 meters, our approach can crack all patterns in five attempts. The success rate drops significantly when the filming distance is greater than 2.5 meters. The quality of the video filmed by a mobile phone tends to drop significantly with many object deformations. The degradation of the video quality makes it difficult for the TLD algorithm to successfully track objects across video frames. This is confirmed by Table III which shows that the tracking precision for the fingertip and the device edge drops from around 99% to 68% when the filming distance increases from 2 meters to 3.5 meters. The increased tracking failures result in an increased number of missing points on the tracked trajectory, leading to a deteriorative performance in identifying candidate patterns. This can be seen from Figure 19 where the quality of tracking clearly decreases when the filming distance is greater than 3 meters. Nonetheless, our approach can achieve a high success rate when the filming distance within 2.5 meters. Such a distance allows an attacker to record the video without raising suspicions in many day-to-day scenarios (some of these are described in Figure 1).

We also evaluated our approach on videos filmed using a Nikon D90 single-lens reflex (SLR) camera with a 105mm lens. The SLR camera was placed from a distance of 9 meters away from the target device. For this set of videos, we are able to achieve the same performance when compared to using videos filmed by a mobile phone camera with a 2-meter filming distance. The further filming distance is largely due to better video quality brought by the advanced SLR camera and the lens. Therefore, in practice, an attacker can also use a professional video recording device to launch the attack from a further distance.

C. Impact of Camera Shake

Result 3: *Our method can tolerate a certain degree of camera shake in the hand-held mode.*

In this experiment, we used an IPhone4S smartphone to record how a pattern is drawn on a Huawei Honor7 phone.

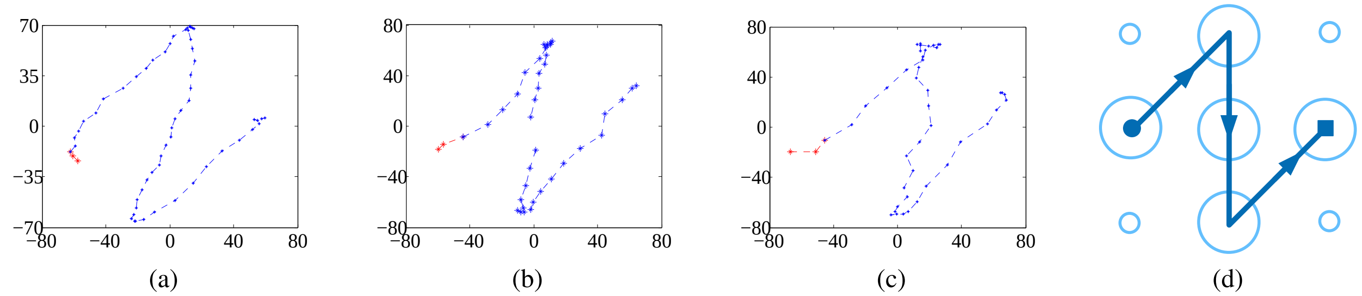


Figure 19. Tracked fingertip trajectories (user's perspective) for the pattern shown in (d) from a video filmed from a distance of 2m (a), 3m (b), and 3.5m (c) respectively away from the target device. The tracking quality decreases when the filming distance is greater than 3m.

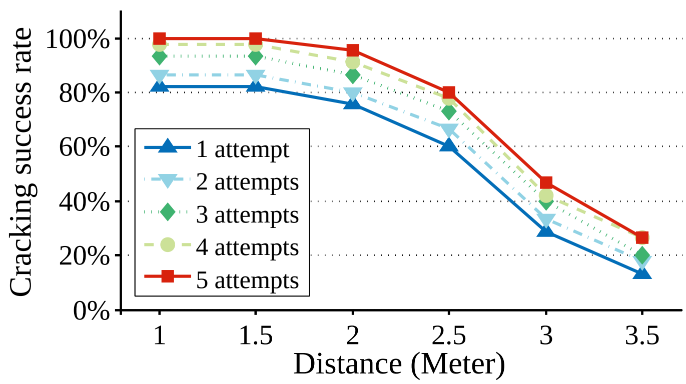


Figure 20. Impact of the filming distance.

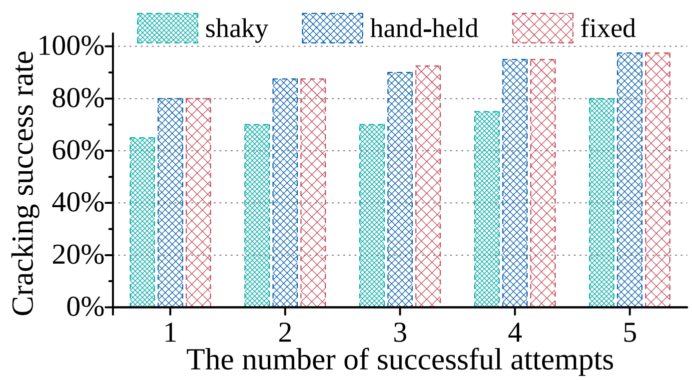


Figure 22. Impact of camera shake. Our approach has the same success rate under the hand-held and the fixed modes and the performance degradation under the shaky mode is modest.

