A Fast Eavesdropping Attack Against Touchscreens

Federico Maggi
Alberto Volpatto
Politecnico di Milano
{volpatto, fmaggi}@elet.polimi.it

Simone Gasparini
INRIA Grenoble–Rhone-Alpes
simone.gasparini@inria.fr

Giacomo Boracchi
Stefano Zanero
Politecnico di Milano
{boracchi, zanero}@elet.polimi.it

***Abstract*—The pervasiveness of mobile devices increases the risk of exposing sensitive information on the go. In this paper, we arise this concern by presenting an automatic attack against modern touchscreen keyboards. We demonstrate the attack against the Apple iPhone—2010's most popular touchscreen device—although it can be adapted to other devices (e.g., Android) that employ similar key-magnifying keyboards. Our attack processes the stream of frames from a video camera (e.g., surveillance or portable camera) and recognizes keystrokes online, in a fraction of the time needed to perform the same task by direct observation or offline analysis of a recorded video, which can be unfeasible for large amount of data. Our attack detects, tracks, and rectifies the target touchscreen, thus following the device or camera's movements and eliminating possible perspective distortions and rotations In real-world settings, our attack can automatically recognize up to 97.07 percent of the keystrokes (91.03 on average), with 1.15 percent of errors (3.16 on average) at a speed ranging from 37 to 51 keystrokes per minute.**

I. Introduction

Since more and more people use mobile devices, there is an increasing risk of inadvertently exposing sensitive information. For instance, attackers may break into mobile devices by exploiting vulnerabilities exposed via wireless links. Even when digital attacks are ineffective, malefactors may closely follow and observe the victim's portable device or use recording mechanisms and extract the target information. However, this strategy is highly inefficient from an attacker's viewpoint, especially when the target information is time sensitive (e.g., onetime passwords) or ample (e.g., long emails).

We designed and implemented a practical, automatic attack against touchscreen mobile devices. Through a quasi real-time processing, our attack accurately recovers the sequence of keystrokes input by the user. Balzarotti *et al*. proposed in [1] an attack that, albeit limited to desktop scenarios, is similar in spirit to ours; however, their approach relies on conditions that well suit desktop scenarios (i.e., camera and mechanical keyboard in relative fixed, perpendicular position) but prevent its extension to mobile settings, which are characterized by keyboard movements and skewed or rotated viewpoints. To the best of our knowledge, our work is the very first attempt to implement an automatic shoulder-surfing attack against touchscreen mobile devices.

We designed our attack specifically for mobile scenarios and dynamic conditions. In particular, it requires no specific positioning of the spying camera nor of the target device as it compensates the natural movements of both target device and attacker's camera. This makes it suitable for "on the go" applications. Our attack provides accurate text recognition, without relying on any a priori knowledge or manual intervention of the attacker. In particular, it does not implement any grammar check, thus we can automatically recognize context-free text or keystroke sequences not corresponding to dictionary words (e.g., passwords). Our attack shows that feedback mechanisms of modern touchscreens excessively expose sensitive information in a way that an attacker may recover it automatically and very efficiently. Unfortunately, as discussed in §VI and §V, no definitive protection solutions offering the same degree of usability of key magnification exist.

In summary, we make the following original contributions:

- To the best of our knowledge, this is the first study of practical risks brought forth by mainstream touchscreen keyboards, with particular attention to privacy leaks.
- We designed a precise, fast, and practical attack that detects keystrokes while a user is typing on a touchscreen. The technique is robust to rotations and perspective distortions, thus the attacker is not required to stand behind the victim nor to observe the face of the screen perpendicularly.
- We made our attack robust to occlusions (e.g., fingers, onscreen reflexes), by developing an efficient filtering technique that validates detected keys and reconstructs keystroke sequences accurately.

II. Threat Model & Requirements

In our threat model the attacker just points a video camera to the target touchscreen while the victim enters a text. No remote nor local access to the device is required. Our attack automatically reconstructs the text based *solely* on the keystroke feedback displayed on the screen: No visibility of typed text is required. For example, the victim may type text on a password input field, where the letters are hidden by asterisks or dots. In our envisioned scenario, exemplified in Fig. 1a, an attacker would stand behind the victim (e.g., waiting at a bus station) and point a smartphone camera towards the target device. The attack works nearly in real-time and requires only an adequate buffer of frames.

