iType: Using Eye Gaze to Enhance Typing Privacy

Zhenjiang Li[1], Mo Li[2], Prasant Mohapatra[3], Jinsong Han[4], Shuaiyu Chen[4]
[1]City University of Hong Kong; [2]Nanyang Technological University;
[3]University of California, Davis; [4]Xi'an Jiaotong University

***Abstract*—This paper presents *iType*, a system that uses eye gaze for typing private information on commodity mobile platforms. The design combats three primary challenges: 1) relatively low accuracy of mobile gaze tracking; 2) difficulties in correcting input errors due to lacking the comparison with the true text-entry value; and 3) device motions and other noises that may interfere gaze tracking accuracy and thus the iType performance. We devise a set of effective techniques, including leveraging a collective behavior of the gaze tracking results, unique correlation of the typing error spatial distributions, and motion sensor hints from mobile devices, to address above challenges. A set of enhancement techniques are applied to further improve iType's robustness and reliability. We consolidate above designs and implement iType on iOS platform. Evaluations show that iType achieves high keystroke detection accuracy for the secure typing within a reasonable short latency.**

I. Introduction

Motivation. Imagine the following scene. Bob notices a new voice mail when he is on the bus and needs to input the password to access his voice mailbox. He sequentially looks at the buttons on the screen. The front camera of the phone reads his gaze and infers each character he intends to type for completing the input. Later when Bob enters a coffee shop, he leverages his gaze again to input the password of his bank card for the payment, prohibiting it to being exposed to nearby customers. As a matter of fact, such an eye gaze based typing can be potentially adopted to input various private information in practice, *e.g.,* for account login, phone unlocking, dialing private numbers, etc.

While this may be an intriguing vision about the future, it has unveiled an emerging and urgent privacy issue. Nowadays mobile devices offer us the most convenient user experience ever [10], *e.g.,* at anytime and anywhere, but we unavoidably face a new potential threat at the same time: the interaction between users and mobile devices may be exposed to public directly, which may leak very sensitive information of the user, *e.g.,* passwords, private data, account information, etc. If the input of such information is not properly protected, the user's privacy can be easily emanated and compromised in public.

How do existing solutions prevent privacy leakage on mobile devices? Recent approaches mainly leverage human's biometrics or behaviors, *e.g.,* voices, faces [9], fingerprints [24], iris and its move [7, 26], phone usage features [11], etc., to avoid explicit input of the private information. However, most of these approaches usually require specialized sensors, which will increase the device cost and make solutions accessible to a limited set of (and also expensive) devices. More importantly, existing solutions are not generic — they are concentrated at certain services, like unlocking or authenticating a device, but

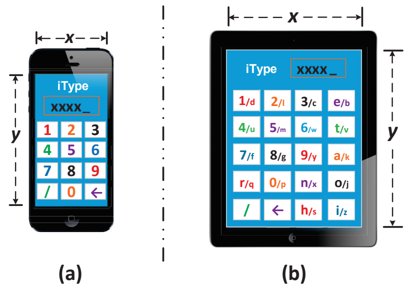

Fig. 1: Illustration of iType design. Screen is divided into (**a**) a 4×3 grid on phones holding 10 numbers and (**b**) a 5×4 grid on tablets holding both 10 numbers and 26 English letters on two pages. Button "←" means *delete*. When one button hosts multiple characters, we could use button "/" for *paging*.

they are not usable to many conventional services that require an explicit private text-entry, *e.g.,* providing textual password for an on-line login, inputting the personal data in transactions, unlocking devices without biometric sensors, etc.

In this paper, we propose iType for the private information input using the eye gaze, as illustrated in Figure 1. In iType, the keyboard consists of multiple buttons and each button represents unique character(s) (number or letter). For the ease of presentation, we refer *password* to various kinds of private information for short. To type a password, the user looks at the corresponding buttons sequentially, and iType essentially solves a decoding puzzle: it reads the user's gaze, infers the buttons being looked at, and assembles the password. The iType typing is secure primarily due to the fact that the eye gaze is difficult to eavesdrop. Even an adversary in front of the user could decode the eye gaze, the gaze itself conveys no meaningful information, unless it matches with the keyboard layout, which however can be user-defined and changed.

Challenges. Translating the iType idea to a practical system entails crucial challenges, covering *accuracy*, *latency* and *mobility* several aspects:

1) *Low accuracy of mobile gaze tracking*. iType is designed atop the gaze tracking technique [12, 14], which relies on the gaze tracker trained in advance. As the relative position between front camera and user's eyes may vary over time during the real usage, it could make existing gaze tracking solutions inherently inaccurate and unreliable when they are used on mobile devices. Precisely inferring typed characters based on low-accuracy gaze tracking results is difficult.