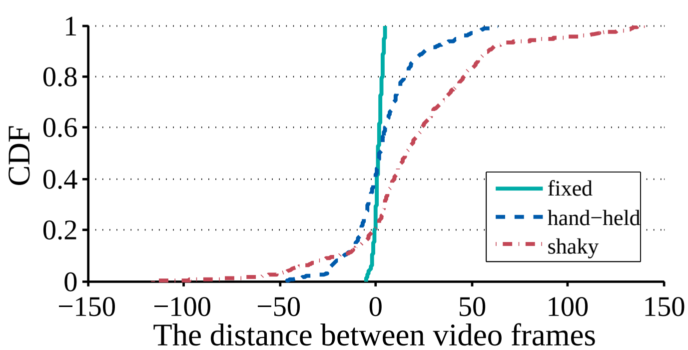


Figure 21. The cumulative distribution function (CDF) for different video recording modes.

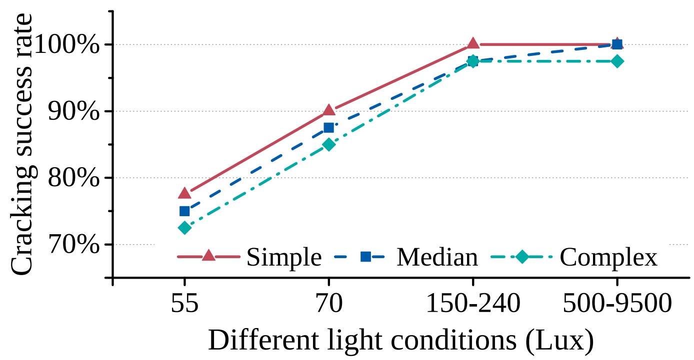


Figure 23. The cracking success rate within five attempts under different lighting conditions.

This experiment was carried out under three settings: *fixed*, *hand-held* and *shaky*, where the filming device was respectively fixed using a tripod, hand-held, and hand-held but with constant movements of approximate 2cm in the horizontal or the vertical directions. The recording device was placed on the left-front, front, and right-front of the target device. In the experiment, we fixed the target device on a table using double-sided tapes.

We use a reference point to quantify camera shake. The point is the center position of an area of the target device. The area is marked by a boundary box on the first frame (see Figure 5). We calculate the difference (in terms of pixels) for where the reference point was seen in two consecutive video frames. We then use the difference to measure the degree of camera shake. Figure 21 shows the cumulative distribution function (CDF) of camera shake under the three different filming settings. Here, the wider the distribution is, the less steady the filming is. The shaky mode is least stable where the difference of the reference point between two video frames can be up to 250 pixels.

Figure 22 shows that our approach has the same performance under the hand-held and the fixed modes. The modest camera sake under the hand-held mode has little impact on performance thanks to our camera-shake-calibration method. We observe deteriorative performance under the shaky mode, but the performance degradation is modest (80% vs 97% in 5

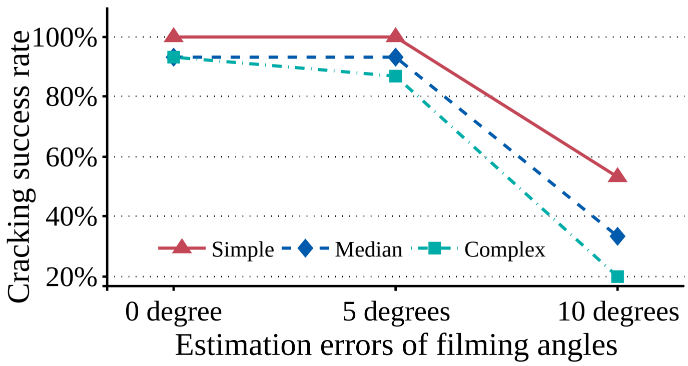


Figure 24. Impact of estimation errors of filming angles.

Table IV
LIGHTING CONDITIONS

Scenarios	Indoor	Indoor	Indoor	Outdoor
Time	nighttime	nighttime	daytime	daytime
Light Source	warm LED	white fluorescent	sunlight	sunlight
Light Intensity (Lux)	$55-70$	$70-100$	150–240	500–9500

attempts). In reality, an attacker would avoid drastic camera shake by firmly holding the video recording device.

D. Impact of Lighting Conditions

Result 4: *Low-light has a negative impact on the success rate of the attack but our approach can still break over 70% of the patterns when the video was filmed in a low-light environment.*

In this experiment, videos were recorded under different lighting conditions both indoor and outdoor. The experimental settings are given in Table IV. The light intensity of these condidtions ranges from 9500 lux (strong light), onto 240 lux (normal light), and 55-70 lux (low light). These represent some of the day-to-day scenarios where filming can take place. For each setting, we asked each of our 10 participants to draw all collected 120 patterns on a Xiaomi MI4 phone. We used an iPhone4S phone to record the video. The filming camera was place on the left-front, front, and the right-front of the target device from a distance of 2 meters.

Figure 23 shows that the success rate increases when video filming were performed in a brighter lighting condition as the light intensity changes from 55 lux to 9500 lux. This is expected as low-light leads to increased video noise, blurred motion and poor focus, which all have a negative impact on the TLD algorithm. Nonetheless, our attack can still crack over 70% of the patterns in a filming environment with low light.

E. Impact of Filming Angle Estimation

Result 5: *Our attack performs well when the error of filming angle estimation is less than 5 degrees.*

Our approach needs to transform the tracked fingertip movement trajectory to the user's perspective based on an estimation of the filming angle (Section IV-C). Because our filming angle estimation algorithm gives highly accurate estimation, we did not find the estimation error to be a problem in our experiments. Nonetheless, it is worth studying how the estimation error affects the success rate of our attack. To do so, we deliberately added an error of 5-10 degrees to the estimation.