The generality of our approach depends *exclusively* on the following simple and realistic requirements:

Requirement 1 The target virtual keyboard must display a feedback whenever a key is pressed. From now on, we refer to such a feedback as the *magnified key*. Magnified keys must be partially visible (at least in one frame after

978-1-4577-2155-7/11/$26.00 ©2011 IEEE

Phase 1	*Screen Detection and Rectification*
Input	Current frame.
Task	Detect the touchscreen image by feature extraction and matching against the template of the target screen. When a match is found, rectify and crop the screen area. Any successful match improves matches in the next frame(s).
Output	A rectified, cropped and scaled image of the device screen in the current frame. This is close to the image that a fixed camera had acquired when the device is at a fixed distance, with its screen parallel to the camera's sensor.
Phase 2	*Magnified Keys Detection*
Input	Rectified image of the target screen.
Task	Isolate magnified-key candidates, i.e., high-contrast areas of the rectified image that are different from the template and previous frames.
Output	A segmented image (i.e., a map of the image areas) identifying the magnified-key candidates (blobs). Typically, there is more than one blob per frame.
Phase 3	*Keystroke Sequence Recognition*
Input	A set of magnified-key candidates.
Task	Filter out wrong candidates, by matching them with the corresponding template of the magnified key, thus identifying the best-matching key.
Output	The symbol, if any, associated to the best-matching key.

Table I: Overview of our attack.

each keystroke). Our attack works even when fingers partially cover the magnified keys, as it typically happens.

Requirement 2 The attacker must know the model of the target device and obtain the following static information:

- *Screen template*: A screenshot or a photograph of the target device and application used by the victim.
- *Key template*: The appearance (i.e., sizes and font family or symbol set) of each magnified key.
- *Magnified layout*: Set of coordinates of the magnified key centers. In what follows it is represented by $ML = \{c_1, \cdots, c_L\}$; note that these points can be easily mapped onto a regular grid. For example, in the US English keyboard (Fig. 1e), the magnified layout contains the coordinates of $L = 26$ magnified keys.

iPhone, Android, and recent BlackBerry devices, the most popular mobile touchscreen phones, all meet Requirement 1 Requirement 2 is easily met by taking a screenshot of the target application(s) (e.g., Mail, Twitter) on (a copy of) the target device.

III. Attack Description

Fig. 1 and Tab. I summarize the proposed attack. The input of Phase 1 is a frame of the video sequence (b), while the output is the corresponding rectified, rotated, and cropped image of the device screen (c). In Phase 2 this image is compared with the screen template and the previous (rectified) frames, to detect differences. Typically, these differences (d) do not allow to identify yet the magnified key, as there could be other magnified-key candidates. Phase 3 determines the best match among all these candidates, by direct comparison of the selected areas against their corresponding templates. Thus, the typed symbol is successfully recognized.

Notation

We consider frames as grayscale images. An image $\mathtt{I}$ is a matrix of real values in $[0,1]$, and $\mathtt{I}(x,y)$ indicates the intensity of its pixel at coordinate (x,y). $\mathtt{I}_t$ and $\mathtt{Z}_t$ are the acquired frame and the corresponding rectified screen at time t, respectively. For easier notation, we indicate the 2D pixel coordinates with vector $\mathbf{x} = (x,y)$ and, where not specified, we assume that we assume that $\mathbf{x}$ belongs to the domain of $\mathtt{Z}_t$.

A. Phase 1: Detection and Rectification

We first leverage *screen detection* to search for any occurrence of the screen in the input video, and then exploit *image rectification* to rectify the image of the detected screen by estimating a suitable perspective transformation. Both methods rely on feature extraction and matching. An image feature is a small image patch centered on a peculiar point of the image, usually where the image presents a discontinuity (e.g., a corner or an edge). Given two images and their features, these are matched to find image correspondences (i.e., two features representing the same object in the scene). We use SURF features [2] because they are invariant to rotation, scaling and skew transformations, and they have a lower computational cost which makes them more suitable for real-time applications. For the sake of clarity, we first explain the rectification task.

Image Rectification

When the spying camera aims at the device from a skewed position the resulting image of the screen is perspectively distorted. Our approach corrects this by constructing a (synthetic) rectified image that preserves the screen's geometry. In general, the distorted image of a planar surface is related to its rectified version by a 3×3 matrix $\mathtt{H}$ called *homography* [3]. The homography maps corresponding points of the two images, that is $\left[x,y,1\right]^T \sim \mathtt{H}\left[x',y',1\right]^T$, where (x',y') and (x,y) are the image coordinates of the points of the acquired images before and after rectification, respectively. $\mathtt{H}$ is a full-rank matrix (hence the relation is invertible) defined up to scale (i.e., it has 9 elements but only 8 of them are independent [4]) and can be estimated from the distorted and rectified images using a minimum of 4 corresponding points on the two images.