2) *Lack of true text-entry value in error correction*. The assembled password may contain typing errors, which will be

rejected by the underlying application, and the user has to type it again to correct the errors. Due to the privacy concern, the password plain text is not displayed on the screen, and the correctness of each recognized character is thus unknown to the user. If iType verifies to the application only when an intact password is obtained, it may significantly impair the typing efficiency and prolong the typing delay.

3) *Noises from device motions*. As front camera continuously tracks the user's face, device motions during typing may blur the captured frames and degrade the quality of gaze tracking. The gaze tracking accuracy further deteriorates, which will in turn degrade the iType performance.

Contributions. In this paper, we tackle above challenges based on the following observations and techniques.

1) We find that although the individual gaze tracking results (points) are unreliable, their statistics can give good approximations. We thus look at a group of gaze tracking points and leverage their collective behavior to approximate the user's true gaze position. Based on that we propose a technique to confirm the typing of each character using a minimal number of gaze tracking points. It ensures both the accuracy and delay for typing individual characters.

2) When assembled password contains typing errors, iType requires another round of user's input. We observe that rearranging the layout of buttons shuffles the vicinity of true gaze positions, which thus provides opportunities to migrate typing errors and jointly decode user's input cross multiple rounds even the typing channel has a low signal-to-noise ratio.

3) The quality of the frames generated from the front camera is impacted by device motions. iType leverages accelerometer readings to select frames with the best expected quality to enhance the typing performance.

We implement iType on the iOS platform and comprehensively evaluate its performance in various environments. We show iType can achieve high accuracy for private information typing, *e.g.,* 97% and 89% for individual keystroke detections in static and dynamic scenarios, respectively. Although private information normally has limited length, *e.g.,* 4 to 8 characters, iType still controls the input within reasonable short latency, *e.g.,* $2.0s$ per character, and 84% (and all) correct passwords are obtained after the first (and second) typing round, indicating that good typing accuracy and latency can be achieved simultaneously. Field trial studies further verify the efficacy of iType cross different users.

Roadmap. §II is the overview. iType is designed in §III, developed in §IV and evaluated in §V. We conduct literature review in §VI before concluding in §VII.

II. Overview

Utility of iType. iType is specifically designed for applications where private information needs to be explicitly typed on mobile devices in public, such as passwords for logging into on-line accounts, personal data needed in authentications or transactions (*e.g.,* birthday, security codes of credit cards, etc.), unlocking devices without bio-information sensors, etc. iType design principle can be applied beyond commodity personal devices as well, *e.g.,* to support interactions with variant third-party point-of-sale (POS) terminals, like depositions at bank ATMs, and payments using membership cards or credit cards.

		Screen size (mm)	Exp. error ($^\circ$)
Phones	x	$51 \sim 69$	$1.7 \sim 2.3$
	y	$90 \sim 122$	$1.8 \sim 2.4$
Tablets	x	$120 \sim 148$	$2.5 \sim 3.0$
	y	$161 \sim 200$	$2.2 \sim 2.9$

TABLE I: Screen size survey for mobile devices.

Although the gaze based typing is inherently less efficient than traditional keyboard based methods due to the responsiveness of human's eyes, iType is positioned as a secure channel (or alternative) to type private information on mobile devices, in stead of replacing these methods. Even private information is usually short, *e.g.,* less than 8 characters, iType still strives to achieve high typing accuracy within reasonable short latency, *e.g.,* $2.0s$ per character, for this purpose.

Gaze tracking accuracy needed in iType. We survey on the mainstream smartphones and tablets in Table I, where x and y two directions are illustrated in Figure 1. Given these prevalent screen sizes, the iType keyboard layout design considers both the character recognition accuracy and the typing efficiency. In particular, the screen is divided into a 4×3 grid to accommodate 10 numbers, $0 \sim 9$, for smart phones, and a 5×4 grid to contain 10 numbers and 26 English letters for tablets as shown in Figure 1. If the number of buttons is smaller than the total number of characters, we can organize characters to multiple pages and introduce a "/" button for paging, *e.g.,* button "1/d" in Figure 1(b) means "1" and "d" are placed on two pages, and the user sequentially types buttons "/" and "1/d" to choose letter "d".