Figure 24 shows the results of this experiment. When the error is less than ± 5 degrees, there is little impact on *complex* patterns and no impact at all on *simple* and *median* patterns. However, an estimation error of more than 10 degrees can significantly affect the success rate. Given such an error, the resulted trajectory after transformations will be significant different from the correct pattern. For example, when the estimation error is 10 degrees from the true value, on average, 0.8, 2.6 and 4.2 line segments per pattern respectively will be incorrectly labelled for *simple*, *median* and *complex* patterns. This explains why the success rate for complex patterns drops significantly when there is an error of 10 degrees.

F. Impact of Screen Sizes and Cameras

Result 6: *The size of the smartphone touch-screen and the mobile camera have little impact on the success rate.*

Intuitively, the success of the attack would be influenced by the size of the touch-screen because the physical length of the pattern depends on the size of the screen. Likewise, the accuracy also may be affected by the quality of the video because of the filming camera. To evaluate this effect, we asked 10 participants to randomly select 90 patterns (30 patterns from each of the three pattern categories). We used three recording devices: IPhone6, Xiaomi MI4, Vivo X7 and Samsung Note4 to record the video footage. The target devices are a Huawei Honor7, a Samsung Galaxy Tablet, and an IPhone4S[6]. The filming distance is 2 meters from target device. Table V gives the screen size of each device, which ranges from 3.5 *inch* (IPhone4S) to 9.6 *inch* (Samsung Tablet). Table VI lists the mobile cameras and their main performance parameters, which represent some of the most commonly used smartphone cameras.

Figure 25 (a) confirms that our attacking method is not significantly affected by the size of the target device's screen. Our approach achieves a 98.3% success rate within five attempts when the screen size is 3.5 *inch* (a small screen in the current mobile market). It can crack all tested pattern locks when the size of screen is larger than 5.2 *inch* (Honor7). We also evaluated our approach using different types of filming cameras. The result is presented in Figure 25 (b). Our method can reconstruct all patterns (100%) within five trials using the IPhone6 and the Note4 because their camera settings (hardware and software) lead to a better video quality. When using MI4 and Vivo X7 for filming, the success rate can reach 96.7% and 98.9% within five attempts respectively, where our approach incorrectly identifies 3 and 1 (out of 90) patterns respectively due to blur motion in the video footage. This experiment shows that the screen size and filming camera have little impact on the attacking success rate.

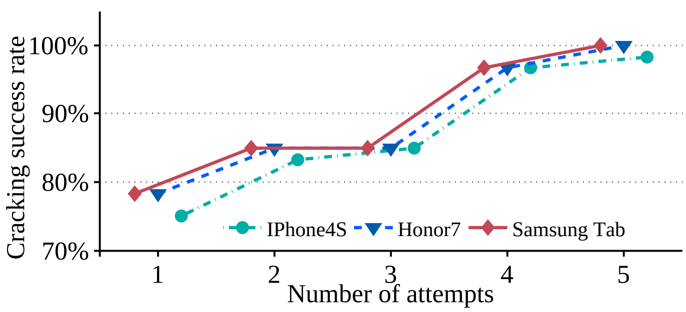
[6]For iOS, participants are asked to draw patterns on the login interface of Alipay as iOS currently does not support pattern lock natively.

Table V
SCREEN SIZES PER TARGET PHONE

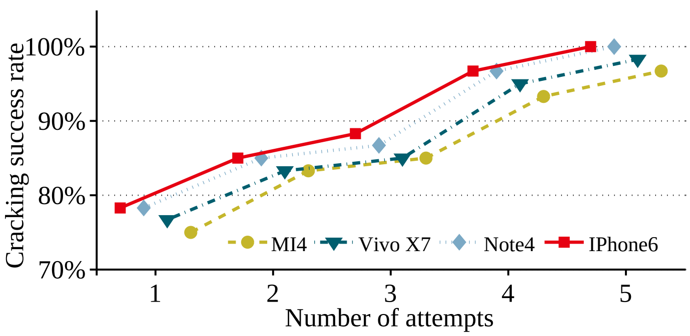
Screen-size	IPhone4S	Honor7	Samsung Tablet
Height(cm)×Width(cm)	11.5 × 5.9	14.3 × 7.2	24.2 × 15.0

Table VI
FILMING CAMERA PARAMETERS

Parameters	IPhone6	Vivo X7	MI4	Note4
Frame Rate (fps)	30	30	30	30
Pixels	8mp	13mp	13mp	16mp
Focus (mm)	4.15	4	4	4.2
Sensitivity (ISO)	3200	3200	3000	5000



(a) screen size



(b) camera brands

Figure 25. The cracking success rate for different target screen sizes and filming cameras.

G. Inferring Patterns with Eyes

Result 7: *Our attacking methodology significantly outperform direct observation techniques.*

In this experiment, we investigate whether an attacker can infer the pattern by simply watching the video or through direct observations. To answer this question, we asked each of our 10 participants to watch 60 videos (where a pattern was drawn by other participants) to guess the pattern. We only played the video segment during which a pattern is drawn to the participant (around 3 seconds per video). To familiarize participants with the process, we played five sample videos and showed the correct patterns at the end of each video to our participants before the experiment. Each participant then had 10 minutes to watch a video and five chances to guess a pattern. They could adjust the playing speed and replay the video multiple times as they wished.

Figure 26 (a) shows the success rate of pattern guessing with bare eyes. Our participants correctly guessed for nearly half of the simple patterns in five attempts. However, they found that it is difficult to infer complex patterns with many line segments, overlapping lines and intersections. The success rate of guessing complex patterns is less than 10% in five attempts. This is not a surprising result because although it is possible to correctly guess patterns with simple structures by watching the video, doing so for patterns with more complex structures is much harder.