In our case there are many invariant parts on the screen, e.g., the keyboard and other graphical elements. Therefore, we use the screen template as a reference rectified image, and look for matches to estimate, at any time t, the matrix $\mathtt{H}_t$. Therefore, for each input frame $\mathtt{I}_t$, we obtain the rectified image $\mathtt{Z}_t$ by applying the estimated $\mathtt{H}_t$ to each pixel belonging to the device screen: $\mathtt{Z}_t(x,y) = \mathtt{I}_t(x',y')$, where (x,y) and (x',y') are related by the homography. The rectified image $\mathtt{Z}_t$ contains only the device screen and has the same size of the screen template—thanks to image interpolation. Finally, we scale $\mathtt{Z}_t$ to guarantee that the darkest area correspond to 0 and the lightest to 1, such that $\mathtt{Z}_t$ can be easily compared with the screen template.

Screen Detection

The degree of distortion and the position of the screen in the frame can vary as the camera moves; therefore, we must search the whole frame to isolate the screen image. In doing this, we account that the screen can be (dynamically) occluded by fingers or other objects a priori unknown.

(a) Envisioned attack settings. (b) Input of Phase 1. (c) Input of Phase 2. (d) Input of Phase 3. (e) Output (i.e., the 'R' key).

Figure 1: Intermediate outputs captured in a sample attack (a). Phase 1: The device screen is detected in each frame $\mathtt{I}_t$ (b), cropped and rectified, yielding $\mathtt{Z}_t$ (c). Phase 2: The magnified-key candidates are selected within the foreground, i.e., the image areas shown in (d). Phase 3: According to the coordinates of the magnified layout *ML* (e), each candidate is compared to its template to identify the typed key. The template of 'R' is selected as it shows the best match.

To ensure efficient and reliable screen detection we pursue a feature-based template-matching approach. Features of the screen template (e.g., screen corners) are matched with the features extracted from $\mathtt{I}_t$ to find corresponding points and detect the region of $\mathtt{I}_t$ where the screen appears. For more accurate detections, we rule out false correspondences by enforcing the following procedure: all the correspondences are used to estimate the homography $\mathtt{H}_t$ in a RANSAC [5] process, which allows to discriminate inliers and outliers, i.e., good and false corresponding points. If the number of inliers is sufficiently larger than the number of outliers, we considered the screen as detected and rectify the image of the screen through the estimated homography. Otherwise, no screen is detected and we discard the frame. This approach is faster than a pixel-wise comparison of the two images and it can be easily extended to any other device just by using the proper template image.

B. Phase 2: Magnified Keys Detection

Magnified keys are nonstationary elements of the rectified frame sequence and we can detect them by leveraging a *background subtraction* technique [6]. For each screen image $\mathtt{Z}_t$, we estimate the *background* $\mathtt{B}_t$, which describes the stationary elements, and the *foreground* $\mathtt{F}_t$, which discloses those parts of $\mathtt{Z}_t$ that have changed. The foreground is defined as follows: $\mathtt{F}_t = 1$ where $|\mathtt{Z}_t(\mathbf{x}) - \mathtt{B}_{t-1}(\mathbf{x})| > k\,\Sigma_{t-1}(\mathbf{x})$ and zero otherwise. Here $k > 0$ is a tuning parameter, $\mathtt{B}_{t-1}$ and Σ_{t-1} are the estimates, at $t-1$, of the background image and its standard deviation, respectively. These are defined as

$$\mathtt{B}_t(\mathbf{x}) = \begin{cases} \mathtt{B}_{t-1}(\mathbf{x}), & \text{if } \mathtt{F}_t(\mathbf{x}) \neq 0 \\ \alpha\,\mathtt{F}_{t-1}(\mathbf{x}) + (1-\alpha)\,\mathtt{B}_{t-1}(\mathbf{x}), & \text{otherwise} \end{cases}, \tag{1}$$

and

$$\Sigma_t(\mathbf{x}) = \begin{cases} \Sigma_{t-1}(\mathbf{x}), & \text{if } \mathtt{F}_t(\mathbf{x}) \neq 0 \\ \sqrt{\alpha(\mathtt{F}_t(\mathbf{x}) - \mathtt{B}_t(\mathbf{x}))^2 + (1-\alpha)\,\Sigma^2_{t-1}(\mathbf{x})}, & \text{otherwise} \end{cases}, \tag{2}$$

where $\alpha \in [0,1]$ is an update parameter.