According to above keyboard layout design, we derive the gaze tracking accuracy required by iType, which is summarized in Table I. At a common device-to-eye distance, *e.g.,* $25cm$ to $35cm$, the error[1] is expected to be about 2.0° to 3.0° to reliably confirm a typed character. Table I indicates iType requires to precisely track the user's eye gaze in the design.

Gaze tracking accuracy on mobile devices. The input of gaze tracking is a stream of frames from front camera that captures user's eyes. The core module, *gaze tracker*, then calculates (x, y) coordinates of the user's gaze on the screen plane for each frame. The implementation of one state-of-the-art gaze tracker is detailed in §IV. We now evaluate the mobile gaze tracking performance under various settings including both static and dynamic environments. The expected error range in Table I, *e.g.,* 2.0° to 3.0°, is marked in Figure 2(a) and (b), which indicate that the achieved performance cannot obtain the desired accuracy to directly support the iType design. We thus need novel solutions to effectively cope with this unreliable gaze tracking issue in the next section.

[1]Errors are normally measured by the angle between two line segments: from the eye to the target point and the derived gaze point on the screen [19].

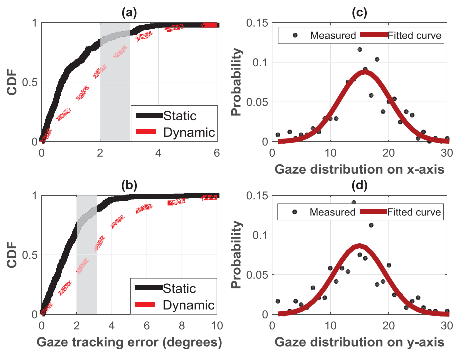

Fig. 2: Gaze tracking errors. Accuracy achieved on (**a**) phones and (**b**) tablets. Distributions along (**c**) x-axis and (**d**) y-axis. The shadow area indicates the exp. errors in Table I.

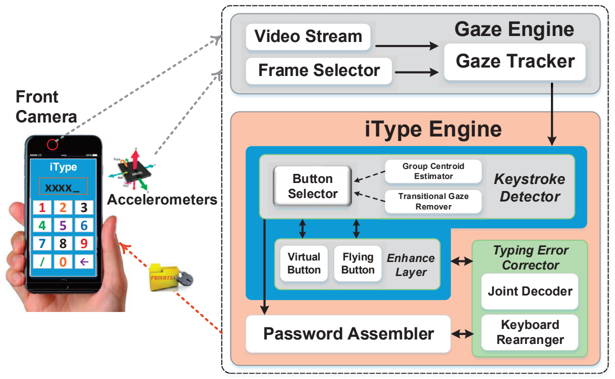

Fig. 3: iType architecture. iType converts eye frames and motion sensor signals to the securely typed information.

III. iType Design

Figure 3 depicts the architecture and components of iType.

Gaze engine. Gaze engine converts the live video frames to a stream of gaze point coordinates. After iType starts, video frames are extracted from front camera and they are attached with accelerometer readings. Due to the processing delay, only a subset of frames can be utilized to produce gaze tracking points (§IV), and *frame selector* leverages accelerometer readings to select best quality frames against external dynamics (§III-C). The derived gaze coordinates (x_i, y_i), from the i-th processed frame, are streamed to the iType engine.

iType engine. iType engine converts gaze tracking points to the typed information. The keystroke detector embraces *group centroid estimator* and *transitional gaze remover* two techniques to recognize each typed character (§III-A), which are concatenated as a password in the password assembler. If the password contains typing errors, the typing error corrector leverages *joint decoder* and *keyboard rearranger* two techniques to migrate errors and jointly decode user's input cross multiple typing rounds (§III-B). We further propose *virtual button* and *flying button* two techniques in the enhancement layer to improve the robustness of iType (§III-D).

A. Keystroke detector

1) Group centroid estimator: **Observation.** To address the low accuracy issue of gaze tracking on mobile platforms, we observe when a user stares at one position on the screen, although individual gaze tracking results could contain highly disturbed errors, they all scatter around user's true gaze position. This phenomenon inspires us to look at a group of gaze tracking results, instead of individual ones, and leverage their collective behavior to approximate the user's gaze position.