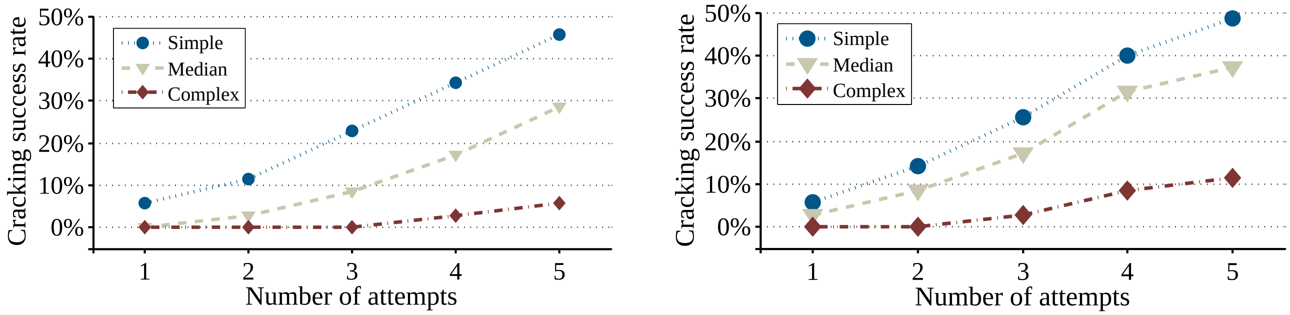
We also asked participants to directly observe how a pattern was drawn from a distance of 2 meters away from the target device. The intuition behind this evaluation is that human eyes can catch richer information over a video camera. The results of this experiment are shown in Figure 26 (b). As can be seen from the diagram, although the success rate is improved compared to directly watching the video, the chances for guessing the correct pattern in 5 attempts are quite low. In fact, the success rates are just 48.3%, 38.3% and 11.7% respectively for simple, median and complex patterns.c

H. Evaluation on Other Pattern Grids

Result 8: *A pattern grid with more dots provides stronger protection but our attack can still crack most of the patterns.*

There are a few applications (such as CyanLock) and customized ROMs available to increase the size of the pattern grid from 3×3 to 4×4, 5×5, and 6×6. Although a 3×3 grid remains a popular choice (as it is supported by the native Android OS), it is worth studying whether having more touch dots on a pattern grid leads to stronger security. In this experiment, we first ranked all possible patterns for each grid setting in ascending order according to their complexity scores. We then equally divided the patterns into three groups, simple, medium and complex, and asked our participants to randomly select 20 patterns from each group for evaluation. We report the success rate of our attack within five attempts. In the experiments, we have adapted our algorithms for each grid setting by adjusting the algorithm parameters (such as the line direction numbers).

Figure 27 shows the success rate of our attack for different grids. Similar to a 3×3 grid, our approach achieves a higher success rate for complex patterns over simple ones. On average, we can crack 90% of the complex patterns. We observed that a grid with more dots does provide stronger protection. For complex patterns, the success rate of our attack drops from 95% on a 4×4 grid to 87% on a 6×6 grid. For simple patterns, the success rate of our attack drops from 85% on a 4×4 grid to 75% on a 6×6 grid. This is because a fingertip trajectory can be mapped to a larger number of candidates on a grid with more dots. For instance, the pattern shown in Figure 2 (f) can be mapped to 55 candidate patterns on a 6×6 grid as opposite to 5 on a 3×3 grid. Overall, our attack can crack over 75% (up to 95%) of the patterns within five attempts. One of the purposes of introducing pattern grids with more dots is to allow users to use more complex patterns. However, this experiment suggests that complex patterns remain less security on these grids under our attack.



(a) video watching (b) direct observations

Figure 26. Success rates of guessing patterns through watching the video (a) or direct observations (b).

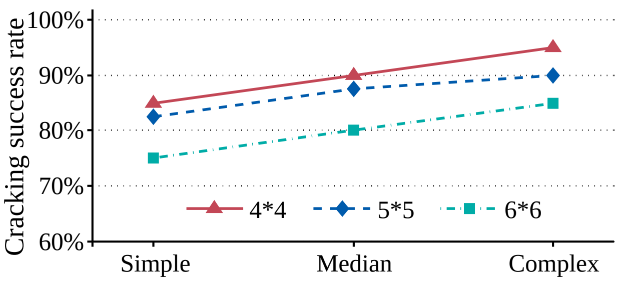
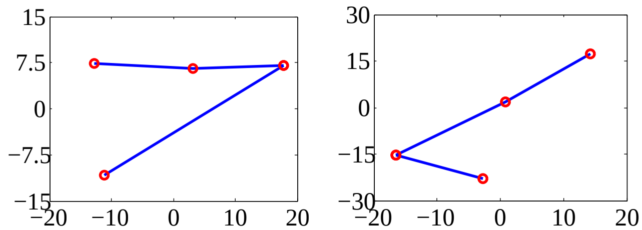
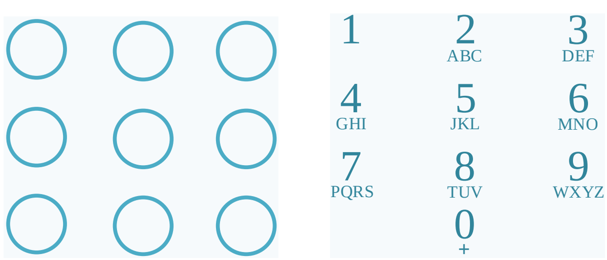


Figure 27. Success rates of our attack for different locking grids.