We initialize $\mathtt{B}_0(\mathbf{x})$ using the screen template, and $\Sigma_0(\mathbf{x}) \sim \sigma\ \forall \mathbf{x}$, where σ is the standard deviation of the image noise, computed as in [7]. This initialization method allows us to detect keystrokes since the very first frame of the video, without need of any training sequence or attacker's manual intervention. Although these are quite naïve estimates, the updating process (1) and (2) guarantees satisfactory recognition.

1) Magnified Keys Identification

The foreground highlights occlusions (most probably typing fingers, illumination variations, rectification errors) and magnified keys. We disambiguate magnified keys from other occlusions by exploiting the following priors:

- Key magnification lasts for few frames, often less than other occlusions. Thus, we define a short-term foreground $\mathtt{S}_t$, which highlight image parts that have recently changed: $\mathtt{S}_t(\mathbf{x}) = 1$ where $|\mathtt{F}_t(\mathbf{x}) - \overline{[\mathtt{F}_t(\mathbf{x})]}_n| > 0$, and zero otherwise, with $\overline{[\mathtt{F}_t(\mathbf{x})]}_n = \frac{1}{n}\sum_{i=1}^{n} \mathtt{F}_{t-i}(\mathbf{x})$, where $n \in \mathbb{N}$ is the minimum number of frames a magnification lasts.
- Magnified keys—black characters over a white key area—are characterized by higher contrast than other occlusions and the background. These provide a high response when $\mathtt{Z}_t$ is processed by high-pass filters. Therefore, we compute the gradient $\mathtt{G}_t$ and the Laplacian $\mathtt{L}_t$ magnitudes by means of convolutional filters:

$$\mathtt{G}_t(\mathbf{x}) = [(\mathtt{Z}_t \circledast g_x)(\mathbf{x})]^2 + [(\mathtt{Z}_t \circledast g_y)(\mathbf{x})]^2, \text{ and} \tag{3}$$

$$\mathtt{L}_t(\mathbf{x}) = \left[\left(\mathtt{Z}_t \circledast g_x^2\right)(\mathbf{x})\right]^2 + \left[\left(\mathtt{Z}_t \circledast g_y^2\right)(\mathbf{x})\right]^2, \tag{4}$$

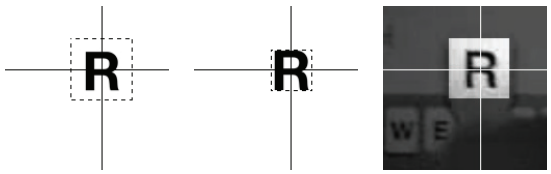
where g_x, g_y and g_x^2, g_y^2 denote the first- and second-order derivative Sobel filters [8], respectively, whereas $\circledast$ indicates discrete 2-D convolution.

The heuristic pixel-wise measure indicating of how likely the foreground contains a magnified key is defined as

$$\mathtt{M}_t(\mathbf{x}) = \begin{cases} \frac{1}{3}\left(\frac{\mathtt{G}_t(\mathbf{x})}{\max \mathtt{G}_t(\mathbf{x})} + \mathtt{S}_t(\mathbf{x}) + \frac{\mathtt{L}_t(\mathbf{x})}{\max \mathtt{L}_t(\mathbf{x})}\right), & \text{if } \mathtt{F}_t(\mathbf{x}) \neq 0 \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Note that $\mathtt{M}_t(\mathbf{x})$ is in $[0,1]$, and that, $\mathtt{G}_t(\mathbf{x})$ and $\mathtt{L}_t(\mathbf{x})$ in (3) and (4) are taken into account only in the foreground pixels (i.e., where $\mathtt{F}_t(\mathbf{x}) \neq 0$).

We experienced that by thresholding $\mathtt{M}_t(\mathbf{x})$ we can reliably isolate the magnified keys. The threshold $\Gamma > 0$ is determined as in [9] and provides $\mathtt{K}_t(\mathbf{x})$, which is defined as $\mathtt{K}_t(\mathbf{x}) = 1$ where $\mathtt{M}_t(\mathbf{x}) > \Gamma$ and zero otherwise. This binary image $\mathtt{K}_t$ is then segmented, using morphological image processing techniques [8], to identify its connected components (blobs). Each blob is a set of pixels coordinates and we select the keystrokes among these blobs. Fig. 1d shows the image values in these blob's areas.



(a) Full $T^f(c_4)$ (b) Cropped $T^r(c_4)$ (c) $ROI(c_4)$

Figure 2: Examples of full key template, (a), used for computing $\mathrm{d_{bw}}$, and the cropped key template, (b), used for computing ncc, defined in (7), both performed with respect to the *ROI*, (c), for a given key that, in this example, corresponds to 'R'.