We analyze the distribution of gaze points when the user looks at different positions on the screen. Figure 2(c) and (d) plot the histograms of gaze points from one group along x and y axises. We divide each axis into multiple bins and count the percentage of gaze points falling into each bin. The result suggests that the gaze points along each direction nearly follow a Gaussian distribution, *i.e.,* $\mathcal{N}(\mu, \sigma)$, where μ and σ are the expectation and standard deviation, respectively. Gaze points from other groups exhibit similar distributions but with different μ and σ values. For each group, the expectations (μ)s of the two Gaussian distributions along x and y directions together indicate the user's intended gaze position. This motivates us to leverage the *centroid* of each group to approximate this position, and the question turns to how many gaze points are enough to derive the centroid, since each point is merely a sample from two underlying unknown Gaussian distributions.

Solution. A natural solution is to collect enough gaze points and apply curve fitting to determine the expectations when the fitting error becomes sufficiently small. It could however incur a long computation delay, and may not be suitable for real-time mobile applications. To address this issue, we employ *T-distribution* [13] to define a confidence range for quantifying the possibility that the difference between the group centroid and the expectations of underlying Gaussian distributions falls into this range. Their closeness can be determined as follows:

Suppose there are n gaze points g_i, where $i = 1, 2, \ldots, n$. A confidence interval $(\tilde{x} - \frac{\tilde{\sigma}}{\sqrt{n}} t_{\frac{\alpha}{2}}, \tilde{x} + \frac{\tilde{\sigma}}{\sqrt{n}} t_{\frac{\alpha}{2}})$ can be constructed along the x axis. For this interval, parameters $\tilde{x}$ and $\tilde{\sigma}$ are the average and the standard deviation of g_i respectively, α represents the error rate, and t_j is can be determined from the *t-distribution* look-up table. The confidence interval along the y axis can be similarly defined. The expression of the interval indicates that with more gaze points, the confidence interval becomes smaller (*i.e.,* the confidence level is increased). It can be proven that the confidence level for the true expectation of the Gaussian distribution within this constructed confidence interval equals to $1 - \alpha$.

Algorithm 1 details the execution of above process. As gaze points are streamed into the gaze engine, C_x and C_y are used to temporarily store the x and y coordinates of n gaze points received so far, respectively, and we calculate their standard deviations $\tilde{\sigma}_x$ and $\tilde{\sigma}_y$. Given α (*e.g.,* 10%) and the confidence range thresholds l_x and l_y, which are set to be half of the distances between two neighboring buttons

Algorithm 1: Group centroid estimator

```
1 input: $C_x = \{x_i\}$, $C_y = \{y_i\}$, where $i = 1, 2, \ldots, n$, and
  $(x_i, y_i)$ is the coordinates of the $i$-th gaze point;
2 output: the group centroid $(x_c, y_c)$;
3 calculate $\tilde{\sigma}_x$ and $\tilde{\sigma}_y$ for $C_x$ and $C_y$, respectively;
4 if $\frac{\tilde{\sigma}_x}{\sqrt{\tilde{n}}} t_{\alpha/2} \leq l_x$ && $\frac{\tilde{\sigma}_y}{\sqrt{\tilde{n}}} t_{\alpha/2} \leq l_y$ then
5     return $(x_c = \frac{1}{n}\sum_{i=1}^{n} x_i,\ y_c = \frac{1}{n}\sum_{i=1}^{n} y_i)$;
```

along x and y directions respectively, the group centroid (x_c, y_c) can precisely approximate the expectations of two underlying Gaussian distributions when the condition in line 4 of Algorithm 1 is satisfied. The keystroke is confirmed as the button that is closet to group centroid (x_c, y_c). The device then displays an "x" sign on the screen to inform the completion of the current keystroke to the user.

2) Transitional gaze remover: After finishing one keystroke, the user's gaze may delay on the previous button[2] and then shifts to the next one. As front camera generates continuous video frames, the transitional gaze points will also be streamed to the iType engine, which however should be excluded; Otherwise the Gaussian property for next keystroke will be violated that may cause incorrect detections.

Observation. Our key observation here is that gaze points can demonstrate distinctive spatial distributions in different phases, *e.g.,* a tail-like distribution in the transition phase v.s. the Gaussian distribution in the typing phase. Although we have profiled the Gaussian distribution in the previous subsection, it is difficult to precisely describe transitional gaze points, since their exact distribution is determined by from which two buttons the transition phase starts and ends, and how a user shifts eyes in between. We thus need to separate them even the exact distribution in the transition phase is unknown.