(a) password: 1234 (b) password: 3570

Figure 29. Examples of tracked fingertip trajectory comprised of touching points. The red circle represents the touching points.



(a) pattern-based interface (b) pin-based interface

Figure 28. Pattern- (a) and pin-based (b) authentication interfaces.

Table VII
PIN-BASED PASSWORDS USED IN OUR EXPERIMENTS

Category	PIN-based Passwords				
Commonly used PINs	1234	1111	1212	1004	2000
	6969	4321	1122	2001	2580
	1357	2468			
Randomly generated PINs	1205	3570	0729	3719	9867
	5946	3451	7403	2209	3560
	1043	4628	5372	2830	7102
	6193	2941	3471		

I. Attacking PIN-base Passwords

Result 9: *PIN-based passwords are also vulnerable under our video-side channel attack. We can break over 85% of the pin-based passwords within five attempts using a variant method based on our approach.*

An interesting question to ask is, could this method be used to attack PIN-based passwords? To answer this question, we apply our attacking method on 30 4-digital passwords. Among these passwords, 12 of them are most common used passwords (given by a PIN analysis survey conducted by Berry [9]). The remaining passwords are randomly selected. For each password, we ask our participants to type in the password on a Xiaomi MI4 phone. Table VII lists all PIN-based passwords considered in this experiment. Figure 28 shows the subtle differences between the PIN-based keypad and the Android pattern grid. In this experiment, we used an IPhone4S phone to record the video from three angles of the target device: the left-front, front and right-front. The filming distance is 2 meters.

Variant Methodology. Because the differences between the PIN-based password, we need to adapt our method. The two main differences are summarized as follow: (1) the number of the touch dots are different; and (2) each dot on the PIN pad can be visited multiple times, while each dot can only be visited once on pattern lock. The later difference requires us to identify dots that have been visited multiple times. Our preliminary experiments suggest that we can reconstruct the trajectory of PIN-based password by connecting the touching points[7]. We can obtain the location of touching point by tracking the up-and-down motion direction of the fingertip. This is inspired by the prior work conducted by Shukla *et al.* [31]. To reconstruct PIN-based passwords, our attacking method records some additional geometric information, including the direction and length information. Figure 29 shows

[7]Touching points are the points that are tracked when the user's fingertip touches the screen.

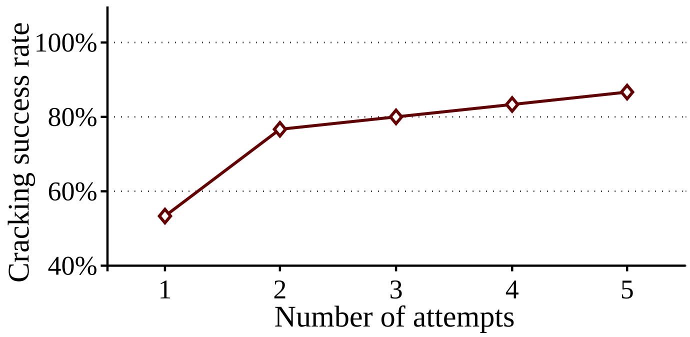


Figure 30. The success rate of cracking PIN-based passwords with different number of attempts.

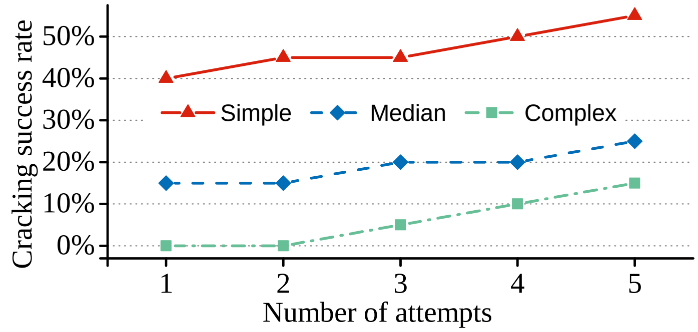


Figure 31. The cracking success rate drops significantly when the video footage only captures the user's fingertip.

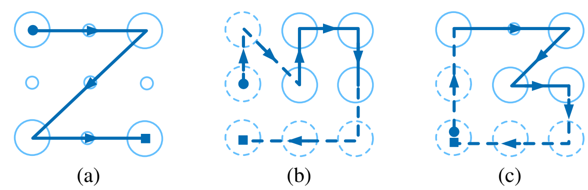


Figure 32. Examples of the variant pattern lock mechanism for defending against our video-based attack. Here the patterns with the solid lines are the true patterns. Before or after drawing the true pattern, the user can draw a random pattern shown in (b) and (c), which are marked with dash lines. FIXED: The solid lines in (b) and (c) mark the true patterns. In order to increase the number of candidate patterns, a previously unvisited dot, such as the top-middle smaller dot in (a) or (c), can be bypassed if it is part of a horizontal, vertical or diagonal line segment of the pattern.

the reconstructed fingertip trajectory using the new attacking method.

Figure 30 shows the results of applying our attacking method to PIN-based passwords. Using first attempt, the success rate is 53.3% because most of evaluated passwords have two or more candidate patterns. However, the success rate increases with more attempts and reaches 86.7% using 5 attempts. Using five attempts, our method fail to reconstructor four passwords. For two of failed passwords, our approach produces more than five candidate passwords; and for the other two, our approach incorrectly locates one of the touching points due to blur motion of the video footage. Overall, our method can achieve a high cracking success rate for PIN-based passwords. Our findings are in line with prior works on video-based side channel for PIN-based passwords [31].