C. Phase 3: Keystroke Sequence Recognition

Each blob $b \in \mathcal{B}_t$ yields one or more magnified-key candidates, belonging to the magnified layout *ML*: Phase 3 selects $c_t^\star$, the *best-matching key* at time t among these candidates as the one that maximizes the *key similarity*.

The set of magnified-key candidates $\mathcal{C}_t$ is determined as follows. For each blob $b_i \in \mathcal{B}_t$, we compute its barycenter $\dot{b}_i$ and its closest key $c_i = \mathrm{argmin}_{c \in ML}\, \mathrm{d}(c, \dot{b}_i)$, where $\mathrm{d}(c, \dot{b}) = \|c - \dot{b}\|$ is the Euclidean distance. Then, for each c_i, we define its *neighborhood* $\mathcal{N}(c_i)$ as the set of the coordinates of c_i and its adjacent keys (see Fig. 4). Specifically, in our implementation $\mathcal{N}(c_i) = \{c_i^l, c_i, c_i^r\}$ is used, where labels 'l' and 'r' indicate the magnified keys at the left and the right of c_i, respectively. The set of *candidate keys* is $\mathcal{C}_t = \bigcup_{i=1}^{|\mathcal{B}_t|} \mathcal{N}(c_i)$.

1) Key Similarity

For each key $c \in ML$, the attacker devises the *full key template*, $T^f(c)$, the *cropped key template*, $T^r(c)$ (as stated in Requirement 2). Then, from the rectified frame $\mathtt{Z}_t$ we crop the *Region Of Interest* of c, $ROI(c)$, which contains the magnified key whenever c is pressed. More formally, $ROI(c) \subset \mathtt{Z}_t, \forall t$, is an image area centered in c, having the same size of its full template $T^f(c)$. When c has not been pressed, $ROI(c)$ contains the keyboard background or occluding objects. Templates and *ROI* are illustrated in Fig. 2. The key similarity $\Phi_t(c)$ involves a pixel-wise comparison of the *ROI* against the key templates, and as such it could be computationally demanding. For this reason, we evaluate it only when $ROI(c)$ is likely to contain a specific magnified key by discarding most of candidate keys: We compare, for each $c \in \mathcal{C}_t$, the *percentage of black and white pixels* in $ROI(c)$ and in the corresponding full template, $T^f(c)$. When these percentages are similar, we put c in a set of *selected candidates*, $\mathcal{C}_t^\star \subseteq \mathcal{C}_t$, otherwise we discard c. More precisely, the *black-white distance* between $ROI(c)$ and the corresponding template is

$$\mathrm{d_{bw}}(c) = \mathrm{d}\left(\mathrm{bw}(ROI(c)) - \mathrm{bw}(T^f(c))\right)/\sqrt{2}, \qquad (6)$$

where $\mathrm{bw}(\mathtt{A}) = (A_b, A_w)$ is a function providing the percentages of black (A_b) and white (A_w) pixels for any image region $\mathtt{A}$, and the factor $\sqrt{2}$ guarantees that $\mathrm{d_{bw}} \in [0,1], \forall c$. This distance is fast to compute and we leverage it to build $\mathcal{C}_t^\star = \{c \in \mathcal{C}_t \mid \mathrm{d_{bw}}(c) \leq \Gamma_{\mathrm{bw}}\}$, which contains only those keys whose full template matches the corresponding *ROI*. The threshold Γ_{bw} is determined as described in §III-D; we consider "black" pixels with intensity lower than 0.3, and white those pixels with intensity above 0.5. We determine these parameters experimentally, by exploiting the fact that areas not belonging to magnified keys typically have values in $[0.3, 0.5]$.

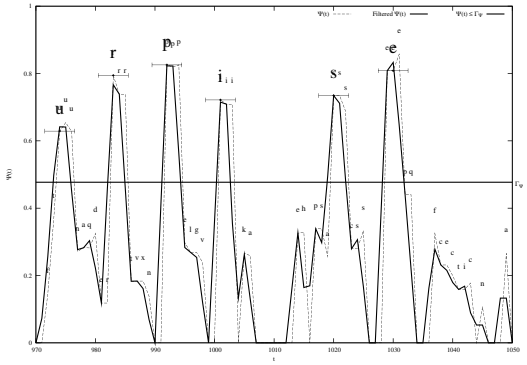


Figure 3: Low-passed key similarity measure $\Psi(t)$ of the best matching key $c_t^\star$, threshold Γ_Ψ, and local maxima. Frames providing values of $\Psi(t)$ below Γ_Ψ are discarded. The brackets above each local maximum indicate the minimum distance between two local maxima, which can be considered as the minimum number of frames the key magnification lasts. The selected keys are also displayed. Note that selected magnified keys last longer than one frame, and that the zero values in $\Psi(t)$ correspond to frames where phase 2 does not detect any blob.

