2018 IEEE Symposium on Security and Privacy

EyeTell: Video-Assisted Touchscreen Keystroke Inference from Eye Movements

Yimin Chen*, Tao Li*, Rui Zhang†, Yanchao Zhang*, and Terri Hedgpeth*
*School of Electrical, Computer and Energy Engineering (ECEE), Arizona State University
†Department of Computer and Information Sciences, University of Delaware
{ymchen, tli}@asu.edu, ruizhang@udel.edu, {yczhang, terrih}@asu.edu

***Abstract*—Keystroke inference attacks pose an increasing threat to ubiquitous mobile devices. This paper presents EyeTell, a novel video-assisted attack that can infer a victim's keystrokes on his touchscreen device from a video capturing his eye movements. EyeTell explores the observation that human eyes naturally focus on and follow the keys they type, so a typing sequence on a soft keyboard results in a unique gaze trace of continuous eye movements. In contrast to prior work, EyeTell requires neither the attacker to visually observe the victim's inputting process nor the victim device to be placed on a static holder. Comprehensive experiments on iOS and Android devices confirm the high efficacy of EyeTell for inferring PINs, lock patterns, and English words under various environmental conditions.**

I. Introduction

Keystroke inference attacks pose an increasing threat to mobile devices which have penetrated into everyday life. In a typical attack scenario, a victim types on the soft keyboard of his[1] smartphone or tablet in an insecure public environment such as a public library, a coffee shop, or a train. The attacker tries to infer the victim's keystrokes in order to obtain sensitive information such as the victim's device passwords, web account passwords, or even emails. Based on the inferred keystrokes, the attacker can proceed to launch further attacks. One example is that the attacker can use the inferred password to pass the authentication system of the victim's device. The severe security and privacy implications make keystroke inference a very active research topic in mobile device security.

Many keystroke inference attacks rely on analyzing a video recording the victim's typing process. They require that either the recorded video capture the victim's typing process with little or no visual obstruction [1]–[9] or the device be placed on a static holder [9]. Given the video recording, the attacker infers keystrokes by analyzing touchscreen reflection [6], spatial hand dynamics [7], relative finger movements on the touchscreen [8], or the backside motion of the device [9]. While these attacks have been demonstrated quite effective, their strong assumptions may not always hold in practice.

In this paper, we report the design and evaluation of EyeTell, a novel video-assisted keystroke inference attack that can infer a victim's keystrokes on his touchscreen device from a video capturing his eye movements. EyeTell is inspired by the observation that human eyes naturally focus on and follow the keys they type such that a typing sequence on a soft keyboard results in a unique gaze trace of continuous eye movements. Under EyeTell, the attacker records a video of the victim's eye movements during his typing process and then extracts a gaze trace. By analyzing the gaze trace, the attacker can infer the victim's input with high accuracy.

Although conceptually intuitive, EyeTell faces three main design challenges. First, it needs to extract a gaze trace from the recorded video without any prior information about the victim (e.g., what his eyes look like). Second, the gaze trace is usually very noisy, making it very difficult to recover the correct typing sequence. Third, the gaze trace does not tell the exact number of keystrokes on the soft keyboard. To tackle the first challenge, we explore a user-independent model-based gaze tracking method [10]. To deal with noisy gaze traces and accommodate unknown keystroke counts, we develop a novel decoding algorithm to rank all possible typing sequences and finally output the ones with high rank.

Our contributions in this paper are summarized as follows.

- We propose EyeTell, a novel video-based attack that can infer a victim's keystrokes on a touchscreen device from a video capturing his eye movements. In comparison with prior work [1]–[9], EyeTell requires neither the attacker to visually observe the victim's typing process nor the victim device to be placed on a static holder. Therefore, EyeTell is more practical, sneaky, and launchable from a large distance, thus posing a more serious threat to user privacy.
- We prototype and evaluate EyeTell through experiments on both iOS and Android devices, which involve the PIN, pattern-lock, and alphabetical soft keyboards. We show that EyeTell can identify the top-5, top-10, and top-50 likely PINs that must contain a target 4-digit PIN with probabilities up to 65%, 74%, and 90%, respectively. Similarly, EyeTell can output the top-5, top-10, and top-50 possible lock patterns that must include a target Android lock pattern with probabilities up to 70.3%, 75.3%, and 85.1%, respectively. In addition, EyeTell can identify the top-5, top-10, top-25, and top-50 likely words that must include a target word with probabilities up to 38.43%, 63.19%, 71.3%, and 72.45%, respectively.
- We point out future directions to improve EyeTell and also possible countermeasures. Although currently EyeTell works only under a short recording distance and a small recording angle, we believe that the adoption of better optics and eye tracking techniques can readily relieve such limitations.

[1]No gender implication.

DOI 10.1109/SP.2018.00010

II. Related Work

In this section, we discuss the prior work most related to EyeTell in two research directions: keystroke inference attacks and eye-tracking-related security implications.

A. Keystroke Inference Attacks

Prior keystroke inference attacks can be broadly classified into video-based, sensor-based, and WiFi-based attacks.

1) Video-based attacks: In this category, the attacker uses a recorded video to infer keystrokes. Early work targets physical keyboards. For instance, Backes *et al.* [1], [2] recovered the content on a computer screen from its reflections on nearby objects such as glasses and tea pots. As another example, Balzarotti *et al.* [3] inferred the keystrokes by characterizing the light diffusion around the keyboard in the video recording. This work [3] requires the attacker to directly video-record the victim's finger typings on the physical keyboard.

More recent research along this line targets soft keyboards on ubiquitous touchscreen mobile devices. In [4], Maggi *et al.* tried to recover keystrokes from key magnifications on the touchscreen. In [5], Raguram *et al.* inferred keystrokes from the touchscreen's reflection on the victim's sunglasses. In [6], Xu *et al.* extended the attack in [5] to recover keystrokes from double reflections of the touchscreen. In [7], Yue *et al.* inferred keystrokes by exploiting the homographic relationship between captured images and a reference image of a soft keyboard. Similar homographic relationship was also used in [8] by matching finger movements. In [9], Sun *et al.* showed that the keystrokes can be actually inferred from the motion of a tablet's backside. All these attacks require the attacker to record a video capturing at least part of the victim's typing process or device backside, so they do not work if no such video is available. For example, the surrounding environment may prevent the attacker from having an unobstructed, stealthy view of the victim's typing process.

In contrast, EyeTell requires no unobstructed view of the victim's device or typing process and only needs the attacker to record the victim's eye movements during the typing process. When a user types, he usually holds his device in one hand or places it on a table or his knee. This means that his eyes are normally at much higher positions than his device during the typing process. So it is much easier and more sneaky to video-record the user's eye movements from a distance than ti video-record his device motion or typing process. EyeTell is thus applicable to much wider contexts.

2) Sensor-based attacks: In this category, the attacker uses on-board sensor data to infer a victim's keystrokes. In [11], [12], it was shown that the accelerometer data of a mobile device can be used to infer the victim's password. Subsequently, keystrokes were inferred in [13], [14] by combining both accelerometer and gyroscope data. In [15], [16], the authors exploited microphones and front cameras for keystroke inference. In comparison with video-based attacks (including EyeTell), these attacks require the attacker to acquire sensor data from the victim device through either malware infection or unprotected data transmissions. Such assumptions may not always hold in reality.

There is also work on using device sensors as the side channels to infer keystrokes of nearby physical keyboards. In [17]–[19], keystrokes on a physical keyboard were recovered through analyzing the acoustic emanations of the keyboard recorded by a nearby malicious microphone. In [20], [21], keystrokes were inferred by analyzing the time difference of arrival of acoustic signal recordings. In [22], Marquardt *et al.* used the accelerometer on a smartphone to measure the vibration induced by a nearby physical keyboard for keystroke inference. In [23], Liu *et al.* inferred keystrokes by exploiting the accelerometer data of a smartwatch worn by the victim while he typed. Similar to sensor-based attacks [11]–[16], these schemes [17]–[23] assume that the attacker can obtain sensor data from the victim device or that sensor data can be collected by other devices close to the physical keyboard. By comparison, EyeTell has no such restriction and can be launched from a larger distance.

3) WiFi-based attacks: In this category, the attacker infers a victim's keystrokes from recorded channel state information (CSI). The idea is that different keystrokes lead to distinct changes in wireless channels and the corresponding CSI. It has been shown that CSI information can be exploited to infer a victim's keystrokes on a physical keyboard [24], or a soft keyboard [25], or a pattern lock keyboard [26]. All these attacks are user-dependent and require the attacker to first obtain the victim's data with known labels to train a classifier. In addition, they cannot tolerate any change in the surrounding environment other than the victim's hand or finger movement. Furthermore, the distance between the WiFi transmitter and receiver, the orientation of the victim device, and the victim's typing gestures were all fixed in the experiments. These shortcomings limit the applicability of WiFi-based keystroke inference attacks in practice.

B. Eye-Tracking-Related Security Implications

Considering eye tracking as an input method for user-device interaction, researchers have proposed to use it for user authentication and inferring user input.

1) User authentication: In the early days, researchers tried to use eye movement as a biometric identifier for user authentication. In [27], the authors put forward this idea and evaluated the identification rate among users. In [28], [29], the authors proposed novel features extracted from eye movements and designed specific stimulus to enhance the performance.

More recent research in this line mainly focuses on designing novel challenge-and-response schemes for user authentication in a contactless manner. The key motivation is that eye tracking as an input method is more secure against shoulder-surfing attacks, besides novel two-factor authentication [30] and anti device-theft [31] schemes. For example, the authentication systems in [32]–[35] ask a user to follow moving objects on the screen, draw pre-selected shapes, perform eye gestures to input PIN passwords, etc.

2) Inferring user input: There are few efforts to work on inferring user inputs on device touchscreen by exploiting eye tracking as a side channel. In [36], the authors pointed out that the victim's eyes would follow his finger movements on the touchscreen of mobile device, which may leak his inputs. To show such feasibility, they mannually analyzed the images

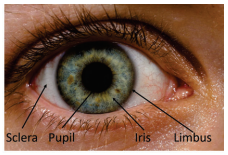


Fig. 1. Anterior segment of a human eye [37].

taken by the front camera of the victim's device to infer his input digits. Through a small scale of experiments (three participants and nine trials in total), they obtained an accuracy result of around 67% on PIN keyboard.

Compared with the above work, EyeTell exhibits two main differences. First of all, EyeTell works on a more challenging scenario of inferring user keystrokes on mobile device touchscreen, while most user authentication schemes based on eye tracking aim at much larger screens such as TV. Furthermore, these schemes were engineered in a way that their eye tracking module can obtain a user's eye trace easily and effectively. On the contrary, EyeTell can only obtain a much noisier eye trace due to two reasons: (1) the attacker does the video recording from a distance, and the victim's eye movements on a mobile touchscreen is much more subtle. Secondly, EyeTell (2) involves a set of tools to infer user inputs and comprehensive investigations on different types of soft keyboards to better evaluate its security and privacy impacts.

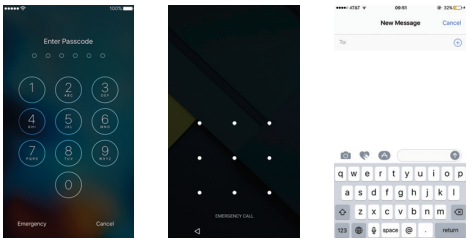
III. Background on Video-Based Gaze Tracking

EyeTell is based on the intuition that a victim's gaze trace can reveal his typing sequence on a soft keyboard. Fig. 1 depicts the anterior segment of a human eye. According to the definition in [38], the gaze actually refers to the gaze direction. We now briefly introduce the background of video-based gaze tracking, which is used in EyeTell to extract gaze traces.

Gaze tracking refers to the techniques that determine the gaze direction of the eyes. Gaze tracking has numerous applications such as human attention analysis and gaze-based human-computer interfaces. So far video-based gaze tracking is most popular because it achieves high accuracy without requiring the target to wear any special device.