J. Limited Study

Our attacking method requires the user's fingertip and part of the device to be seen in the video footage. As described in Section IV-B2, this is essential for calculating the coordinates of the fingertip and for camera shake calibration. An interesting question to ask is can we relax this requirement, i.e., how will the attack perform if the video footage only captures the user's fingertip. If our findings suggest that the success of the attack requires seeing part of the device, a potential countermeasure is to educate users to cover their devices when entering their patterns. As a limited study, in this experiment we ensure the video only capture user's fingertip. This study is performed on 20 patterns randomly selected by our participants.

Figure 31 shows the success rate of our attacking method is low when the device is not seen in the video footage. This is large due to the fact the negative impact of the camera shake on tracking fingertip movement trajectory (see also Section IV-B2). With only the fingertip location, the attacking method fails to cancel the camera shake effect, leading to low quality fingertip movement trajectory which in turn translates to a poor success rate (less than 50%, 30% and 20% for simple, medium and complex patterns respectively).

VII. Discussions

A. Countermeasure

The success of our attack depends on three factors: (1) knowledge of the pattern grid; (2) a decent quality video footage allowing the algorithm to track the fingertip movement; (3) successfully identifying a video segment that captures the entire process of pattern drawing.

For the first factor, the attacker can obtain the relevant information by simply looking the pattern grid of the target operating system or application. Randomization techniques such as randomized pictures [10, 32], which randomly shuffle the location of touch points each time, could be a solution. However, randomization-based solutions often come at the cost of poorer usability. This can prevent them to be used at a large scale. Regarding the second factor, there are ways, such as KALEIDO [47], to prevent unauthorized videotaping by dynamically changing the colour and brightness of the screen to confuse the filming camera. A non-technical solution for this aspect would be to educate users to fully cover their fingers when drawing a pattern. But doing this on a large-screen device could be awkward especially when the device is held by one hand.

For the third factor, the attacker's solution depends on the type of the pattern. For a screen lock, pattern drawing is the first activity (except for receiving a phone call or making an emergency call) when the device is retrieved. Therefore, identifying the video segment is straightforward. When the pattern is used by applications, we have observed that users typically pause for a few seconds before or after entering the pattern. Therefore, an experienced attacker should also be able to identify the video segment in case our automatic algorithm (presented in Section IV-A) fails to do so. A potential countermeasure is to mix pattern unlocking with other on-screen activities. For examples, before and after pattern drawing, the system can ask the user to type in a sentence using a Swype-like method or to draw some graphical shapes. The problem of this approach is it may annoy users by asking them to do more, especially for screen unlocking – an activity that is performed many times a day.

B. A Potential Remedy

We propose a countermeasure to make it hard to obtain a meaningful video footage. When design the approach, we try to find a balance between the usability and the security. Our approach is simple – before or after drawing the correct pattern, the user is asked to draw a random pattern to confuse the attacker. In other words, if the pattern drawn by user contains the true pattern, the system will accept the pattern. To this end, we relax the rules for creating a pattern to allow a touching dot to be visited multiple times. Further, for purpose of increasing the number of candidate patterns, we also change the rule that a previously unvisited dot can be bypassed if it is part of a horizontal, vertical or diagonal line segment of the pattern.

Figure 32 gives some pattern examples generated using our rules. We also asked our participants if they feel these rules decrease the usability. The answers from the all 10 participants are negative (i.e., they do not think this method decrease the usability). We believe our countermeasure makes it harder to launch the attack as the adversary will now need to identify which part of the drawing is the true pattern. To prove hypothesis, we applied our attacking method to reconstruct 30 patterns generated by this new pattern lock design. Our experiment show that only two out of the 30 patterns can be cracked by our attacking method. After having a close look of the two successful cracked patterns, we found that they have little difference with the true patterns. This experiment confirms that the introduction of randomness can improve the pattern lock security under video-side channel attacks.

It is worth mentioning that our countermeasure is not a panacea. In fact, finding a balance between the usability and security for graphical passwords remains an open problemFIX:[]. Even with our countermeasure, an attacker can still use multiple videos that record the same user when doing pattern drawing to find the common pattern structure (which will likely to be the true pattern). Therefore, while our countermeasure is simple to implement and can increase the overhead for performing the attack, an experienced adversary could still successfully launch the attack.

C. Implications

While pattern lock is preferable by many users [11], this work shows that it is vulnerable under video-based attacks. Our attack is able to break most patterns in five attempts. Considering Android allows five failed attempts before automatically locking the device, our work shows that this default threshold is unsafe. We demonstrated that, in contrast to many users' perception, complex patterns actually do not provide stronger protection over simple patterns under our attack.

It is worth mentioning that our approach is only one of the many attacking methods that researchers have demonstrated. Examples of these attacks include video-based attacks on keystroke-based authentication [31, 45], sensor-based attacks for pattern lock [46]. Authentication methods that combine different authentication methods [14, 24, 26, 35] to constantly checks the user's identity could be a solution.

VIII. Related Work

Our work lies at the intersection between computer vision based attacks and cracking graphical- and touch-based authentication methods. This work brings together techniques developed in the domain of computer vision and motion tracking to develop a new attack. Our work is the first attempt of reconstructing a locking pattern from a video footage without capturing the content displayed on the screen.