The *key similarity* of the magnified-key candidate $c \in \mathcal{C}_t^\star$ is proportional to the maximum value of the *normalized cross-correlation*, ncc(), between the *cropped* key template $T^r(c)$ and the $ROI(c)$:

$$\Phi_t(c) := \frac{\max\left(\mathrm{ncc}(T^r(c), ROI(c))\right)}{1 + d_c}, \qquad (7)$$

where d_c is the distance between the candidate c and $\dot{b}_i$, the barycenter of the corresponding blob b_i that yields $c \in \mathcal{N}(c_i)$. Maximizing $\mathrm{ncc}(T^r(c), ROI(c))$ means considering different displacements of $T^r(c)$ to determine the best match with $ROI(c)$. Fortunately, a very fast algorithm for computing ncc() exists [10].

2) Best Matching Key

We identify the *best-matching key* at time t as $c_t^\star := \mathrm{argmax}_{c \in \mathcal{C}_t^\star} \Phi_t(c)$, and the corresponding *key-similarity measure* is $\Psi(t) := \Phi_t(c_t^\star)$. Summarizing, $c_t^\star$ identifies the magnified key selected as the most likely to appear in frame $\mathtt{Z}_t$, whereas $\Psi(t)$ represents the measure of the similarity between the template, $T^r(c_t^\star)$, and the corresponding $ROI(c_t^\star)$.

3) Keystroke Sequence Analysis

Key magnifications typically last longer than one frame, and there are frames that contains no magnified keys, thus it is insufficient to identify the best-matching key at each time step. We address these problems by analyzing $\Psi(t)$ when t varies (an example of Ψ is plotted in Figure 3). We exploit the fact that key magnification has fading transitions, thus the measure $\Psi(t)$ reaches its maximum and then decreases every time a key magnifies. Therefore, the first case is successfully handled by extracting the best-matching keys corresponding to local maxima of $\Psi(t)$.

As frames without magnified keys typically exhibit low values of $\Psi(t)$, we can easily discard them by thresholding. We determine such threshold Γ_Ψ experimentally as the value

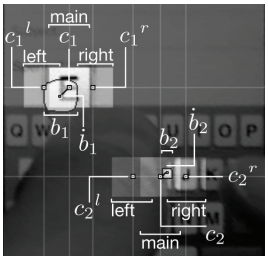


Figure 4: At frame z_t, for each blob b_1, b_2 the candidate keys c_1, c_2 are found as the keys that are closer to each blob's centroid $\dot{b}_1, \dot{b}_2$. The corresponding neighborhood are $\{c_1^l, c_1, c_1^r\}$, and $\{c_2^l, c_2, c_2^r\}$, respectively.

of $\Psi(t)$ where there is no key magnification, as described in §III-D. Furthermore, to reduce fluctuations of the key similarity measure, we preliminarily low pass $\Psi(t)$ (e.g., by an averaging filter). We set a minimum distance of 5 frames between adjacent local maxima (corresponding to a very high typing speed at 25fps). Fig. 3 illustrates the local maxima extraction procedure, and the corresponding keystrokes recognized. We stress that, by extracting the local maxima of the key similarity, we naturally recognize typed doubles.

D. Parameters Estimation

We determine both thresholds Γ_{bw} and Γ_{Ψ} through the following statistical approach. We acquire few videos of different users mimicking text typing without actually pressing any key, and process each video with our system to record the values of $\mathrm{d}_{\mathrm{bw}}(\cdot)$ and $\Psi(\cdot)$ into two sequences. We consider these sequences as realizations of two random variables (i.e., the values of d_{bw} and $\Psi(\cdot)$ when there are no keys magnified on the screen). Hence, with Chebyshev's inequality, we determine two confidence intervals for their expectations as $\Gamma_{\mathrm{bw}} = \hat{\mu}_{\mathrm{d}} + \nu\hat{\sigma}_{\mathrm{d}}$ and $\Gamma_{\Psi} = \hat{\mu}_{\Psi} + \eta\hat{\sigma}_{\Psi}$, where $\nu, \eta \in \mathbb{R}$ are tuning parameters, whereas $\hat{\mu}$ and $\hat{\sigma}$ are the sample mean and standard deviation computed over the respective sequences. Our experiments on the iPhone revealed that $\nu = 0$ and $\eta = 3$ yield satisfactory results.