There are mainly two types of video-based gaze tracking methods: feature-based and appearance-based [38], [39]. Feature-based methods use local features such as contours, eye corners, and reflections from the eye image for gaze estimation. In contrast, appearance-based methods directly use the content of the eye image as input to estimate the gaze direction instead of extracting any local feature.

Feature-based methods can be further divided into interpolation-based and model-based methods according to how the features are used. Interpolation-based methods commonly assume that the mapping between the image features and gaze can be modeled as a parametric form such as a polynomial or nonparametric one like a neural network. In contrast, model-based methods directly calculate the gaze from the image features based on suitable geometric models of the human eye. In this paper, we adopt the model-based gaze tracking method in [10] due to its advantage that the attacker does not need to obtain any training data about the victim prior to the attack. Other model-based methods can be used in EyeTell as well if they require no training data.



(a) PIN (b) Pattern lock (c) Alphabetical

Fig. 2. Three representative soft keyboards.

IV. Adversary Model

We consider a victim using a mobile touchscreen device such as a smartphone or tablet. Assume that the victim holds the device right in front of himself and types on the touchscreen soft keyboard. Such scenarios are very common in practice. For example, the victim may use his mobile device at his workplace or wait in line at a coffee shop. We assume that the victim is alert to conventional shoulder-surfing attacks in the sense that the attacker cannot get too close to the victim when he types on the device.

We consider an attacker who aims to infer the typed sequence on the victim device, which could be PINs, lock patterns, words, or sentences. We assume that the attacker can use a COTS smartphone, digital camera, or camcorder to record the victim's eyes during his typing process, possibly from a long distance. However, the attacker cannot obtain any IMU sensor (accelerometer, gyroscope, microphone, etc.) data by installing malware such as Trojans or malicious web scripts on the victim device. Different from prior work, we assume that the attacker can see neither the touchscreen or backside of the victim device nor the victim's hand movements during his typing process. Under these assumptions, existing video-based [1]–[9], [40] and sensor-based [11]–[16], [18]–[20], [22] keystroke inference attacks no longer work.

V. EyeTell Design

In this section, we give an overview of EyeTell and then detail its design. For convenience only, we assume the victim device to be a smartphone throughout the illustration, though EyeTell can work with any mobile touchscreen device.

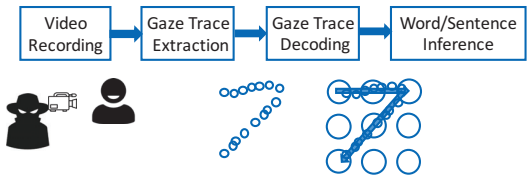


Fig. 3. Workflow of EyeTell.

A. Overview

EyeTell is designed to infer the sensitive inputs on the soft keyboard from the video of the victim's eye movements while he types. The high-level design of EyeTell is shown in Fig. 3, which consists of the following four steps.

(1) Video Recording. We first record a video capturing the victim's eyes during his inputting process using a COTS camcorder. As mentioned in Section IV, we assume that

(a) Side view (b) Attacker's view

Fig. 4. Typical setup for video recording.

neither the touchscreen nor the victim's hand movement can be directly seen from the video. In addition, we do not assume that the smartphone is fixed on a device holder or that the video can capture its backside.

(2) Gaze Trace Extraction. We adapt user-independent gaze tracking [10] to extract the gaze direction from each frame of the recorded video and then combine the directions to obtain a complete gaze trace. In particular, we detect the two eyes in each video frame and then the limbus for each eye, from which we finally estimate the corresponding gaze direction. Due to the noisy and unstable nature of the extracted gaze trace, we further apply outlier detection and low-pass filtering to obtain a cleaner gaze trace.

(3) Trace Decoding. In this step, we design a novel decoding algorithm to match the gaze trace extracted in Step 2 into a set of candidate typing sequences on the soft keyboard. Fig. 2 shows the soft keyboards we investigate in this paper, including the pattern-lock keyboard on Android and the PIN and alphabetical keyboards on iOS. For PIN or pattern-lock inference, each candidate typing sequence corresponds to one or several PINs or lock patterns. For word or sentence inference, an additional step is taken to select meaningful results with the assistance of a dictionary. The decoding algorithm must adapt to different inference scenarios where the attacker's prior information may vary a lot. For example, the attacker knows that a PIN must consist of four or six digits, but he knows very little to none about which word the victim is likely to input before doing word inference.

(4) Word/Sentence Inference. Finally, we select the possible words by considering meaningful alphabetical combinations using a dictionary. We also explore the linguistic relationship between adjacent English words to further infer sentences.

We detail each step above in what follows.

B. Video Recording

In this step, we want to obtain a video of the victim's eyes when he types on the soft keyboard of the smartphone. Fig. 4 shows a typical setting of video recording in our experiments. We ask the participants to sit on the chair and input on a smartphone. A Panasonic HCV7000 camcorder is used to record videos. Using a COTS camcorder can show that EyeTell is low-cost, convenient, and stealthy to launch. In our studies, we find that the following factors affect the result of our gaze tracking algorithm.

Image resolution. The resolution of the recorded video affects eye and limbus detection and therefore the extracted gaze. In the experiments, we always stick to the highest resolution of the camcorder, i.e., each video frame is of 1920×1080 pixels.
Video frame rate. Due to the noisy and instable nature of the extracted gaze trace, we need to collect more sudden changes of the user's eye movement and thus desire a higher video frame rate. In the experiments, we choose the frame rate as 60 fps, which is the highest frame rate supported by our camcorder. Our attack can be more effective if a camcorder supporting higher frame rates is available.
Light condition. The light condition in the environment may also affect the inference result, as the imaging sensor of the camcorder generates larger noise in low-illumination environments and thus produces a polluted gaze trace.
Recording angle. We define the recording angle as the angle between the plane formed by the victim and his smartphone and the plane formed by the victim and the attacker's camcorder. Our current EyeTell implementation requires that the camcorder be placed in the same plane as the victim and his smartphone, typically as shown in Fig. 4. Therefore, our default recording angle is zero degree. We believe that this assumption is fairly easy to achieve in practice with advanced camcorders and can be relieved if more sophisticated gaze tracking algorithms are available.

After video-recording the victim's eye movement, we manually crop the beginning and ending part of the video such that the remaining part contains only the typing process. For example, the video only contains the process of the victim inputting four digits or drawing a pattern on the smartphone.

C. Gaze Trace Extraction

There are three steps in gaze trace extraction: eye localization, limbus detection, and gaze trace estimation.

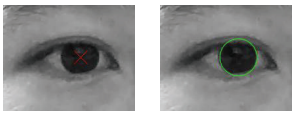
1) Eye detection: EyeTell detects the victim's eyes in each frame through a two-step approach. We first search for a pair of eyes within the entire frame in a coarse-grained manner. Once a rough region is obtained, we further refine the detected eye region and then calculate the accurate eye positions.

In the first step, we use a Haar-like feature-based cascade classifier [41] to detect possible eye regions and always select the first output as the candidate eye region. We then segment the candidate eye region into two area-of-interests (AOIs), one for each eye. The cascade classifier [41] is very efficient and also user-independent, but it may still incur false positives that the candidate region is not the eye region. For example, a rectangular area enclosing the user's clothes may be misclassified as the eye region.

We use two tricks to reduce such false positives. First, we require that the size of the detected eye region be above a minimum threshold. In our implementation, we set the threshold to be 80×40 pixels, which has been shown valid for our video recording setting. Second, we calculate a similarity score between the detected eye region and a reference region, which is the eye region successfully detected in a different frame of the same video. In particular, we resize the candidate eye region to the same size as the reference region and then normalize the pixel values of both regions. After normalization, we calculate a pixel-level similarity score for the same pixel in the two regions, which is the ratio between the absolute difference of the two pixel values and their sum. After that, the similarity score of the two eye regions is calculated as the average pixel-level similarity score across the entire eye region. The smaller the similarity score, the more similar the two eye regions. In our implementation, we use an empirical threshold

of 0.8 to filter out possible false positives of eye regions and manually check them. The threshold needs to be adjusted in practice: a small threshold may result in many possible false positives and thus increase the demand for manual checking, and vice versa. If a detected candidate eye region is indeed a false positive, we manually assign a correct rectangular region enclosing both eyes as the input to the cascade classifier, which leads to correct eye detection in practice.

In the second step, EyeTell uses a shape-based approach to refine the two AOIs by exploring the predicted dark circular appearance of eye features [42]. Specifically, we define the center of a circular pattern as the point where most image gradient vectors intersect. Then we search for the optimal point by using an objective function that measures how well the gradient vectors and the eye center displacement vectors are aligned. Moreover, considering the fact that the eye center usually appears darker than other areas in the eye, we attach each point with a weight of its inverse intensity in the objective function. Once the optimal points of the two AOIs (i.e., the two eye centers) are located, we refine the positions of two AOIs in the frame and then resize them to a fixed ratio. The resizing operation can minimize the areas of the two AOIs while maintaining important eye features within them. The red cross in Fig. 5(a) denotes the detected eye center.



(a) Eye center (b) Fitted limbus

Fig. 5. Examples of our detected eye center and limbus.

2) Limbus detection: In this step, EyeTell determines the elliptical outline of the limbus from each identified AOI by first identifying a set of possible limbus edge points and then fitting an ellipse model from those edge points [10]. In contrast to other popular limbus detection methods [43], [44], this method does not rely on any pre-defined threshold, which allows EyeTell to reliably detect the limbus regardless of eye appearance, users, and lighting conditions. Moreover, it can detect the limbus from out-of-focus images because it does not depend on the existence of very strong edge. We illustrate this process in what follows.

Since limbus edge points are part of the edge, we search for them by analyzing the radial derivatives within each AOI. Specifically, we transform a given AOI into the polar form and then calculate the vertical derivative of each pixel. In our implementation, we select the pixel with the largest radial derivative in each column as the limbus edge point.

Special attention is paid to non-edge points that are incorrectly detected as edge points, which occurs if the radial derivatives of non-edge points are larger than those of true limbus edge points. According to our experimental observations, we use the following process to filter out as many such non-edge points as possible. First, we notice that nearby light sources can leave specularities on the cornea. The pixels within these specularities can have very large radial derivatives and thus be incorrectly identified as limbus edge points. To deal with this case, we compare each pixel value with a threshold, e.g., 150 (the pixel value is between 0 and 255), to identify a set of possible specularities and then inpaint these small connected regions. The effective threshold depends on the recording environment, which we choose empirically. Second, we observe that the upper eyelid may cover part of the iris and therefore lead to incorrect limbus edge points. To cope with this case, we use three points, two eye corners and the iris-eyelid boundary point right above the given eye center, to fit a parabola to approximate the upper eyelid and then discard the points that fall outside the parabola.

Finally, we fit an ellipse model from the set of edge points using the iterative method in [45]. In each iteration, a minimum number of edge points are randomly selected from available ones to fit an ellipse model through a least-square approach. Then a support function is calculated to evaluate how fit the model is to the entire set. We use the support function in [45] that measures how well the geometric gradients of the fitted ellipse model align with the image gradients. Fig. 5(b) denotes a detected limbus in our experiment.

3) Gaze trace estimation: In this step, we estimate one gaze point from each frame to obtain a complete gaze trace from the entire video. To do so, we use the detected eye centers and limbus in the 2D domain to recover the corresponding 3D eye centers and optical axes. We then estimate the gaze point as the intersection between the optical axes and the virtual 3D screen. We further refer to the gaze point as point-of-gaze (PoG), which can be simply denoted by a vector $[x, y]^T$. Here x and y correspond to the coordinates of the PoG along x and y axis on the screen, respectively. For the benefit of better readability, here we omit the detailed mathematical deduction to calculate a PoG. For more details, please refer to Appendix A.

By calculating the PoG for each eye in each frame, we obtain two complete gaze traces from the recorded video, denoted by $\Psi^l = (\text{PoG}_1^l, \ldots, \text{PoG}_{n_f}^l)$ and $\Psi^r = (\text{PoG}_1^r, \ldots, \text{PoG}_{n_f}^r)$ for the left and right eyes, respectively, where n_f is the number of frames in the video.