Computer Vision-based Attacks No work has targeted using video footage to crack Android pattern lock and this is the first to do so. Our work is inspired by the work presented by Shukla *et al.* [31] on video-based attacks of PIN-based passwords. In addition to addressing the new challenges highlighted in Section I, our work differs to their approach in two ways. Firstly, we target a different authentication method, i.e. graphical-based passwords are fundamentally different from PIN-based passwords. Secondly, our approach does not require knowledge of the size of the screen or the grid. Other work in the area including [45] which attacks PIN-based passwords by analyzing how the screen brightness changes when entering a password. But the subtle changes of the screen brightness can be dramatically affected by the lighting condition. This restricts the application of their approach. There is a body of work using reflections to recover information typed by the user [5, 22, 28, 42]. These schemes require having a clear vision of the content displayed on the screen while our approach does not have such a requirement.

Cracking Graphical-based Passwords. Aviv *et al.* demonstrated that it is possible to reconstruct a locking pattern by analyzing the oily residues left on the screen [3]. This method is highly restricted as oily residues can be messed up by any on-screen activities after pattern drawing. Zhang *et al.* exploit the WiFi signal interferences caused by finger motions to recover patterns [46]. Their method requires a complex setup and is highly sensitive to moving objects of the environment because the WiFI signal can be disrupted by a moving object.

Attacks on Touch-based Authentication. Ballard *et al.* implemented a forgery attack on handwriting authentication [6]. Using a small number of training examples, they achieve a high success rate for this attack. More recently, Serwadda *et al.* show that a simple robot can achieve high penetration rates against touch-based authentication systems by analyzing on-screen gestures including swiping and zooming [30]. In this paper, we present a new, video-based attack for graphical-based passwords. Research in this area all demonstrates the need for a closer look of the security risks of touch-based authentication.

Study of Android Pattern Lock. Uellebenk *et al.* study how people use Android pattern lock on a daily basis [39]. They found that although there is a large number of Android patterns, in practice many people only use a small set of them due to users' bias in generating patterns. Løge explored the correlation between human's characteristics (e.g. ages and genders) and the choice of patterns [25]. Her study shows that users have a bias in selecting the starting dot to form a pattern and people tend to use complex patterns for sensitive applications. Aviv *et al.* developed a brutal-force based algorithm to crack pattern lock [2]. Their results show that patterns generated on a grid of 3×3 and 4×4 dots can be cracked within thousands of guesses, where simple patterns needs less attempts. In more recent work [4], they study the representativeness of pattern locks collected through various methods. Their work suggests that there are subtle differences for patterns collected using a pen-and-paper and online survey.

Motion Tracking. In addition to TLD, there are other methods proposed in the past for tracking object motions. Some of them apply image analysis to track the hand and gesture motions from video footage [8, 36, 43]. In this paper we do not seek to advance the field of motion tracking. Instead we demonstrate that a new attack can be built using classical motion tracking algorithms. We show that the attack presented in this work can be a serious threat for Android pattern lock.

IX. Conclusions

This paper has presented a novel video-based side-channel attack for Android pattern lock. The attack is based on a video filmed a distance of 2 meters away from the target device using a mobile phone camera. The attack is achieved by employing a computer vision algorithm to track the fingertip movement from the video, and then using the geometry information of the fingertip movement trajectory to identify the most likely patterns to be tested on the target device. Our approach was evaluated using 120 unique patterns collected from independent users and some of the most complex patterns. The experimental results show that our approach is able to successfully crack over 95% of the patterns in five attempts. We show that, in contrast to many people's belief, complex pattern actually provides weaker protection over simple patterns under our attack. Our study suggests that Android pattern lock is vulnerable to video-based side-channel attacks.

References

[1] P. Andriotis, T. Tryfonas, and G. Oikonomou, *Complexity Metrics and User Strength Perceptions of the Pattern-Lock Graphical Authentication Method*, 2014.

[2] A. J. Aviv, D. Budzitowski, and R. Kuber, "Is bigger better? comparing user-generated passwords on 3x3 vs. 4x4 grid sizes for android's pattern unlock," in *Computer Security Applications Conference*, 2015, pp. 301–310.

[3] A. J. Aviv *et al.*, "Smudge attacks on smartphone touch screens," in *4th USENIX Conference on Offensive Technologies*, 2010.

[4] P. J. L. Aviv A J, Maguire J, "Analyzing the impact of collection methods and demographics for androids pattern unlock," in *Workshop on Usable Security (USEC)*. Internet Society, 2016.

[5] M. Backes *et al.*, "Tempest in a teapot: Compromising reflections revisited," in *IEEE S & P '09*.

[6] L. Ballard *et al.*, "Forgery quality and its implications for behavioral biometric security," *IEEE SMC*, 2007.

[7] D. a. Balzarotti, "Clearshot: Eavesdropping on keyboard input from video," in *IEEE S &P '08*.

[8] J. Beh *et al.*, "Rule-based trajectory segmentation for modeling hand motion trajectory," *Pattern Recognition*, 2014.

[9] N. Berry, "Pin analysis," http://www.datagenetics.com/blog/september32012/index.html, 2012.

[10] R. Biddle *et al.*, "Graphical passwords: Learning from the first twelve years," *ACM Computing Surveys (CSUR)*, 2012.

[11] D. V. Bruggen, "Studying the impact of security awareness efforts on user behavior," Ph.D. dissertation, University of Notre Dame, 2014.