IV. Experimental Evaluation

We show that our attack is faster than a human inspecting the same video while yielding comparable accuracy, and we evaluate the robustness of our attack in extreme working conditions. We recorded videos merely for repeatability of experiments, whereas our attack works perfectly in streaming. We published a sample video of an experimental session[1].

We used three types of text inputs:

- *Context-free text:* Available at http://sqze.it/qMNwy; it has poor context (63 English words, 444 symbols plus spaces). Since we are comparing our attack against human attackers, context-free text ensures that people involved in our experiments cannot to simply guess words using the linguistic context.
- *Context-sensitive text:* Available at http://sqze.it/SGTu-; the first 65 words of the lyrics of Dream Theater's *"Regression"* song, which is rich of context (total 278 symbols plus spaces).
- *Brief text:* Used to evaluate specific features and limitations:. *"Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas"* (80 letters plus spaces and symbols, total 13 words).

[1] http://www.youtube.com/playlist?list=PL81F91E404B928833

	Hits	Errors	Speed
	Context-free text (444 letters)		
Our attack	**91.03**	3.16	**0.674**
Attacker 1	96.09	0.75	0.327
Attacker 2	87.78	1.90	0.550
Attacker 3	94.61	1.39	0.306
	Context-sensitive text (278 letters)		
Our attack	**89.11**	7.64	**0.803**
Attacker 1	97.11	1.07	0.290
Attacker 2	93.20	1.90	0.586
Attacker 3	97.77	1.18	0.280

Table II: Average hit and error rates, and speed (keystrokes per second) of our attack (faster) versus manual recognition (slower).

A. Precision and Speed

We performed 3 sessions on context-free text and 3 on context-sensitive text, each with different victims typing naturally. Without any a priori knowledge, each attacker had to recognize the keystrokes by stopping, rewinding or slowing-down the recorded stream as needed. As summarized in Tab. II, and thoroughly discussed in our experiments detailed in [11], manual inspection is notably slower than our attack. Our system can recognize up to 0.803 keystrokes per second, about one third of the maximum typing speed, and 0.864 in the best case, about half of the average typing speed. Only twice (over 18 trials) the attackers were able to beat such speeds. As expected, human sight can recognize symbols with slightly higher precision than our system, especially with context-sensitive text. However, our system is just 3 percentage points less accurate than the attacker with the highest average precision; plus, our attack is remarkably faster. The state-of-the-art system [1], even under more relaxed hypotheses, works at 0.101 keystrokes per second on average, with a maximum precision of 82%, only achievable in context-sensitive text. Our attack needs no linguistic context to recognize 97.77% of the keystrokes at 0.803 keystrokes per second.

In case of faster typing, manual analysis is more error prone, whereas our system is not influenced by typing speed (as far as there are at least 5 frames between two consecutive keystrokes, as reported in §III-C3)

B. Resilience to Disturbances

We measured the robustness of Phase 1 through a series of brief typing sessions with significant disturbances: (1) we attached a piece of gray tape diagonally on the screen to emulate a permanent occlusion, (2) we asked the victim to shake the device while typing, (3) we shook the camera during recording, and (4) both the camera and the target device were shaking during recording.

Tab. III shows that Phase 1 can rectify parts of the video and recognize the screen. Under conditions (1) to (3), Phase 2–3 recognized up to 96% (44.44% with permanent occlusions) of the symbols with 4% errors (33.33% with permanent occlusions). However, users seldom hold touchscreen devices with permanent occlusions, especially while typing on the go.

(a) Occlusion (b) Camera + device movements. (c) Device movements only.

Figure 5: Permanent occlusions (a) and quick camera movements (c are tolerated with errors. However, when both camera and device shake excessively b) the high blur level prevents the detection of the screen.

As shown in Fig. 5, our system can handle sudden movements of either the camera or device, whereas Phase 1 fails when both the camera and device move excessively, resulting in intra-frame, motion-blurring effects that prevent successful feature extraction.

V. Related Work

Balzarotti *et al*. proposed in [1] an attack related to ours, as discussed in §I, although unsuitable to mobile scenarios and significantly inefficient as mentioned in § IV-A.

Vendors proposed to reduce the viewing angle of touch-screens to limit an attacker's chances of seeing what a victim is typing [12], as in so called "privacy screen filters". Besides the fact that these mechanisms are not widespread on the mainstream market, they could make it harder to launch our attack, but simply because it would be more challenging for the attacker to find a suitable angle to observe the target device from, not specifically because of a limitation of our approach.