Since the extracted gaze traces are usually noisy and unstable, we apply outlier detection and filtering to enhance their quality. To detect possible outliers, we check the distance between the two estimated eye centers in each frame. If the distance in the ith frame is larger than an anatomical threshold, e.g., 80 mm, we consider that at least one PoG between PoG_i^l and PoG_i^r is an outlier. In this case, we replace the PoG that yields a larger PoG change between adjacent frames with the one that leads to a smaller change.

In the subsequent filtering step, we first obtain a raw gaze trace $\Psi = (\text{PoG}_1, \ldots, \text{PoG}_{n_f})$ by taking the average of the left and right gaze traces, where

$$\text{PoG}_i = \frac{\text{PoG}_i^l + \text{PoG}_i^r}{2}, \tag{1}$$

for all $i \in [1, n_f]$. We then apply a triangular kernel [46] to Ψ, which assigns linear weights to each PoG in the time order. Specifically, for each $j \in [1, n_f]$, we calculate

$$\overline{\text{PoG}}_j = \frac{\sum_{i=j-N_1+1}^{j} i \times \text{PoG}_i}{\sum_{i=1}^{j} i}, \tag{2}$$

where N_1 is empirically set to 5 in our implementation. The final gaze trace for keystroke inference is $\overline{\Psi} =$

$(\overline{\text{PoG}}_1, \ldots, \overline{\text{PoG}}_{n_f})$. For convenience, we call each element in $\overline{\Psi}$ a PoG as well.

D. Trace Decoding

In this step, EyeTell decodes the gaze trace $\overline{\Psi}$ to obtain some candidate input sequences on the touchscreen. Depending on the soft keyboard the victim types on, the candidate input sequence may correspond to a lock pattern, a PIN, a word, or a sentence. Generally speaking, trace decoding is done in four steps. First, we identify the turning points in a gaze trace and then divide the whole trace into a sequence of segments, each corresponding to a sudden change in the PoG. Second, we convert each segment into a small set of candidate vectors. Third, given the sequence of segments and their corresponding candidate vector sets, we enumerate all possible combinations of candidate vectors. For each possible combination of candidate vectors, we traverse the soft keyboard to check whether or not the combination can be mapped into a valid input sequence. Finally, we rank all the valid input sequences according to certain heuristic rules and generate a final set of candidate input sequences for a given gaze trace. In what follows, we use the pattern-lock keyboard as the example to illustrate trace decoding and then point out the difference when applying EyeTell to PIN and alphabetical keyboards.

1) Trace segmentation: We first apply a moving average filter to further smooth the gaze trace extracted in the last step, as it does not exhibit any clear pattern for segmentation. The length of the moving window has a direct impact on the segmentation performance. On the one hand, if the window is too short, the filtered gaze trace is not sufficiently smooth. On the other hand, if the window is too long, some sudden changes may be buried, resulting in some undetectable turning points. We empirically set the moving-window length to 10 based on analyzing our experiment data.

We then segment the smoothed trace by identifying the turning points that separate adjacent segments. For simplicity, we abuse the notation by letting $\overline{\Psi} = (\overline{\text{PoG}}_1, \ldots, \overline{\text{PoG}}_{n_f})$ denote the smoothed trace as well. Suppose that $\overline{\Psi}$ consists of two segments as an example. In the ideal case, the points in each segment lie in a straight line, and the intersection of the two lines is the turning point between two segments. Based on this observation, we first estimate the moving direction of each PoG (or element) in $\overline{\Psi}$. Let $\overrightarrow{\text{PoG}}_{i,j} = \overline{\text{PoG}}_j - \overline{\text{PoG}}_i$ be the vector for $\forall i, j \in [1, n_f]$. For each $\overline{\text{PoG}}_i$ and the next N_2 PoGs (i.e., $\{\overline{\text{PoG}}_j\}_{j=i+1}^{i+N_2-1}$), we compute N_2 vectors $\{\overrightarrow{\text{PoG}}_{j,i}\}_{j=i+1}^{i+N_2-1}$, where N_2 is a system parameter empirically set to 5 in our experiment. We further calculate

$$\overrightarrow{\text{PoG}}_i = \frac{\sum_{j=i+1}^{i+N_2-1} \overrightarrow{\text{PoG}}_{i+1,i}}{N_2} \tag{3}$$

as the moving direction of $\overline{\text{PoG}}_i$. Let $\theta_i \in [-\pi, \pi)$ denote the angle of $\overrightarrow{\text{PoG}}_i$. We can then obtain a sequence of angles $\theta_1, \ldots, \theta_{n_f-N_2+1})$ for the gaze trace $\overline{\Psi}$. For every N_3 adjacent PoGs such as $\{\text{PoG}_j\}_{j=i}^{i+N_3-1}$, we consider them in the same segment if and only if

$$\frac{\sum_{j=i}^{i+N_3-1} |\theta_{j+1} - \theta_j|}{N_3} \leq \phi_1 \ ,$$

where N_3 and ϕ_1 are both system parameters that are empirically set to 5 and $\frac{\pi}{4}$ in our experiment, respectively.

We then search for turning points as follows. Starting from $i = 1$, we find the smallest i' such that $\frac{\sum_{j=i'}^{i'+N_3-1} |\theta_{j+1} - \theta_j|}{N_3} > \phi_1$ and then regard $\overline{\text{PoG}}_{i'}$ as the ending point of the first segment. Starting from i', we proceed to find the smallest i'' such that $\frac{\sum_{j=i''}^{i''+N_3-1} |\theta_{j+1} - \theta_j|}{N_3} \leq \phi_1$ and then consider $\overline{\text{PoG}}_{i''}$ as the starting point of the second segment. After determining i' and i'', we search between i' and i'' to find i_1 with the largest $\frac{\sum_{j=i_1}^{i_1+N_3-1} |\theta_{j+1} - \theta_j|}{N_3}$ and consider $\overline{\text{PoG}}_{i_1}$ as the turning point between the first two segments.

Repeating the above process, we can identify all the turning points in the gaze trace. Suppose that n_t turning points are found in total. Combined with the first and last PoGs of the gaze trace, the total $n_t + 2$ points correspond to $n_t + 1$ segments. Denote the $n_t + 2$ points by $\{\text{TP}_i\}_{i=1}^{n_t+2}$, where TP_1 and TP_{n_t+2} correspond to the first and last PoGs of the gaze trace, respectively, and TP_i ($\forall i \in [2, n_t + 1]$) are the turning points. In the remainder of the paper, we denote the number of segments by n_s. Therefore, $n_s = n_t + 1$. The final output of trace segmentation comprises n_s segments, each of which can be represented by its length and angle. Specifically, assuming $\text{TP}_i = [x_i, y_i]^T$, the i-th segment can be characterized by $[x_{i+1} - x_i, y_{i+1} - y_i]^T$ for all $i \in [1, n_s]$.

We use the example in Fig. 6 to shed more light on trace segmentation. Specifically, Fig. 6(a) shows a two-segment gaze trace to decode; Fig. 6(b) shows the gaze trace after applying the moving average filter; Fig. 6(c) shows the angles of the PoGs on the trace; and Fig. 6(d) shows the ending point of the first segment and the starting point of the second segment.

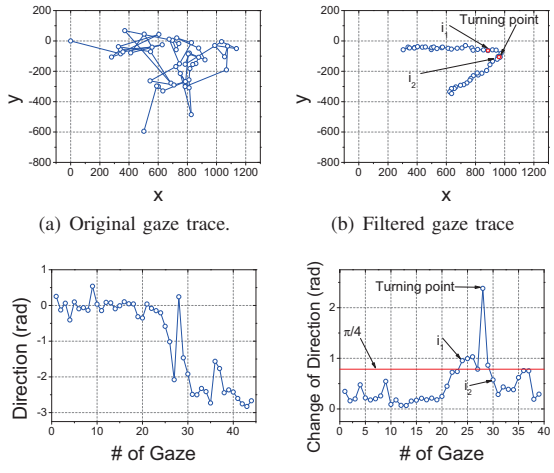
N_2, N_3, and ϕ_1 depend on the factors such as frame rate, signal-to-noise ratio (SNR) of the video, etc. For example, N_2 and N_3 increase with the frame rate and decrease with SNR generally. In this paper, we choose these parameters empirically by experimenting them with a small portion of data and observing the results of segmentation. In practice, we believe that they need to be adjusted or trained in different scenarios.

TABLE I. MAPPING BETWEEN ALPHABETICAL AND QUASI-PIN KEYBOARDS DEPICTED IN FIG. 8(A).

1	q,w,e,r	2	t,y	3	u,i,o,p
4	a,s,d	5	f,g,h	6	j,k,l
7	z,x	8	c,v,b	9	n,m

2) Decoding segment: We observe that only a limited number of gaze segments are permissible on any typical soft keyboard (PIN, pattern lock, and alphabetic), which are referred to as legitimate segments hereafter. In this step, we decode a given gaze segment into a small set of candidate legitimate segments on the pattern-lock keyboard. This is done by calculating the Euclidean distances between the given segment and all legitimate ones and then selecting those with shorter distances as the candidates.

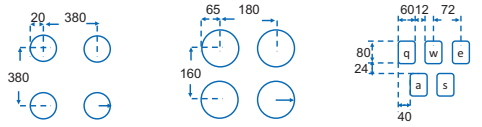
Let us first look into more details of the pattern-lock keyboard and its corresponding legitimate segments. Fig. 7 depicts the dimensions of the pattern-lock keyboard layout on



(a) Original gaze trace. (b) Filtered gaze trace

(c) Direction (d) Change of direction

Fig. 6. An illustration for trace dividing.



(a) Pattern-lock (b) PIN (c) Alphabetical

Fig. 7. Measurement of the three keyboards. The unit is pixel.

a Google Nexus 6 smartphone with Android 5.1.1, including the radius of nine white circles, the horizontal gap between two neighboring circles, and the vertical gap between two neighboring circles. All these dimensions are also listed in Table XI in Appendix B. We further plot all the 24 possible segments on the pattern-lock keyboard in Fig. 9 and then calculate their lengths and angles. The 24 segments lead to five lengths and 16 angles in total.

We first normalize the segment length to facilitate segment decoding. As we can see from Fig. 9, the minimum segment length is 1, so we try to make the minimum normalized segment length be 1 as well via the following approach. First, we sort the segments in the ascending order of their lengths. Let $L_{\max}$ denote the longest segment length. Then we select the shortest segment and calculate the ratio ρ between $L_{\max}$ and its length. According to Table XII, the length ratio between any two legitimate segments is no larger than $2\sqrt{2}$. Therefore, we compare ρ to a threshold $\rho_{\max}$. If $\rho \leq \rho_{\max}$, the currently selected segment is used for normalizing all the segments. Otherwise, we select the next shortest segment, calculate a new ρ, and compare it to $\rho_{\max}$. This process ends until $\rho \leq \rho_{\max}$. The currently selected segment is the one used for length normalization, and the normalized segment lengths smaller than 1 are all set to 1. $\rho_{\max}$ should be larger than $2\sqrt{2}$ to accommodate the noisy and instable nature of the gaze trace.

Next, we compute the Euclidean distance between each normalized segment and each legitimate segment in Fig. 9. Suppose that we look for η candidate legitimate segments for each normalized segment. Those leading to the top-η shortest Euclidean distances are selected as the candidates. Intuitively speaking, the larger η, the more likely that the correct legitimate segment is included in the candidate set, the less pinpointing capability the attacker has, and vice versa.

The final output in this step corresponds to n_s candidate sets, each corresponding to a gaze trace segment. We denote the candidate set for the i-th trace segment by $\mathcal{N}_i$ ($\forall i \in [1, n_s]$), which contains η legitimate segments in the ascending order of their Euclidean distances with the i-th trace segment.

3) Candidate lock patterns: Now we generate the candidate lock patterns for a gaze trace. Let $c_1, \ldots, c_9$ denote the nine white circles of a pattern lock keyboard, as shown in Fig. 8(b). By setting the center coordinate of c_1 to $(0, 0)$, we derive the center coordinates of other circles and list them in Table II. Since the gaze trace comprises n_s segments with each having η candidate legitimate segments, a candidate lock pattern can be represented by a row vector $p = [p_1, \ldots, p_{n_s+1}]$, where p_i refers to the i-th point that corresponds to one of $c_1, \ldots, c_9$.