[12] L. F. Cranor *et al.*, Eds., *It's a Hard Lock Life: A Field Study of Smartphone (Un)Locking Behavior and Risk Perception*.

[13] A. De Angeli *et al.*, "Is a picture really worth a thousand words? exploring the feasibility of graphical authentication systems," *Int. J. Hum.-Comput. Stud.*, 2005.

[14] A. De Luca *et al.*, "Touch me once and I know it's you!: implicit authentication based on touch screen patterns," in *CHI '12*.

[15] S. Egelman, S. Jain, R. S. Portnoff, K. Liao, S. Consolvo, and D. Wagner, "Are you ready to lock?" in *CCS '14*.

[16] v. G. R. Grompone *et al.*, "LSD: a fast line segment detector with a false detection control." *IEEE PAMI*, 2010.

[17] T. Hastie and R. Tibshirani, "Discriminant adaptive nearest neighbor classification," *IEEE PAMI*, 1996.

[18] A. A. J. Heidt S, "Refining graphical password strength meters for android phones," in *USENIX Twelfth Symposium on Usable Privacy and Security (SOUPS)*, 2016.

[19] Z. Kalal, "TLD: Tracking-learning-detection," http://kahlan.eps.surrey.ac.uk/featurespace/tld/.

[20] Z. Kalal *et al.*, "Tracking-learning-detection," *IEEE PAMI*, 2012.

[21] P. G. Kelley, S. Komanduri, M. L. Mazurek, and R. Shay, "Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms," in *Security and Privacy*, 2012, pp. 523–537.

[22] M. G. Kuhn, "Compromising emanations: eavesdropping risks of computer displays," Ph.D. dissertation, University of Cambridge, 2002.

[23] M. H. Kutner, C. J. Nachtsheim, and J. Neter, "Applied linear regression models (5th ed.)," *Technometrics*, vol. 26, no. 4, 2004.

[24] Z. Ling and othersn, "Secure fingertip mouse for mobile devices," in *INFOCOM '16*.

[25] M. D. Løge, "Tell me who you are and i will tell you your unlock pattern," Master's thesis, Norwegian University of Science and Technology, 2015.
[26] M. Mannan and P. C. van Oorschot, "Using a personal device to strengthen password authentication from an untrusted computer," in *Financial Cryptography and Data Security*. Springer, 2007, pp. 88–103.
[27] M. L. Mazurek, S. Komanduri, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, P. G. Kelley, R. Shay, and B. Ur, "Measuring password guessability for an entire university," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 173–186.
[28] R. Raguram *et al.*, "iSpy: automatic reconstruction of typed input from compromising reflections," in *CCS '11*.
[29] J. Rogers, "Please enter your four-digit pin," *Financial Services Technology*, 2007.
[30] A. Serwadda and V. V. Phoha, "When kids' toys breach mobile phone security," in *CCS '13*.
[31] D. Shukla *et al.*, "Beware, your hands reveal your secrets!" in *CCS '14*.
[32] H. Siadati *et al.*, "Fortifying android patterns using persuasive security framework," in *UBICOMM 2015*.
[33] Y. Song, G. Cho, S. Oh, H. Kim, and J. H. Huh, "On the effectiveness of pattern lock strength meters:measuring the strength of real world pattern locks," in *ACM Conference on Human Factors in Computing Systems*, 2015, pp. 2343–2352.
[34] L. Standing, J. Conezio, and R. N. Haber, "Perception and memory for pictures: Single-trial learning of 2500 visual stimuli," *Psychonomic Science*, 1970.
[35] D. Stefan *et al.*, "Robustness of keystroke-dynamics based biometrics against synthetic forgeries," *computers & security*, 2012.
[36] B. Stenger *et al.*, "Model-based hand tracking using a hierarchical bayesian filter," *IEEE PAMI*, 2006.
[37] C. Sun *et al.*, "Dissecting pattern unlock: The effect of pattern strength meter on pattern selection," *Journal of Information Security and Applications*, 2014.
[38] A. Torralba and A. Oliva, "Depth estimation from image structure," *IEEE PAMI*, 2002.
[39] S. Uellenbeck *et al.*, "Quantifying the security of graphical passwords: The case of android unlock patterns," in *CCS '13*.
[40] E. von Zezschwitz *et al.*, "Easy to draw, but hard to trace?: On the observability of grid-based (un)lock patterns," in *CHI '15*.
[41] R. Weiss and A. De Luca, "Passshapes: utilizing stroke based authentication to increase password memorability," in *Nordic Conference on Human-Computer Interaction: Building Bridges*, 2008, pp. 383–392.
[42] Y. Xu *et al.*, "Seeing double: Reconstructing obscured typed input from repeated compromising reflections," in *CCS '13*.
[43] M. H. Yang *et al.*, "Extraction of 2d motion trajectories and its application to hand gesture recognition," *IEEE PAMI*, 2002.
[44] G. Ye, Z. Tang, D. Fang, X. Chen, K. I. Kim, B. Taylor, and Z. Wang, "Cracking android pattern lock in five attempts," in *The Network and Distributed System Security Symposium (NDSS)*, 2017.
[45] Q. Yue *et al.*, "Blind recognition of touched keys: Attack and countermeasures," *arXiv preprint arXiv:1403.4829*, 2014.
[46] J. Zhang *et al.*, "Privacy leakage in mobile sensing: Your unlock passwords can be leaked through wireless hotspot functionality," *Mobile Information Systems*, 2016.
[47] L. Zhang *et al.*, "Kaleido: You can watch it but cannot record it," in *MobiCom '15*.