We divide the screen plane into $k * k$ grids (this division can be denser than the keyboard layout in Figure 1) and the time into a sequence of consecutive time windows with size w, where w gaze points are generated in each time window. For example in Figure 4(a), k and w equal to 4 and 12, respectively. We observe that in the transition phase, the gaze point distributions in different time windows change gradually. However, when the user's gaze falls on the intended button, the gaze points exhibit Gaussian distributions. This motivates us to leverage the changes of the gaze point distribution, instead of its exact expression, to distinguish the transition phase. Our design principle is thus: if the gaze point distributions cross different time windows keep changing, the user's gaze is considered in the transition phase. If such distributions become stable, we conclude the gaze focuses on an intended button.

Solution. For any two discrete distributions $Q = \{q_i\}$ and $P = \{p_i\}$, Kullback-Leibler divergence (KLD) [17] could be

[2]With the virtual button design in §III-D, such residual gaze points will not mis-trigger keystroke on the previous button for the second time.

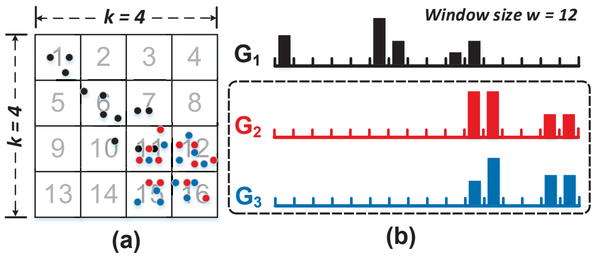

Fig. 4: Transitional gaze remover design. (**a**) 4×4 screen grids. (**b**) Example in 3 windows, where window size $w = 12$.

applied to quantify their similarity as follows:

$$D_{KL}(Q \| P) = \sum_i q_i \log_2 \frac{q_i}{p_i}, \tag{1}$$

where $D_{KL}(Q\|P)$ represents the information lost when P is used to approximate Q. A smaller D_{KL} indicates that two distributions are more similar to each other.

According to our design principle, for time window W_j, we define its discrete distribution G_j as the normalized histogram of w gaze points falling into $k * k$ grids, *e.g.,* G_1 to G_3 in Figure 4(b). Therefore, we obtain a series of $G = \{G_j\}$ as time elapses. Since some bins could be empty as shown in Figure 4(b), we assign a small value, *e.g.,* $1/w$, to all the bins in each G_j initially to avoid $p_i = 0$ in Eqn. (1), and the normalized histogram is added atop this baseline. Transitional gaze remover then works as follows:

1) After one keystroke detection is confirmed (by Algorithm 1 in §III-A1), transitional gaze remover starts to build G for the next keystroke. We denote $P = G_j$ and $Q = G_{j+1}$, and apply Eqn. (1) to calculate their D_{KL} value.

2) If D_{KL} is greater than a threshold τ, the gaze points in P are considered in the transition phase, which will be excluded from the centroid estimation for the next keystroke. In the example of Figure 4(b), we configure $\tau = 0.2$. As $D_{KL}(G_2\|G_1) = 1.806$, gaze points in G_1 will be excluded.

3) If D_{KL} is less than τ, it means that P and Q share a similar distribution, *e.g.,* $D_{KL}(G_3\|G_2) = 0.107$, and we emerge them, *e.g.,* histogram over these two time windows, to improve the reliability for the next comparison.

4) We conclude that the user's gaze is in the typing phase when G_j with similar distributions are consecutively observed. Then the gaze points in these time windows having similar distributions, together with subsequent points (if any), will be used in the group centroid estimation for the next keystroke.

In transitional gaze remover, two parameters, *i.e.,* the window size w and threshold τ, are configured empirically through the experiments.

3) Button selector: In summary, to type a character in iType, transitional gaze points (in the transition after the previous keystroke) are first excluded. Later when transitional gaze remover confirms a new typing phase, group centroid estimator accepts consequent gaze points until the group centroid (x_c, y_c) is derived. Button selector selects the button that is closest

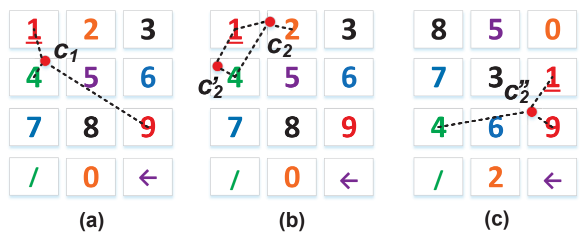

Fig. 5: Typing error corrector: (**a**) Button "1" is recognized as "4" by mistake in the first round. (**b**) If the group centroid is c_2, button "1" is recognized incorrectly again in the second round, but the joint decoder can correct this error. However, joint decoder fails if the centroid is c_2'. (**c**) Keyboard rearranger shuffles keyboard layout to increase button spatial diversity.

to (x_c, y_c) for this keystroke. The device then displays an "x" on the screen to inform this typing to the user.