TABLE II. COORDINATES OF PATTERN-LOCK KEYBOARD DEPICITED IN FIG. 8(B).

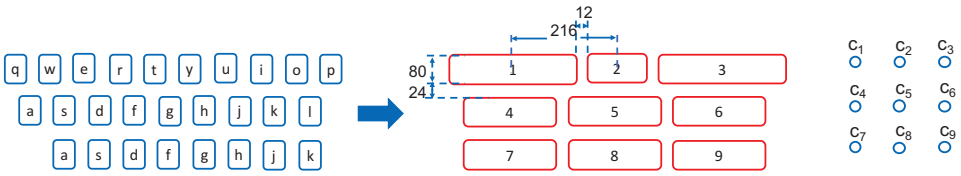
c_1	(0,0)	c_2	(1,0)	c_3	(2,0)
c_4	(1,0)	c_5	(1,1)	c_6	(2,1)
c_7	(2,0)	c_8	(2,1)	c_9	(2,2)

We generate the candidate lock patterns by considering each possible combination of n_s legitimate segments and then checking its feasibility by traversing on the pattern-lock keyboard. In each round, we select a random segment S_i among the η segments in $\mathcal{N}_i$ ($\forall i \in [1, n_t]$) to form a legitimate segment sequence $\{S_1, \ldots, S_{n_s}\}$. There are totally η^{n_s} rounds, each with a unique legitimate segment sequence. Assuming that the length and angle of S_i are l and α, respectively, we rewrite $S_i = (l\cos(\alpha), l\sin(\alpha))$. Given $\{S_1, \ldots, S_{n_s}\}$ and an arbitrary starting point $p_s \in \{c_i\}_{i=1}^{9}$, we can obtain a candidate lock pattern p, where $p_1 = p_s$ and $p_i = p_{i-1} + S_{i-1}$ ($\forall i \in [2, n_s + 1]$). We say that p is *feasible* if $p_i \in \{c_1, \ldots, c_9\}, \forall i \in [1, n_s + 1]$. There are nine possible choices for p_s, each corresponding to a candidate lock pattern. All the feasible lock patterns are then recorded.

An undesirable consequence of length normalization is that larger lock pattens may be mis-recognized as their shrunken versions. The example in Fig. 10 illustrates this aspect. The correct pattern in the example is $[c_1, c_3, c_7, c_9]$, and the gaze trace segments after length normalization are $\{(1, 0), (-1, 1), (1, 0)\}$. So the candidate lock patterns are $[c_1, c_2, c_4, c_5]$, $[c_2, c_3, c_5, c_6]$, $[c_4, c_5, c_7, c_8]$, and $[c_5, c_6, c_8, c_9]$, illustrated in Fig. 10. Our remedy for the issue is that if a legitimate segment sequence $\{S_1, \ldots, S_{n_s}\}$ can generate a feasible lock pattern, we double the length of each segment there and then check if the new sequence $\{\tilde{S}_1, \ldots, \tilde{S}_{n_s}\}$ can generate a feasible lock pattern or not, where $\tilde{S}_i = (2l\cos(\alpha), 2l\sin(\alpha))$. All such feasible lock patterns are recorded as well.

4) Ranking candidate lock patterns: The final step is to rank candidate lock patterns with three heuristics as follows.

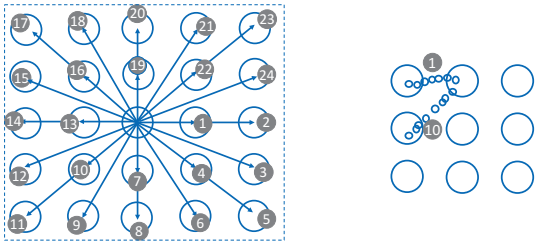
First, we introduce a row vector $r = (r_1, \ldots, r_{n_s})$, where $r_i \in [1, \eta]$ means that the r_i-th segment is chosen from $\mathcal{N}_i$ ($\forall i \in [1, n_s]$). Then we generate the η^{n_s} legitimate segment sequences based on r in the following order: $[1, 1, \ldots, 1, 1], [1, 1, \ldots, 1, 2], \ldots, [1, 1, \ldots, 1, \eta], [1, 1, \ldots, 2, 1]$, $[1, 1, \ldots, 2, 2], \ldots [1, 1, \ldots, 2, \eta], [1, 1, \ldots, 3, 1], \ldots, [\eta, \ldots, \eta]$.



(a) Transform alphabetical keyboard to quasi-PIN keyboard.

(b) Denotions in Table II.

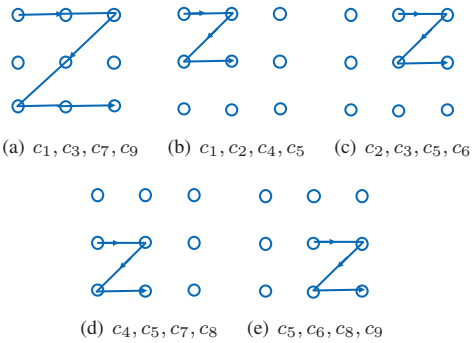
Fig. 8. Quasi-PIN keyboard.



(a) All possible segments.

(b) A pattern and its segments.

Fig. 9. Segments on pattern-lock keyboard.



(a) c_1, c_3, c_7, c_9 (b) c_1, c_2, c_4, c_5 (c) c_2, c_3, c_5, c_6

(d) c_4, c_5, c_7, c_8 (e) c_5, c_6, c_8, c_9

Fig. 10. Ambuguities due to normalization.

Recall that the earlier segments in each $\mathcal{N}_i$ have smaller Euclidean distances to the corresponding gaze trace segment than those of the later segments. Earlier legitimate segment sequences can thus produce higher ranked candidate lock patterns than later ones.

Second, the candidate lock patterns generated from the same legitimate segment sequence are ranked according to their starting points in the order of $c_1 > c_4 > c_7 > c_2 > c_5 > c_8 > c_3 > c_6 > c_9$. Such a heuristics is also adopted in [40].

Finally, the candidate lock patterns generated from an enlarged legitimate segment sequence have higher ranks than those from the original sequence. The intuition is that normal users tend to draw larger patterns.

TABLE III. HIDDEN KEYS ON PIN KEYBOARD.

(1,3)	2	(4,6)	5	(7,9)	8	(1,7)	4	(2,8)	5
(3,9)	6	(1,9)	5	(3,7)	5	(5,0)	8	(2,0)	5,8

5) PIN keyboard: Now we discuss how EyeTell applies to the PIN soft keyboard. Fig. 7 and Table XI (in Appendix B) show the dimensions of the PIN keyboard layout on a Google Nexus 6 with Android 5.1.1, including the radius of each key, the horizontal gap, and the vertical gap. We plot the 30 legitimate segments in Fig. 15 in Appendix B for lack of space. Note that users slide on the pattern-lock keyboard to draw a pattern but touch the keys on the PIN keyboard to input a PIN. A user may input the same key multiple times on the PIN keyboard, in which case there is little displacement in the corresponding gaze trace. Furthermore, a user may input three keys along the same direction sequentially, in which case the attacker does not know how many keys are touched. For example, the gaze traces for two different PINs (e.g., $[1, 4, 7, 9]$ and $[1, 7, 8, 9]$) can be very similar.

We then modify the process in Section V-D3 to generate candidate 4-digit PINs.

- If there are three trace segments, EyeTell directly generates candidate 4-digit PINs as in Section V-D3.
- If there are two trace segments, EyeTell first follows the process in Section V-D3 to generate candidate 3-digit PINs. We abuse the notation by letting a candidate PIN be denoted by a row vector p, in which each element is a key on the PIN keyboard. We have $p = [p_1, p_2, p_3]$ initially and then generate candidate 4-digit PINs as follows. First, we generate and record $[p_1, p_1, p_2, p_3]$, $[p_1, p_2, p_2, p_3]$, and $[p_1, p_2, p_3, p_3]$, as the user may type any key twice. Second, we consider the possible hidden keys between any two original keys. For example, if a possible hidden key p_h lies between p_1 and p_2, we consider and record $[p_1, p_h, p_2, p_3]$ as a candidate PIN as well. Table III shows the possible hidden keys on the PIN keyboard corresponding to each pair of original keys.
- If there is only one trace segment, EyeTell first follows the process in Section V-D3 to generate a 2-digit PIN denoted by $p = [p_1, p_2]$. Then we generate and record the candidate PINs as $[p_1, p_1, p_1, p_2]$, and $[p_1, p_2, p_2, p_2]$.

The above process can be easily extended to 6-digit PINs and omitted here for lack of space.

6) Alphabetical keyboard: Here we discuss how to adapt our algorithm to attack the alphabetical keyboard whose layout dimensions are given in Fig. 7 and Table XI. In contrast to the PIN keyboard, the alphabetical keyboard has more keys (26 instead of 10) and a smaller area (about 48% smaller), which poses a great challenge to keystroke inference. We tackle this challenge by first transforming the alphabetical keyboard into a quasi-PIN keyboard, as shown in Table I and depicted in Fig. 8(a). Then we generate candidate PINs on the quasi-PIN keyboard as in Section V-D3. Next, we produce a list of candidate words from candidate PINs and then use a dictionary to filter out non-existing words.

Keystroke inference on the quasi-PIN keyboard is even harder than that on the PIN keyboard. Specifically, the user may type the same key multiple times or hit some hidden keys on a segment, which is difficult for the attacker to identify. In addition, the attacker knows that the PIN to infer corresponds to 4 or 6 keys on the PIN keyboard, while he has no idea how many keys are contained in a PIN on the quasi-PIN keyboard because the corresponding word to infer may include an arbitrary number of letters. The situation becomes even worse because the same key or a hidden key may be typed multiple times. For example, the combinations of "ty", "er", "gh", and "ui" are quite common in English words.

We amend the process in Section V-D3 to increase the accuracy of inferring English words on the quasi-PIN keyboard and thus the alphabetical keyboard.

- We add a $(0, 0)$ segment into the set of legitimate segments. If a $(0, 0)$ segment is selected, the gaze trace stays on the same key, corresponding to the case that the victim inputs the same key repeatedly.
- Since it is unrealistic to consider all the possible lengths of the typed word, we only consider candidate words of $n_s + 1$ or $n_s + 2$ letters long for a given gaze trace of n_s segments.

As in [9], [22], [23], we refine the candidate words with the popular "corn-cob" dictionary [47] which is an on-line word list of 58,110 common English words.

Given a candidate PIN on the quasi-PIN keyboard, we generate a list of candidate words for the extracted gaze trace in the following two steps. First, we enumerate all the possible combinations of letters of the given PIN. Second, we search in the dictionary and add discoverable combinations into the list of candidate words. The complexity of such a process can be very high. For example, in our experiments, the number of possible PINs for a 13-letter word is in the order of 10^4, the number of possible combinations is in the order of 10^6 ($3^{13} = 1594323$), and the number of strings in the dictionary is 58,110. All these add up to a complexity of 10^{15}. To reduce the search complexity, we build a prefix tree of the "corn-cob" dictionary using trie structure [48] such that the search complexity within the dictionary is $\mathcal{O}(L)$, where L is the length of the given string.

VI. Performance Evaluation

A. Experiment Setup

1) User enrollment: We recruited 22 participants for the experiments, including 5 females and 17 males. Our experiment protocol was approved by the Institutional Review Board (IRB) at our institution and strictly followed in the experiments. Since the participants were only asked to input on smartphones, the experiments did not affect either them or people nearby at all. We only obtained the participants' oral consent because the IRB approved a waiver of the requirement to obtain written consent. All the participants were either graduate students in our lab or others we know in the same university. We did not reward them with any monetary compensation and only treated them to free snacks. Finally, all the recorded videos are stored in password-protected lab computers. As shown in Table IV, the number of participants in our evaluation is larger than those in our closely related work.

TABLE IV. Number of participants in related system evaluations.