In [13], the authors propose to map sensitive, short information onto a different, temporary, mnemonic alphabet (e.g., colors or simple shapes). This mapping is dynamically chosen by users before typing. This approach mitigates casual shoulder surfers, but the authors explicitly mention that no protection is guaranteed against attackers armed with video cameras. Therefore, our attack cannot be effectively mitigated by this type of techniques. A more extensive review of these methods is on [11] and hereby omitted for space limitations.

A different approach was proposed in [14] for public touchscreens, consisting in tracking the user's pupil movements and map them onto a grid layout to implement a gaze-based keyboard. Unfortunately, these countermeasures are not suitable for mobile devices conditions, because the accuracy of the eye-tracking decreases if the use moves or the device shakes. In addition, as the defensive measures discussed above, this mitigation is limited to protecting passwords and short texts.

VI. Conclusions

We proposed an automatic attack that recognizes keystrokes from key-magnification, which is enabled by default in iPhone, newer BlackBerry, and Android devices. In realistic scenarios our attack is as accurate as manual inspection, yet significantly faster. Disabling key magnification, disallowed by Apple mobile devices, can mitigate our attack. Therefore, keyboards that employ key-magnification feedback are unsuitable for high-privacy applications and, given that the most popular touchscreen devices display such feedback.

Mechanical keyboards provide about the same degree of usability of magnifying touchscreen keyboards, although the former are less privacy leaking. In fact, we recorded a typing sessions on a mobile mechanical keyboard[2], using a long English text with no linguistic context. Six volunteers who analyzed the video reported that most of the keystrokes were not actually visible. Only one volunteer was able to recognize up to 1.35% of the text. Other volunteers gave up for excessive fatigue.

Future work will concentrate on detecting nonmagnifying keys (e.g., spacebar), not recognized on our current implementation. Analogous observations apply to alternative layouts (e.g., landscape). The latter will require minor modifications. Instead of using only one screen template (e.g., portrait-alphabetical), Phase 1 will need to cycle through several alternative layouts (e.g., portrait alphabetical, portrait numerical, landscape numerical) and choose the best-matching one before. Similarly, Phase 2 would need additional key templates.

Disturbance	Phase 1	Phase 2–3	
		Hits %	Errors %
(1) Permanent occlusion	difficult	44.44	33.33
(2) Jiggled device	feasible	67.74	8.70
(3) Jiggled camera	feasible	96.00	4.00
(4) Jiggled device + camera	unfeasible	0.00	-

Table III: Detection results under different working conditions.

References

[1] D. Balzarotti, M. Cova, and G. Vigna, "ClearShot: Eavesdropping on Keyboard Input from Video," in *SSP '08*, Oakland, CA, May 2008.

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

[3] J. G. Semple and G. T. Kneebone, *Algebraic Projective Geometry*. Oxford Classic Texts, 1998.

[4] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2004.

[5] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communication of ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[6] M. Piccardi, "Background subtraction techniques: a review," in *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, vol. 4, October 2004, pp. 3099 – 3104 vol.4.

[7] D. L. Donoho and I. M. Johnstone, "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol. 81, no. 3, pp. 425–455, 1994.

[8] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (3rd Ed.)*. Prentice-Hall, Inc., 2006.

[9] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 9, no. 1, pp. 62 –66, jan. 1979.

[10] J. Lewis, "Fast normalized cross-correlation," in *Vision Interface*, vol. 10, 1995, pp. 120–123.

[11] F. Maggi, A. Volpatto, S. Gasparini, G. Boracchi, and S. Zanero, "Don't touch a word! a practical input eavesdropping attack against mobile touchscreen devices," Politecnico di Milano, Tech. Rep. TR-2010-59, 2010.

[12] Y.-M. Tsuei, "Method and apparatus for preventing on-screen keys from being accidentally touched using the same," US Patent 12427767, HTC Corporation, April 2009. [Online]. Available: http://www.google.com/patents?id=VU_TAAAAEBAJ

[13] D. S. Tan, P. Keyani, and M. Czerwinski, "Spy-resistant keyboard: more secure password entry on public touch screen displays," in *OZCHI '05*. Narrabundah, Australia: CHISIG of Australia, 2005, pp. 1–10.

[14] M. Kumar, T. Garfinkel, D. Boneh, and T. Winograd, "Reducing shoulder-surfing by using gaze-based password entry," in *SOUPS '07*. New York, NY, USA: ACM, 2007, pp. 13–19.

[2] http://www.youtube.com/watch?v=JxrqYA56A48