B. Typing error corrector

After all characters are typed, password assembler concatenates them as the password and verifies to the application. The verification however could get rejected if the password contains typing errors. In this case, the user needs to type them again, but incorrect characters may still occur in the new typing round. As the password content is not shown due to the privacy concern, the correctness of each typed character is unknown to the user. If the input terminates only when an intact password is obtained, it may dramatically impair the typing efficiency and prolong the input delay.

1) Joint decoder: **Observation.** We have the following observation to address this issue. Suppose the first character of a password is "1". Unfortunately, the centroid for the first keystroke is estimated as c_1 in the first round of typing, as illustrated in Figure 5(a), which will cause button "4" to be typed. However, we note that although this keystroke detection is incorrect, the intended button, *e.g.,* button "1" in Figure 5(a), is not far away from centroid c_1. Even the keystroke detection for this character is incorrect in the second round of typing as well, the intended button is still in the vicinity of the estimated centroid, *e.g.,* centroid c_2 causes button "2" to be typed in Figure 5(b), but the intended button "1" is still close to c_2. As a result, typing the same character within a password, only the intended button can be *consistently* close to the estimated gaze group centroids cross different rounds, *e.g.,* c_1 and c_2 in Figure 5(a) and (b), respectively.

Solution. This spatial correlation provides us an important hint to correct typing errors even the password plain text is not visible to the user. The idea is to leverage the *cumulative* distances from group centroids to each button cross different rounds. After the m-th retry, we propose the following joint decoding approach to recognize each character $\tilde{j}$ in the password:

$$\tilde{j} = \arg\min_j \sum_{i=1}^{m+1} d(c_i, j), j = 1, 2, \ldots, N, \qquad (2)$$

where $d(c_i, j)$ represents the distance between centroid i and the center of button "j" and N is the total number of buttons on the screen. For the first character in the example of Figure 5, solving above equation (when $m = 1$ and $N = 12$) leads to $j = 1$, which is the intended button to be typed. Joint decoding thus provides one more password candidate in addition to those derived from each typing round directly.

2) Keyboard rearranger: Although the joint decoder can leverage the typed information cross multiple rounds to correct typing errors, it may still fail in certain scenarios. For instance, if the group centroid is estimated as c_2' in Figure 5(b), the output of Eqn. (2) is button "4" instead of "1". To address this issue, we propose to further reshuffle the keyboard layout for each new typing round. By doing so, the mutual distance between different buttons is enforced to be changed, and it becomes rare that the group centroids are consistently close to a same *incorrect* button cross different rounds. In light of this, Eqn. (2) could fully leverage the spatial diversity in the error correction. For example, if the keyboard layout is rearranged as Figure 5(c) in the second round, $d(c_1, 1) + d(c_2'', 1)$ is less than any $d(c_1, j)+d(c_2'', j)$, where j indicates all other buttons on the screen, and button "1" can be selected correctly.

To permutate buttons in different rounds, the pseudo-random arrangement is a feasible solution and a set of optimal keyboard layouts can also be determined. In particular, denote $\mathbf{B}^m = [b_{i,j}^m]$, where $b_{i,j}^m$ represents that button i accommodates character j on the m-th layout, and $m = 1, 2, \ldots, M + 1$, where M is the total number of retries allowed to correct the typing errors. The optimal layouts for each typing round can be determined by maximizing the minimum cumulative inter-button distances cross $M + 1$ rounds as follows:

$$\max_{\mathbf{B^1},\mathbf{B^2},\ldots,\mathbf{B^{M+1}}} \{\min_j \sum_{m=1}^{M+1} d(b_{u,j}^m, b_{v,j}^m)\}. \qquad (3)$$

The above problem could be solved off-line in advance to derive the optimal layouts, which is an one-time effort. When users need a new round of typing, the derived layout for this round can be loaded to use directly. As §V reveals, a small number of error correction rounds, *e.g.,* $M = 1$, could be sufficient in practice. Users can thus familiarize with few additional layouts (from pseudo-random arrangement or Eq. (3)) in advance and smoothly type for the error correction.

C. Frame selector

Frame selector in the gaze engine (Figure 3 on p. 3) ensures the frame quality for the gaze tracker to derive more reliable gaze points in dynamic environments. When device shakes due to motions, although users has the instinct to track the stared button, the generated frames could get blurred, which will degrade the gaze tracking accuracy.