System	[19]	[22]	[23]	[9]	[40]	EyeTell
Number of participants	N/A	N/A	5	4	10	22

2) Data collection: We used a Panasonic HCV700 camcorder for video recording in our experiment. This camcorder has a $21\times$ zoom len and can record 1080p60 HD videos. Two smartphone models were used in the experiments: Apple iPhone 6s with a $10.5\text{cm} \times 5.8\text{cm}$ screen size and Google Nexus 6 with a $12.3\text{cm} \times 7.5\text{cm}$ screen size.

A typical data collection process is as follows. A participant was asked to sit on a chair (illustrated in Fig. 4), hold a smartphone in front of herself/himself, and input on the touchscrren. The input can be a PIN on the PIN keyboard, a pattern on the pattern-lock keyboard, or an English word on the alphabetical keyboard. The participant was asked to input in her/his normal typing/drawing speed. We observed that the participant almost always kept her/his head relatively steady during each inputting process which was very short and less than 5 s in our experiments. Such relatively steady head positions are explored by almost all existing gaze tracking methods, including the one used in EyeTell. The following default settings were used, unless noted otherwise. The distance between the participant and camcorder was around 2 m. The participant, smartphone, and camcorder lay in the same plane. The resolution and frame rate of the camcorder were set as 1920×1080 and 60 fps, respectively. We also adjusted the zoom of the camcorder such that the captured face of the participant was focused and larger than 500×500 pixels.

In general, we conducted two sets of experiments: one without task randomization and the other with task randomization. The former involved 12 participants, each of whom performed experiments sequentially from one session to the next. For example, a participant first performed all experiments on inferring a single lock-pattern segment, then complete lock patterns, and so on. In contrast, the latter involved 10 participants, each of who was given randomly permuted tasks. As an example, a participant performed one trial on inferring a single lock-pattern segment, then two trials on complete lock pattens, then three trials on 4-digit PINs, and so on.

We use the experiment on inferring a single segment on the pattern-lock keyboard to examplify how we reduced the impact of fatigue. For this experiment, a participant was asked to draw each segment in Table XII on the pattern-lock keyboard. For a given segment, she/he was asked to draw it five times. To counteract the impact of possible fatigue, the participant was asked to take pauses between two consecutive inputs. Before the experiment, we informed all the participants that they could stop the ongoing experiment freely whenever they felt a need to rest. Finally, we purposely asked each participant to stop and rest for one or two minutes about every ten minutes.

For the set of experiments without task randomization, we designed multiple sessions to fully evaluate EyeTell. In the following sessions (from Section VI-C to Section VI-G), we will describe the details of these experiments (e.g., the number of participants, the experiment requirements, etc.) and

the corresponding results. The same participant took part in multiple sessions, resulting in a total time between two and three hours. To further reduce the impact of possible fatigue, we collected the data of the same participant on different days. As a result, the total time of doing experiments for each participant was less than one hour on the same day.

For the set of experiments with task randomization, we also designed different experiments for the same participant. Particularly, task randomization was done in two steps. First, we prepared all the experiments (tasks) for the same participant, assembled them together, and assigned each of them an order number. In our evaluation, a participant was assigned 24 single segments, 10 lock patterns, 10 4-digit PINs, and 10 6-digit PINs. Therefore, the order numbers are from 1 to 54 (i.e., $24+10+10+10=54$), which we use a vector $[1,2,\ldots,54]$ to denote. Second, we permuted the order vector randomly and obtained a new randomized one for each individual participant. Finally, each participant performed experiments according to her/his given vector.

As can be imagined, our experiments required a participant to look at the touchscreen of a mobile device and input on it repeatedly, which can result in fatigue. There are mainly two factors leading to fatigue in our experiments: experimental time and task similarity. Intuitively, if the experimental time is longer with very similar tasks, participants may easily suffer from fatigue. As mentioned above, we adopted two methods to reduce the impact of passible fatigue as much as possible. On the one hand, we asked the participants to take sufficient pauses during the experiments and stop the experiments freely, and controlled the duration of data collection on the same day. On the other hand, we conducted two sets of experiments, with and without task randomization. Since we did not observe large difference between the results of the two sets of experiments, we present the details and results of task randomization in Appendix C-B.

B. Performance Metrics

We use top-k inference accuracy as the main performance metric, as in [9], [19], [22], [23], [40]. Specifically, EyeTell generates a set of ranked candidate inputs (PINs, lock patterns, or letters) for each trial. We claim that a trial succeeds if the true input appears in the top-k candidate inputs. Top-k inference accuracy is defined as the percentage of successful trials. We compare the inference accuracy of EyeTell with that in [9], [19], [22], [23], [40]. Specifically, we compare EyeTell with [40] on inferring lock patterns and with [9], [19], [22], [23] on inferring English words.

C. Experiments on Pattern-Lock Keyboard

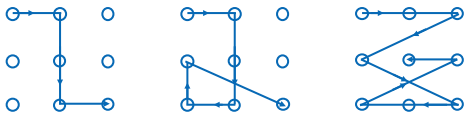
We first evaluate how accurately EyeTell can infer a single segment on the pattern-lock keyboard. Considering that inferring a single segment is the simplest task for EyeTell and the basis for more complicated ones, we want to see how well it performs. For this experiment, we asked each participant to draw each segment in Table XII on a Nexus 6 for five times. Recall that Table XII consists of all the possible single segments on a pattern-lock keyboard. For the segments with multiple possible starting points (e.g., segment ① can start from any point in $\{c_1, c_2, c_4, c_5, c_7, c_8\}$), the participants had the freedom to pick any starting point. Since there is only one segment in the resulting gaze trace, EyeTell can only calculate its angle but not its length. The output length is always 1 due to normalization. Therefore, we group the segments with the same angle together and obtain Table V from Table XII. Therefore, both segment ① and ② in Table XII correspond to segment ① in Table V. Here we ignore the impact of the segment length, which is reported in later evaluations. As we can see in Table VI, EyeTell can infer the angle of a single segment on the pattern-lock keyboard with top-1, top-2, and top-3 inference accuracy up to 87.76%, 98.65%, and 99.74%, respectively.

TABLE V. ANGLES OF A SINGLE SEGMENT ON THE PATTERN-LOCK KEYBOARD. DERIVED FROM TABLE XII.

Index	Angle	Index	Angle	Index	Angle	Index	Angle
①	0	⑤	$\frac{\pi}{2}$	⑨	π	⑬	$-\frac{\pi}{2}$
②	0.464	⑥	2.03	⑩	-2.68	⑭	-1.11
③	$\frac{\pi}{4}$	⑦	$\frac{3\pi}{4}$	⑪	$-\frac{3\pi}{4}$	⑮	$-\frac{\pi}{4}$
④	1.11	⑧	2.68	⑫	-2.03	⑯	-0.464

TABLE VI. INFERENCE ACCURACY ON A SINGLE SEGMENT OF PATTERN-LOCK KEYBOARD.

Index of segment	top-1	top-2	top-3	top-4	top-5
①	87.78%	100%	100%	100%	100%
②	82.5%	90.83%	100%	100%	100%
③	96.67%	100%	100%	100%	100%
④	95%	100%	100%	100%	100%
⑤	80%	100%	100%	100%	100%
⑥	92.22%	100%	100%	100%	100%
⑦	85%	96.67%	100%	100%	100%
⑧	93.33%	100%	100%	100%	100%
⑨	90%	100%	100%	100%	100%
⑩	93.33%	100%	100%	100%	100%
⑪	93.33%	100%	100%	100%	100%
⑫	60%	100%	100%	100%	100%
⑬	80%	92.5%	95.83%	100%	100%
⑭	88.33%	98.33%	100%	100%	100%
⑮	100%	100%	100%	100%	100%
⑯	87.67%	100%	100%	100%	100%
Average	87.76%	98.65%	99.74%	100%	100%



(a) Simple (b) Medium (c) Complex

Fig. 11. Examples of simple, medium, and complex lock patterns.

Then we evaluate the performance of EyeTell inferring lock patterns. We used the same set of lock patterns as those in [40], which includes 120 lock patterns in total [49]. In [40], the authors assigned a lock pattern to one of three categories, i.e., simple, medium, and complex, according to its complexity score. Specifically, the complexity score CS_P of an arbitrary lock pattern P is estimated as

$$\mathrm{CS}_P = n_P \times \log_2(L_P + I_P + O_P), \tag{4}$$

where n_P denotes the number of connecting dots, L_P is the length of P, I_P denotes the number of intersections, and O_P is the number of overlapping linear segments. Based on the complexity score, P can then be categorized according to the following rule. If $\text{CS}_P < 19$, P is simple; if $19 \leq \text{CS}_P < 33$, P is medium; and if $\text{CS}_P \geq 33$, P is complex. Fig. 11 gives an example for each pattern category. We use the simple pattern in Fig. 11(a) to explain the calculation of CS_P, for which we have $n_P = 5, L_P = 4, I_P = 0, O_P = 0$, and $\text{CS}_P = 10$. Each participant was assigned with four simple lock patterns, three medium ones, and three complex ones. The assignment of lock patterns was generated randomly. Besides, each lock pattern was drawn five times on a Nexus 6.

As shown in Table VII, the average top-1, top-5, top-10, and top-50 accuracy of EyeTell inferring pattern locks are 57.5%, 70.3%, 75.3%, and 85.1%, respectively. In [40], the authors reported average top-5 accuracy more than 95%, which is much higher than what EyeTell can achieve. But such high accuracy in [40] was achieved based on the strong assumption that the attacker can directly capture how the victim drew her/his lock pattern on the screen. In contrast, EyeTell assumes that the attacker can only capture the victim's eyes (possibly from a large distance), which is much more realistic. We can also see that the inference accuracy increases with the complexity score of a lock pattern, which is consistent with the observation in [40].The reason is that higher pattern complexity helps reduce the number of candidate patterns.

TABLE VII. INFERENCE ACCURACY ON PATTERN-LOCK KEYBOARD.

Pattern category	top-1	top-5	top-10	top-20	top-50
Simple	47.75%	69.5%	74.5%	79.5%	88.75%
Medium	59.3%	70%	75%	77%	83%
Complex	65%	71%	76%	78%	83%
Average	57.5%	70.3%	75.3%	78.3%	85.1%

D. Experiment on PIN Keyboard

We asked each participant to input 10 4-digit PINs and 10 6-digit PINs on the PIN keyboard on an iPhone 6s. Each PIN was input five times. All the PINs were randomly generated and then assigned to the participants. We showed the results in Table VIII. As we can see, EyeTell can infer 4-digit PINs with average top-1, top-5, top-10, and top-50 accuracy up to 39%, 65%, 74%, and 90%, respectively. In addition, the average top-1, top-5, top-10, and top-50 accuracy on 6-digit PINs are 39%, 70%, 80%, and 90%, respectively. As for pattern locks, the inference accuracy for 6-digit PINs is slighter higher than that for 4-digit PINs, as 6-digit PINs are longer, more complex, and thus easier to infer.

TABLE VIII. INFERENCE ACCURACY ON PIN KEYBOARD.

# of digits	top-1	top-5	top-10	top-20	top-50
4-digit	39%	65%	74%	81%	90%
6-digit	39%	70%	80%	85%	90%

E. Experiment on Word Inference

We used the 27 English words in Table XIII (Appendix C-A) from the corn-cob dictionary to evaluate the performance of EyeTell for word inference. The same words were also used in [9], [19], [22], [23]. The length of the 27 words ranges from 7 to 13 letters. We asked each participant to input each word five times on the alphabetical keyboard of an iPhone 6s.

Table IX compares the word-inference performance of EyeTell with some existing schemes. As we can see, the average top-5, top-10, and top-50 accuracy on inferring English words are 38.43%, 63.19%, and 72.45%, respectively. EyeTell has comparable performance to the attacks in [9], [19], [22], [23] but with weaker assumptions. For example, they assume that the attacker can obtain the exact length of the typed word, while EyeTell does not rely on this assumption. In addition, as detailed in Section II, they require that the attacker obtain on-board sensor data of the victim device [19], [22], [23] or that the victim device be placed on a static holder.

TABLE IX. WORD-INFERENCE ACCURACY.

System	top-5	top-10	top-25	top-50	top-100
EyeTell	38.43%	63.19%	71.3%	72.45%	73.38%
[19]	N/A	43%	61%	73%	87%
[22]	N/A	43%	50%	57%	60%
[23]	54.80%	63%	75%	82.40%	86%
[9]	48%	63%	78%	93%	N/A

F. Experiment on Sentence Inference

EyeTell infers a complete sentence in two steps. In the first step, we generate a candidate set for each typed word. In the second step, we use the linguistic relationships between English words to manually select the best candidate for each typed word. Essentially, inferring a complete sentence is based on inferring each individual word (in Section VI-E). Therefore, for this experiment, we only involved four participants to demonstrate the feasibility of our approach. Each participant was asked to input two sentences twice on the alphabetical keyboard of an iPhone 6s. The same sentences were also used for evaluation in [9]. Since the results for different participants are comparable, we only show the result of one trial for one participant for lack of space. We leave the results for other participants in Appendix C-C.

Table X shows the result. If a typed word does not appear in the candidate set generated by EyeTell, we use a $*$ to denote it. The words in italic form are those EyeTell infers successfully. We also show the number of candidates for each word (including itself). We can see that EyeTell can recover a large portion of the two sentences with the aid of post-inference human interpretation. We believe that we can further improve the performance on sentence inference by predicting unknown words using advanced linguistic models such as [50].

G. Influence Factors

In this section, we evaluate the impact of multiple factors on EyeTell for inferring 4-digit PINs on the PIN keyboard of an iPhone 6s, including the number of candidates (η) for segment decoding, the number of eyes used for extracting a gaze trace, the frame rate of the camcorder, the lighting condition for video recording, the distance between the victim and camcorder, and the recording angle. The following default setting was adopted, unless noted otherwise: $\eta = 5$, both eyes used for extracting a gaze trace, a frame rate of 60 fps, indoor normal lighting, 2 m between the victim and camcorder, and a zero-degree recording angle.

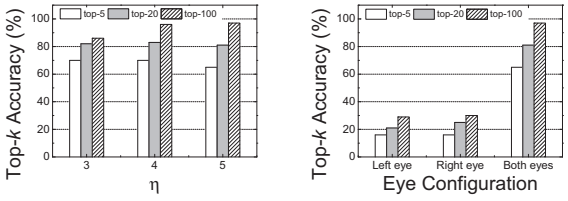
Among the 12 participants, only two of them do not wear glasses while the others do. Wearing glasses has little effect on the performance of our system. The reason is that we employ an image inpainting step to eliminate possible specularities

TABLE X. SENTENCE-INFERENCE RESULT.

Input	our	friends	at	the	university	of	texas	are	planning	a
Output	our	$*$	*at*	*the*	*university*	*of*	*texas*	*are*	*planning*	*a*
# of candi.	33	N/A	6	3	1	16	6	78	2	N/A
Input	conference	on	energy	economics	and	finance	in	february	of	next
Output	$*$	on	*energy*	$*$	and	finance	in	$*$	of	next
# of candi.	N/A	5	3	N/A	54	N/A	8	N/A	16	30
Input	year	we	discuss	the	major	factors	underlying	the	exceptionally	high
Output	year	*we*	*discuss*	*the*	major	$*$	*underlying*	*the*	$*$	high
# of candi.	15	7	8	5	44	N/A	1	5	N/A	85
Input	volatility	of	electricity	prices						
Output	$*$	*of*	*electricity*	*prices*						
# of candi.	N/A	16	2	26						

within the eye region for limbus detection, as mentioned in Section V-C2. As a result, we do not distinguish participants with glasses from those without glasses.

1) Impact of η: Fig. 12(a) shows the top-5, top-20, and top-100 inference accuracy of EyeTell for $\eta = 3, 4,$ or 5. As we can see, the inference accuracy increases with η, and the top-100 accuracy exhibits the largest increase. Such results are as expected because larger η leads to more enumerations in Section V-D3 so that the probability of the typed PIN falling into its candidate set increases. In our experiment, we found that when $\eta = 5$, most PINs and lock patterns were included in their respective candidate sets. Though a larger η always leads to higher accuracy, we set $\eta = 5$ by default to reduce computation time.



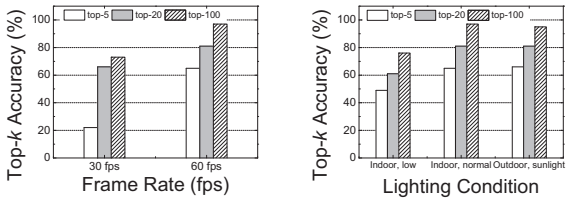
(a) η (b) Eye configuration

Fig. 12. Impact of η (left) and eye configuration (right).

2) Impact of eyes: Here we compare the inference accuracy when the gaze trace from only one eye (left or right) or from both eyes are used for PIN inference. The result is shown in Fig. 12(b). It is not surprising to see that EyeTell achieves much higher inference accuracy when the gaze traces of both eyes are used. The reason is that the gaze trace from one eye exhibits large noise due to the nature of human eyes while the gaze trace averaged from both eyes is much less noisy.

3) Impact of frame rate: Now we compare the inference accuracy of EyeTell under two frame rates for video recording, 30 fps and 60 fps. Since the default frame rate is 60 fps in our experiment, we down-sampled Ψ^l and Ψ^r in Section V-C3 by half to simulate the gaze trace obtained from 30-fps videos. As shown in Fig. 13(a), EyeTell can yield better inference results under a higher frame rate. The reason is that the gaze traces from videos of higher frame rates are more accurate than those of lower frame rates, thus resulting in higher accuracy.

4) Impact of lighting conditions: In this experiment, we evaluate the impact of environmental lighting conditions on EyeTell. Three types of environments are investigated, including indoor normal lighting with 300-360 lux illumination,



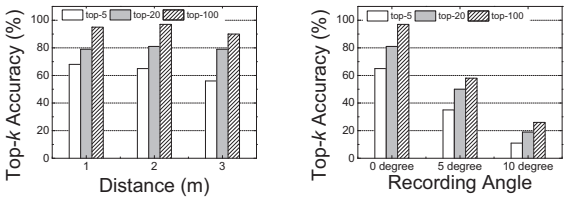
(a) Frame rate (b) Lighting condition

Fig. 13. Impact of frame rate (left) and lighting condition (right).

indoor low lighting with 60-100 lux illumination, and outdoor daytime sunlight with around 1200 lux illumination. In each environment, each participant was asked to input 10 4-digit PINs on an iPhone 6s, and each PIN was input five times. As mentioned above, the PINs were generated randomly and then assigned to the participants. Fig. 13(b) summarizes the result for this experiment. EyeTell exhibits similar performance under indoor normal lighting and outdoor daytime sunlight conditions. However, the performance becomes worse in indoor low lighting environments. The reason is that low illumination in the shooting environment causes more noise in detected eye regions, thus degrading the accuracy of ellipse fitting for limbus and later gaze trace extraction.

5) Impact of recording distance: In this experiment, we evaluate EyeTell when the recording distance is $1\mathrm{m}$, $2\mathrm{m}$, and $3\mathrm{m}$, respectively. In each scenario, each participant was asked to input 10 4-digit PINs on an iPhone 6s, and each PIN was input five times. The PINs were generated randomly and then assigned to the participants. We show the result in Fig. 14(a). As we can see, EyeTell has similar performance when the distance is $1\mathrm{m}$ or $2\mathrm{m}$. The slight performance degradation when the distance is $3\mathrm{m}$ can be attributed to the larger zoom-in setting from a longer shooting distance. As a result, the captured video may be more sensitive to small head movements of the victim. However, we believe that the impact of the recording distance can be very limited if the attacker has more advanced camcorders.

6) Impact of recording angle: In this experiment, we study the performance of EyeTell when the recording angle is 0°(the default), 5°, or 10°, respectively. In each scenario, each participant was asked to input 10 4-digit PINs on an iPhone 6s. Each PIN was input five times. The PINs were generated randomly and then assigned to the participants. Fig. 14(b) shows the results. As expected, the inference accuracy quickly decreases as the recording angle increases. This is mainly due to two reasons. First, the gaze tracking method [10] EyeTell



(a) Recording distance (b) Recording angle

Fig. 14. Impact of recording distance (left) and angle (right).

adopts assumes that the recording angle is zero. Second, when the recording angle increases, the recorded video may not be able to capture the limbus of both eyes. Accurate gaze trace extraction under arbitrary recording angles (or equivalently arbitrary head postures) is very challenging and requires more advanced gaze tracking methods. We plan to look further into this issue in our future work. Note that the attacker with an advanced camcorder may not have much difficulty achieving a near-zero recording angle in practice from a long distance to the victim.

H. Computational Time

We implemented EyeTell in two components. The first one is for gaze trace extraction implemented in C++, and the second for trace decoding implemented in Matlab. We run the experiments on a DELL desktop with 2.67 GHz CPU, 9 GB memory, and Windows 10 64-bit Professional. In the experiments, it takes less than 40s to generate a gaze trace from an input video. For trace decoding, the most time-consuming part is to generate the candidate set in Section V-D3, which is jointly determined by the number of segments and the number of candidates for each segment. Most PINs and lock patterns are associated with a few segments. For example, it takes less than 1s to generate the candidate set for a 4-digit PIN. In contrast, it takes about 40min for an English word with 13 letters. Overall, the computational time incurred by EyeTell is quite affordable for a determined adversary.

VII. Discussion

In this section, we discuss the limitations of EyeTell and point out possible countermeasures.

A. Limitations

First, the inference accuracy of EyeTell is slighter lower than that of other video-based inference attacks [9], [40], especially for the alphabetical keyboard. There are two main reasons. First, other attacks use more direct observations about the keystrokes, such as the device's backside motion [9] and the victim's finger movement [40]. In contrast, the gaze trace that EyeTell exploits only contains indirect keystroke information which is much more noisy and instable. Second, the efficacy of EyeTell on the alphabetical keyboard is largely limited by the uncertain number of keystrokes. We plan to explore extra side information such as eye fixation time in our future work to have more accurate estimation of the number of keystrokes and thus improve the inference accuracy of Eyetell.

Second, EyeTell currently requires the video to be recorded within a small recording angle, e.g., less than 5°based on our experiments. While such small recording angles make EyeTell detectable by vigilant users in uncrowded space, EyeTell is likely to succeed in crowded areas. This limitation can be alleviated by using more advanced camcorders or employing more advanced gaze tracking methods that are less sensitive to the victim's head posture. With better optics, the attacker can record the video from a longer distance. In addition, Gaze tracking based on machine learning [51] has shown to be effective even under different recording angles. We intend to explore this direction in our future work.

Finally, our experiment scale is comparable to that in the most recent work [40] but still limited. Though costly, larger-scale experiments may further evidence the efficacy of EyeTell.

B. Countermeasures

Since the only information EyeTell uses for keystroke inference is a video of the victim's eyes, mobile users should be alert when they input important sensitive information on their touchscreen devices. The following countermeasures can be adopted to thwart EyeTell. The most effective way against EyeTell is to prevent the attacker from video-recording the victim's eyes. For example, the user can wear sunglasses with dark colors to hide his gaze trace. In addition, users can input keystrokes without looking at the keys so that the gaze trace extracted by EyeTell is irrelevant to keystrokes. However, this method may be practical only when the user incurs a small number of keystrokes, e.g., 4-digit PINs. Finally, sophisticated users can increase their typing speed on the touchscreen. In case that the frame rate of the attacker's camcorder is not high enough, the extracted gaze trace should be much less accurate and noisy, therefore degrading the inference result.

VIII. Conclusion and Future Work

In this paper, we introduced EyeTell, a video-based keystroke inference attack framework to infer the victim's typed input from a video capturing his eyes. We adopted a user-independent model-based gaze tracking method to obtain a gaze trace of the victim's eyes and designed novel decoding algorithms to infer the typed input. We confirmed the high efficacy of EyeTell via extensive experiments on iOS and Android devices under various circumstances.