Observation. To understand this impact, we experiment in three typical cases for iType. We record accelerometer readings (denoted as *acc*) to indicate the mobility level and calculate the number of detectable image features to quantify the frame quality, *e.g.,* more features indicate a sharper frame [31]. Figure 6(a) depicts the result when a device is shaken (at index 20 on the x-axis). Figure 6(b) and (c) report the results when a user is moving and on a vehicle, respectively. Figure 6 shows

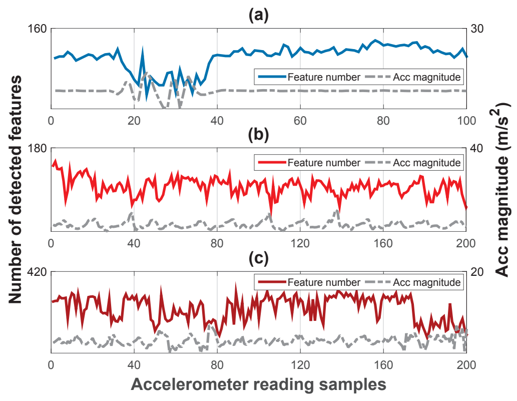

Fig. 6: Relation between frame quality and device motions. (**a**) Shaken by hand. User is (**b**) moving and (**c**) on the vehicle.

that the frame quality (sharpness) relates to the device mobility level, consistent with the recent observation in [31].

Front camera of a mobile device generates frames at fixed rate r, *e.g.,* 30 to 120 fps (frames per second). Nevertheless, due to the processing delay, the effective gaze point generation rate $\tilde{r}$ is less than r, and we denote $r/\tilde{r} = v$. It means that to derive one gaze point, *i.e.,* (x, y) coordinates, v new frames are generated. Gaze points can thus be derived using every v new frames. However, in dynamic environments, after processing the i-th frame, the newly generated $(i+v)$-th frame could have a highly unclear frame quality.

Solution. We leverage accelerometer readings to select the frames experiencing least external motions with the best expected quality. As frame generation rate r is higher than gaze point generation rate $\tilde{r}$, this principle can be instantiated by buffering sufficient (*e.g.,* b) fresh and high-quality frames that are not applied for gaze tracking yet. Specially, when a new frame is generated from front camera and the previous frame is still under processing, the new frame is pushed into the buffer if the buffer is empty; Otherwise, the new frame could replace the existing frame in case its associated acc value is smaller. When the processing of the previous frame is completed, frame selector feeds the buffered frame into gaze tracker as input to derive the gaze point. By doing so, a fixed frame buffer size will be sufficient for selecting better quality frames, and this design is compatible with various devices' hardware specifications, *e.g.,* camera's frame rate and CPU's processing delay. In summary, frame selector works as follows:

Step 1. A new frame is generated from camera.
Step 2. Push new frame into buffer if empty; Otherwise, keep the frame with a smaller acc value associated.
Step 3. Pop out the buffered frame when the processing of the previous frame is completed.

D. Enhancement layer

1) Virtual button: To handle the case that a password contains repeated characters, *e.g.,* "...11...", we propose virtual button, where some external item, *e.g.,* the front camera itself, device home button, unused typing area, etc., can be viewed as a functional button for the separation purpose. Thus, to input repeated characters, like two "1"s, the virtual button needs to be typed in between, *e.g.,* "1" $\rightarrow$ virtual button $\rightarrow$ "1".

As discussed in §III-A2, after finishing one keystroke, the user's gaze may delay on the previous button. Virtual button can also ensure that such residual gaze points will not type the previous button one more time by mistake, since the virtual button is not recognized in between.

2) Flying button: Different buttons could have different numbers of neighboring buttons. Since a typing error usually occurs as the result that one neighbor of the intended button is selected by mistake, more neighbors imply more typing error candidates, *e.g.,* button "5" in Figure 3. In iType, we introduce an extra copy for this button so that the new copy flies along certain trajectory at a constant speed. As flying button has no explicit neighbors, user can also track its flying to type by the trajectory matching. We omit design details due to page limit.

IV. iType Implementation and Configuration

We implement iType on the iOS mobile platform, including all the components introduced in §III, and adopt iPhone 6 and 6 Plus as primary implementation devices.