We plan to improve EyeTell in three directions in the future. First, we intend to develop novel gaze tracking methods that are less sensitive to the victim's head posture, which will greatly enhance EyeTell's applicability. Second, we will investigate novel methods to determine the number of keystrokes in order to improve the inference accuracy of EyeTell on alphabetical keyboards. Finally, we plan to evaluate EyeTell in a larger scale.

IX. Acknowledgement

We thank our shepherd and anonymous reviewers for their comments and help in preparing the final version of the paper. This work was supported in part by the US Army Research Office (W911NF-15-1-0328) and US National Science Foundation under grants CNS-1619251, CNS-1514381, CNS-1421999, CNS-1320906, CNS-1700032, CNS-1700039, CNS-1651954 (CAREER), and CNS-1718078.

REFERENCES

[1] M. Backes, M. Dürmuth, and D. Unruh, "Compromising reflections-or-how to read lcd monitors around the corner," in *IEEE S&P*, Oakland, CA, May 2008.

[2] M. Backes, T. Chen, M. Duermuth, H. Lensch, and M. Welk, "Tempest in a teapot: Compromising reflections revisited," in *IEEE S&P*, Oakland, CA, May 2009.

[3] D. Balzarotti, M. Cova, and G. Vigna, "ClearShot: Eavesdropping on keyboard input from video," in *IEEE S&P*, Oakland, CA, May 2008.

[4] F. Maggi, A. Volpatto, S. Gasparini, G. Boracchi, and S. Zanero, "A fast eavesdropping attack against touchscreens," in *Information Assurance and Security*, Melaka, Malaysia, December 2011.

[5] R. Raguram, A. White, D. Goswami, F. Monrose, and J.-M. Frahm, "iSpy: Automatic reconstruction of typed input from compromising reflections," in *ACM CCS*, Chicago, IL, October 2011.

[6] Y. Xu, J. Heinly, A. White, F. Monrose, and J. Frahm, "Seeing double: Reconstructing obscured typed input from repeated compromising reflections," in *ACM CCS*, Berlin, Germany, October 2013.

[7] Q. Yue, Z. Ling, X. Fu, B. Liu, K. Ren, and W. Zhao, "Blind recognition of touched keys on mobile devices," in *ACM CCS*, Scottsdale, AZ, November 2014.

[8] D. Shukla, R. Kumar, A. Serwadda, and V. Phoha, "Beware, your hands reveal your secrets!" in *ACM CCS*, Scottsdale, November 2014.

[9] J. Sun, X. Jin, Y. Chen, J. Zhang, R. Zhang, and Y. Zhang, "VISIBLE: Video-assisted keystroke inference from tablet backside motion," in *NDSS*, San Diego, CA, February 2016.

[10] E. Wood and A. Bulling, "Eyetab: Model-based gaze estimation on unmodified tablet computers," in *ACM ETRA*, Safety Harbor, FL, March 2014.

[11] L. Cai and H. Chen, "Touchlogger: Inferring keystrokes on touch screen from smartphone motion," in *USENIX HotSec*, San Francisco, CA, August 2011.

[12] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang, "Accessory: password inference using accelerometers on smartphones," in *ACM HotMobile*, San Diego, CA, February 2012.

[13] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. Choudhury, "Tapprints: your finger taps have fingerprints," in *ACM MobiSys*, Low Wood Bay, Lake District, UK, June 2012.

[14] Z. Xu, K. Bai, and S. Zhu, "Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors," in *ACM WiSec*, Tucson, AZ, April 2012.

[15] L. Simon and R. Anderson, "PIN skimmer: Inferring pins through the camera and microphone," in *ACM SPSM*, Berlin, Germany, November 2013.

[16] S. Narain, A. Sanatinia, and G. Noubir, "Single-stroke language-agnostic keylogging using stereo-microphones and domain specific machine learning," in *ACM WiSec*, Oxford, UK, July 2014.

[17] D. Asonov and R. Agrawal, "Keyboard acoustic emanations," in *IEEE S&P*, Oakland, CA, May 2004.

[18] L. Zhuang, F. Zhou, and J. Tygar, "Keyboard acoustic emanations revisited," in *ACM CCS*, Alexandria, VA, November 2005.

[19] Y. Berger, A. Wool, and A. Yeredor, "Dictionary attacks using keyboard acoustic emanations," in *ACM CCS*, Alexandria, VA, November 2006.

[20] T. Zhu, Q. Ma, S. Zhang, and Y. Liu, "Context-free attacks using keyboard acoustic emanations," in *ACM CCS*, Scottsdale, AZ, November 2014, pp. 453–464.

[21] J. Liu, Y. Wang, G. Kar, Y. Chen, J. Yang, and M. Gruteser, "Snooping keystrokes with mm-level audio ranging on a single phone," in *ACM MobiCom*, Paris, France, September 2015.

[22] P. Marquardt, A. Verma, H. Carter, and P. Traynor, "(sp)iphone: Decoding vibrations from nearby keyboards using mobile phone accelerometers," in *ACM CCS*, Chicago, IL, November 2011.

[23] X. Liu, Z. Zhou, W. Diao, Z. Li, and K. Zhang, "When good becomes evil: Keystroke inference with smartwatch," in *ACM CCS*, Denver, CO, October 2015.

[24] K. Ali, A. Liu, W. Wang, and M. Shahzad, "Keystroke recognition using wifi signals," in *ACM MobiCom*, Paris, France, September 2015.

[25] M. Li, Y. Meng, J. Liu, H. Zhu, X. Liang, Y. Liu, and N. Ruan, "When CSI meets public WiFi: Inferring your mobile phone password via WiFi signals," in *ACM CCS*, Vienna, Austria, October 2016.

[26] J. Zhang, X. Zheng, Z. Tang, T. Xing, X. Chen, D. Fang, R. Li, X. Gong, and F. Chen, "Privacy leakage in mobile sensing: Your unlock passwords can be leaked through wireless hotspot functionality," *Mobile Information Systems*, 2016.

[27] R. Bednarik, T. Kinnunen, A. Mihaila, and P. Fränti, "Eye-movements as a biometric," in *SCIA*, Copenhagen, Denmark, June 2005.

[28] O. Komogortsev, A. Karpov, and C. Holland, "CUE: counterfeit-resistant usable eye movement-based authentication via oculomotor plant characteristics and complex eye movement patterns," in *SPIE Defense, Security, and Sensing*, Baltimore, May 2012.

[29] C. Holland and O. Komogortsev, "Complex eye movement pattern biometrics: Analyzing fixations and saccades," in *IAPR ICB*, Madrid, Spain, June 2013.

[30] J. Sun, X. Chen, J. Zhang, Y. Zhang, and J. Zhang, "TouchIn: Sightless two-factor authentication on multi-touch mobile devices," in *IEEE CNS*, San Francisco, CA, October 2014.

[31] T. Li, Y. Chen, J. Sun, X. Jin, and Y. Zhang, "iLock: Immediate and automatic locking of mobile devices against data theft," in *ACM CCS*, Vienna, Austria, October 2016.

[32] A. D. Luca, R. Weiss, and H. Drewes, "Evaluation of eye-gaze interaction methods for security enhanced pin-entry," in *ACM OZCHI*, Adelaide, Australia, November 2007.

[33] A. D. Luca, M. Denzel, and H. Hussmann, "Look into my eyes!: Can you guess my password?" in *ACM SOUPS*, Mountain View, CA, July 2009.

[34] D. Liu, B. Dong, X. Gao, and H. Wang, "Exploiting eye tracking for smartphone authentication," in *ACNS*, New York, NY, June 2015.

[35] Z. Li, M. Li, P. Mohapatra, J. Han, and S. Chen, "iType: Using eye gaze to enhance typing privacy," in *IEEE INFOCOM*, Atlanta, GA, May 2017.

[36] A. Al-Haiqi, M. Ismail, and R. Nordin, "The eye as a new side channel threat on smartphones," in *IEEE SCORed*, Putrajaya, Malaysia, December 2013.

[37] "Structure of human eye," https://en.wikipedia.org/wiki/Human_eye.

[38] D. Hansen and Q. Ji, "In the eye of the beholder: A survey of models for eyes and gaze," *IEEE Trans. PAMI*, vol. 32, no. 3, pp. 478–500, March 2010.

[39] Z. Zhu and Q. Ji, "Novel eye gaze tracking techniques under natural head movement," *IEEE Trans. BME*, vol. 54, no. 12, pp. 2246–2260, December 2007.

[40] G. Ye, Z. Tang, D. Fang, X. Chen, K. Kim, B. Taylor, and Z. Wang, "Cracking Android pattern lock in five attempts," in *ISOC NDSS*, San Diego, CA, February 2017.

[41] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *IEEE CVPR*, Kauai, HI, December 2001.

[42] F. Timm and E. Barth, "Accurate eye centre localisation by means of gradients." in *VISAPP*, Algarve, Portugal, March 2011.

[43] J. Daugman, "High confidence visual recognition of persons by a test of statistical independence," *IEEE Trans. PAMI*, vol. 15, no. 11, pp. 1148–1161, November 1993.

[44] J. Wang, E. Sung, and R. Venkateswarlu, "Eye gaze estimation from a single image of one eye," in *IEEE ICCV*, Nice, France, October 2003.

[45] L. Świrski, A. Bulling, and N. Dodgson, "Robust real-time pupil tracking in highly off-axis images," in *ACM ETRA*, Santa Barbara, CA, March 2012.

[46] M. Kumar, J. Klingner, R. Puranik, T. Winograd, and A. Paepcke, "Improving the accuracy of gaze input for interaction," in *ACM ETRA*, Savannah, GA, March 2008.

[47] "corn-cob dictionary," http://www.mieliestronk.com/wordlist.html.

[48] "Trie data structure," https://en.wikipedia.org/wiki/Trie.

[49] "120 patterns for pattern lock keyboard," http://www.research.lancs.ac.uk/portal/files/138568011/Patterns.pdf.

[50] D. Ping, X. Sun, and B. Mao, "Textlogger: inferring longer inputs on touch screen using motion sensors," in *ACM WiSec*, New York, NY, June 2015.

[51] K. Krafka, A. Khosla, P. Kellnhofer, H. Kannan, S. Bhandarkar, W. Matusik, and A. Torralba, "Eye tracking for everyone," in *IEEE CVPR*, Las Vegas, NV, June 2016.

[52] E. Wood, "Gaze tracking for commodity portable devices," Ph.D. dissertation, University of Cambridge, 2013.

APPENDIX A
POINT-OF-GAZE ESTIMATION

In this step, we estimate one gaze point from each frame to obtain a complete gaze trace from the entire video. First, we calculate the 3D center and optical axis of each eye from the eye center and limbus obtained from limbus detection. Denote the coordinate of the eye center on the 2D image plane by (e_x, e_y) and the fitted ellipse of limbus by $E(x,y) = Ax^2 + Bxy + Cy^2 + Dx + Ey + F$. The 3D center of an eye, denoted by $\mathbf{c} = [c_x, c_y, c_z]^T$, can be calculated as

$$c_x = c_z \frac{(e_x - \mu_0)}{f_x}, c_y = c_z \frac{(e_y - \upsilon_0)}{f_y}, c_z = \frac{f_x + f_y}{2} \cdot \frac{r_0}{r_{\max}}, \tag{5}$$

where f_x and f_y are the focal lengths in pixel along horizontal and vertical axis, respectively, (μ_0, υ_0) is the coordinate of the principal point on the 2D image plane, $r_{\max}$ is the semi-major axis of the fitted ellipse $E(x,y)$ on the 2D image plane, and r_0 is the actual size of human limbus. By definition, the line determined by the focal point and the principal point is perpendicular to the 2D image plane, which allows us to calculate the principal point from the focal point. In practice, f_x, f_y, μ_0, and υ_0 can be obtained by one-time camera calibration. In addition, parameters e_x, e_y, and $r_{\max}$ can be computed from $E(x,y)$, and r_0 is set to $6\,\mathrm{mm}$ in our implementation.

The optical axis of an eye, denoted by $\mathbf{k}$, can be written as $\mathbf{k} = \mathbf{c} + m\mathbf{n}$. Here $\mathbf{n}$ is the unit normal vector of the supporting plane of the limbal circle, and m is a constant. In the coordinate system of the eye, $\mathbf{n}$ is equal to $[0,0,1]^T$. Next, we obtain its corresponding form in the coordinate system of the camera by the rotation matrix between the two coordinate systems through the following equation [52],

$$\mathbf{n} = [\mathbf{v_1} \quad \mathbf{v_2} \quad \mathbf{v_3}] \begin{bmatrix} h \\ 0 \\ g \end{bmatrix}, \tag{6}$$

where $\mathbf{v_1}$, $\mathbf{v_2}$, and $\mathbf{v_3}$ are three eigenvectors of $\mathbf{Q_e}$ defined as

$$\mathbf{Q_e} = \begin{bmatrix} A & \frac{B}{2} & -\frac{D}{f_x+f_y} \\ \frac{B}{2} & C & -\frac{E}{f_x+f_y} \\ -\frac{D}{f_x+f_y} & -\frac{E}{f_x+f_y} & \frac{4F}{(f_x+f_y)^2} \end{bmatrix}, \tag{7}$$

$$g = \sqrt{\frac{\lambda_2 - \lambda_3}{\lambda_1 - \lambda_3}}, h = \sqrt{\frac{\lambda_1 - \lambda_2}{\lambda_1 - \lambda_3}}, \tag{8}$$

and λ_1, λ_2, and λ_3 are the eigenvalues corresponding to $\mathbf{v_1}$, $\mathbf{v_2}$, and $\mathbf{v_3}$, respectively.

After obtaining the optical axis of each eye, we calculate the PoG as

$$\text{PoG} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix} + m \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}. \tag{9}$$

It follows that

$$m = -\frac{c_z}{n_z} \quad \text{and} \quad \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} c_x + mn_x \\ c_y + mn_y \end{bmatrix}, \tag{10}$$

where $[x,y]^T$ is the estimated PoG of a video frame.

APPENDIX B
KEYBOARD SPECIFICATION

Table. XI shows the soft keyboard dimensions illustrated in Fig. 7 and Fig. 8(a).

TABLE XI. SOFT KEYBOARD DIMENSIONS IN PIXEL ILLUSTRATED IN FIG. 7 AND FIG. 8(A).

Keyboard	Radius	Width	Height	Horizontal Gap	Vertical Gap
PIN	65	N/A	N/A	50	30
Pattern lock	20	N/A	N/A	340	340
Alphabetical	N/A	60	80	12	24
Quasi-PIN	N/A	216	80	12	24

Fig. 15 shows all the possible segments on PIN keyboard, similar to Fig.9.

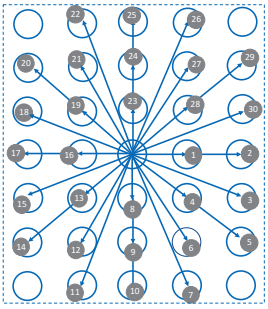


Fig. 15. All possible segments of a PIN keyboard.

Table. XII shows the lengths and angles of the segments in Fig. 9.

APPENDIX C
ADDITIONAL EVALUATION RESULTS

A. Word for Inference

Table XIII shows the 27 English words from the corn-cob dictionary to evaluate the performance of EyeTell for word inference.

B. Experiments with Task Randomization

In this session, we present more details and results on the set of experiments with task randomization. For these experiments, the number of participants is 10. As mentioned in Section VI-A2, each participant was assigned 54 ordered tasks, of which the order was indicated by her/his given vector. A task can be inputting a single segment, a lock pattern, a 4-digit PIN, or a 6-digit PIN. Each participant was asked to repeat the same task for five times. To reduce the impact of fatigue as much as possible, besides following the instruction in Section VI-A2, the participants were told to stop their

TABLE XII. ALL POSSIBLE SEGMENTS OF PATTERN-LOCK KEYBOARD.

Index	Length	Angle	Index	Length	Angle	Index	Length	Angle	Index	Length	Angle
①	1	0	⑦	1	$\frac{\pi}{2}$	⑬	1	π	⑲	1	$-\frac{\pi}{2}$
②	2	0	⑧	2	$\frac{\pi}{2}$	⑭	2	π	⑳	2	$-\frac{\pi}{2}$
③	$\sqrt{5}$	0.464	⑨	$\sqrt{5}$	2.03	⑮	$\sqrt{5}$	-2.68	㉑	$\sqrt{5}$	-1.11
④	$\sqrt{2}$	$\frac{\pi}{4}$	⑩	$\sqrt{2}$	$\frac{3\pi}{4}$	⑯	$\sqrt{2}$	$-\frac{3\pi}{4}$	㉒	$\sqrt{2}$	$-\frac{\pi}{4}$
⑤	$2\sqrt{2}$	$\frac{\pi}{4}$	⑪	$2\sqrt{2}$	$\frac{3\pi}{4}$	⑰	$2\sqrt{2}$	$-\frac{3\pi}{4}$	㉓	$2\sqrt{2}$	$-\frac{\pi}{4}$
⑥	$\sqrt{5}$	1.11	⑫	$\sqrt{5}$	2.68	⑱	$\sqrt{5}$	-2.03	㉔	$\sqrt{5}$	-0.464

TABLE XIII. WORDS FOR INFERENCE.

Length	Words
7	between, spanish, nuclear
8	identity, emirates, platinum, homeland, security
9	institute, extremely, sacrament, dangerous
10	difference, wristwatch, processing, unphysical
11	inquisition, pomegranate, feasibility, polytechnic, obfuscating
13	paediatrician, interceptions, abbreviations, impersonating, soulsearching, hydromagnetic

experiments at any time they wished. Also, we collected the data of the same participant on different days. For each participant, the experimental time on the same day was less than half an hour. The total experimental time for a participant ranged from one and a half to three hours.

1) Inferring a Single Lock-Pattern Segment: For these experiments, each participant input each segment in Table XII on a Nexus 6 for five times under task randomization. As we can see in Table XIV, EyeTell can infer the angle of a single finger movement on the pattern-lock keyboard under task randomization with top-1, top-2, and top-3 inference accuracy up to 87.19%, 97.10%, and 99.62%, respectively.

TABLE XIV. INFERENCE ACCURACY ON A SINGLE SEGMENT OF PATTERN-LOCK KEYBOARD.

Index of segment	top-1	top-2	top-3	top-4	top-5
①	82.5%	100%	100%	100%	100%
②	82.5%	92.5%	100%	100%	100%
③	92.5%	96.7%	100%	100%	100%
④	96.67%	100%	100%	100%	100%
⑤	75%	90.8%	100%	100%	100%
⑥	91.3%	100%	100%	100%	100%
⑦	83%	94.3%	100%	100%	100%
⑧	92.8%	98.2%	100%	100%	100%
⑨	90.3%	100%	100%	100%	100%
⑩	95.5%	100%	100%	100%	100%
⑪	93%	98%	100%	100%	100%
⑫	72%	100%	100%	100%	100%
⑬	84%	91%	94%	100%	100%
⑭	82%	100%	100%	100%	100%
⑮	97%	100%	100%	100%	100%
⑯	85%	92%	100%	100%	100%
Average	87.19%	97.10%	99.62%	100%	100%

2) Experiments on Inferring Lock Patterns: For these experiments, each participant input four simple patterns, three medium patterns, and three complex patterns from [49] on a Nexus 6 under task randomization. The patterns were randomly selected when preparing all the tasks for each participant. As shown in Table XV, the average top-1, top-5, top-10, and top-50 accuracy of EyeTell inferring pattern locks under task randomization are 55.8%, 70.1%, 75.1%, and 84.1%, respectively.

TABLE XV. INFERENCE ACCURACY ON PATTERN-LOCK KEYBOARD.

Pattern category	top-1	top-5	top-10	top-20	top-50
Simple	45.4%	70.4%	75.4%	77.2%	85.6%
Medium	58.6%	69.6%	74.0%	78.0%	83.2%
Complex	63.4%	70.2%	75.8%	77.6%	83.4%
Average	55.8%	70.1%	75.1%	77.6%	84.1%

3) Experiments on Inferring PINs on PIN Keyboard: For these experiments, a participant input 10 4-digit PINs and 10 6-digit PINs on an iPhone 6s under task randomization. The PINs were randomly generated when preparing all the tasks for each participant. As shown in Table XVI, EyeTell can infer 4-digit PINs with average top-1, top-5, top-10, and top-50 accuracy up to 37.5%, 67.2%, 78.0%, and 92.0%, respectively. In addition, the average top-1, top-5, top-10, and top-50 accuracy on 6-digit PINs are 38.8%, 68.9%, 81.3%, and 91.0%, respectively.

TABLE XVI. INFERENCE ACCURACY ON PIN KEYBOARD.

# of digits	top-1	top-5	top-10	top-20	top-50
4-digit	37.5%	67.2%	78.0%	81.2%	92.0%
6-digit	38.8%	68.9%	81.3%	84.6%	91.0%

C. Additional Results on Sentence Inference

Here we show more experimental results on sentence inference. As mentioned in Section VI-F, we involved four participants to take part in the experiments on sentence inference. Table XVII, Table XVIII, and Table XIX show the results for the other three participants. As we can see, the results for the four participants are comparable.

TABLE XVII. SENTENCE-INFERENCE RESULT FOR PARTICIPANT A.

Input	our	friends	at	the	university	of	texas	are	planning	a
Output	our	∗	*at*	*the*	*university*	*of*	*texas*	*are*	*planning*	*a*
# of candi.	27	N/A	8	3	1	14	6	90	2	N/A
Input	conference	on	energy	economics	and	finance	in	february	of	next
Output	∗	on	*energy*	∗	and	*finance*	in	*february*	of	next
# of candi.	N/A	5	3	N/A	30	2	8	5	12	18
Input	year	we	discuss	the	major	factors	underlying	the	exceptionally	high
Output	year	*we*	*discuss*	*the*	*major*	∗	*underlying*	*the*	∗	high
# of candi.	18	3	5	5	8	N/A	1	8	N/A	20
Input	volatility	of	electricity	prices						
Output	∗	*of*	*electricity*	*prices*						
# of candi.	N/A	23	1	14						

TABLE XVIII. SENTENCE-INFERENCE RESULT FOR PARTICIPANT B.

Input	our	friends	at	the	university	of	texas	are	planning	a
Output	our	∗	*at*	*the*	*university*	*of*	*texas*	*are*	*planning*	*a*
# of candi.	20	N/A	16	3	1	6	6	53	2	N/A
Input	conference	on	energy	economics	and	finance	in	february	of	next
Output	*conference*	on	*energy*	∗	and	finance	in	february	of	next
# of candi.	1	15	6	N/A	54	N/A	8	10	7	25
Input	year	we	discuss	the	major	factors	underlying	the	exceptionally	high
Output	year	*we*	*discuss*	*the*	major	∗	*underlying*	*the*	∗	high
# of candi.	21	3	5	3	60	N/A	1	5	N/A	100
Input	volatility	of	electricity	prices						
Output	∗	*of*	*electricity*	*prices*						
# of candi.	N/A	18	1	10						

TABLE XIX. SENTENCE-INFERENCE RESULT FOR PARTICIPANT C.

Input	our	friends	at	the	university	of	texas	are	planning	a
Output	our	∗	*at*	*the*	*university*	*of*	*texas*	*are*	*planning*	*a*
# of candi.	40	N/A	8	2	2	11	6	63	2	N/A
Input	conference	on	energy	economics	and	finance	in	february	of	next
Output	∗	on	*energy*	∗	and	finance	in	*february*	of	next
# of candi.	N/A	7	3	N/A	42	N/A	8	2	14	18
Input	year	we	discuss	the	major	factors	underlying	the	exceptionally	high
Output	year	*we*	*discuss*	*the*	major	∗	*underlying*	*the*	∗	high
# of candi.	12	5	12	8	32	N/A	1	5	N/A	91
Input	volatility	of	electricity	prices						
Output	∗	*of*	*electricity*	*prices*						
# of candi.	N/A	12	1	16						