Gaze engine. We develop a mobile gaze tracker based on a state-of-the-art gaze tracker design [1] using ordinary cameras [12] on Mac OS X from the gaze tracking domain. To train a gaze tracker, the user sequentially looks at a set of stimulus points on the screen and the training is completed within about 35s. The trained gaze tracker is locally stored and utilized for the typing purpose. To derive the gaze (x,y) coordinates for each frame, the gaze engine tracks the user's face and chops sub-images of the eye area [14]. To further enhance the gaze tracking performance, the development in [12] contains the following techniques: the user's eye area on each frame is scaled to a consistent predefined eye image size for alleviating the mismatch, the eye blink frames are excluded and the image normalization is conducted to minimize illumination impacts, which are also included in our implementation. Reported accuracy in Figure 2(a) and (b) is already the performance using these enhancing techniques.

After iType starts, front camera generates frames at 30fps. However, Figure 7(a) shows that the delay to locate the user's face and then derive one gaze point is about 364ms even a recent A8 CPU chip is used, which leads to 2.7 gaze points per second merely. The major processing overhead originates from the face and eye detection, and the lightweight optical flow technique can be adopted [25] to track the user's face and eyes after they are detected for the first time. Figure 7(b) implies this processing can be accelerated by 6.97x to 8.93x.

iType engine. We develop keystroke detector, typing error corrector and enhancement layer three components in the implementation of the iType engine. We also study the two parameters used in the design but not investigated yet: window size w for transitional gaze remover and the KLD similarity threshold τ, and conduct an empirical configuration for these two parameters, *e.g.,* $w = 5$ and $\tau = 0.2$. Due to the page limitation, we omit these experimental details.

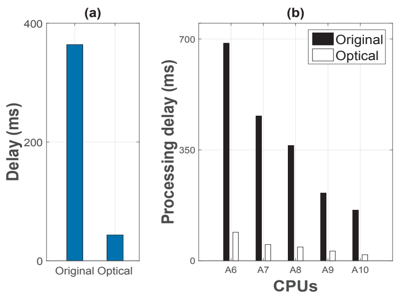

Fig. 7: Processing delay to derive one gaze point a one frame. (**a**) Comparison on the A8 chip. (**b**) processing delays on five different chips.

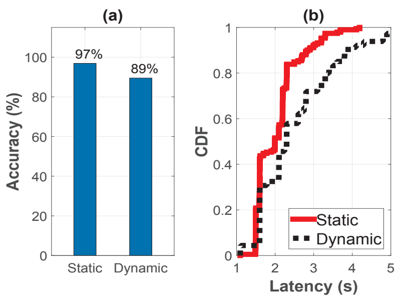

Fig. 8: Individual keystroke detections. (**a**) Keystroke detection accuracy. (**b**) CDF of latency to type each individual character.

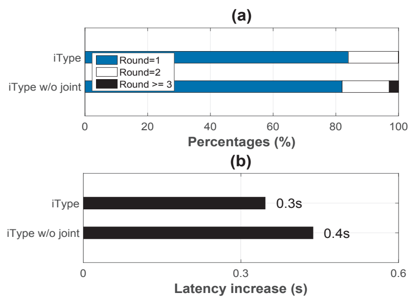

Fig. 9: Overall iType typing performance. (**a**) Typing rounds. (**b**) Increase of per-character latency cross all typing rounds.

V. Performance Evaluation

A. End-to-end performance

Individual keystroke detections. We first evaluate the performance of individual keystroke detections. We experiment in static and dynamic two scenarios and focus on the evaluation on *accuracy* and *latency* two performance metrics. We generate random passwords with 5 to 8 characters and input the passwords using iType.

Accuracy. We compare each recognized password with its ground truth to examine the accuracy of individual keystroke detections. Figure 8(a) shows that keystroke detection in iType is accurate, *e.g.,* 97% and 89% accuracy in static and dynamic environments, respectively. The performance deterioration in the latter case is possibly because dynamics could distort the parameters of the Gaussian distribution of gaze points and also intensify the variation of the relative device-to-eye position.

Latency. Figure 8(b) shows the latency performance that iType achieves to type one character, *i.e.,* the latency between the ends of two consecutive keystrokes. From the result, we observe that iType is responsive for the keystroke typing, where the latency is $2.0s$ and $2.6s$ on average in static and dynamic environments, respectively.

We note that the gaze based input designs are inherently slower than traditional keyboard based input methods, which is dominated by the responsiveness of human's eyes. However, as iType is positioned as a secure channel (or alternative) to enhance the typing privacy, instead of replacing existing input methods, and the password normally has a short length, Figure 8 suggests that iType could achieve reasonable latency performance for such a purpose.

Overall typing performance. Next we evaluate the overall typing performance of iType. When the assembled password contains typing errors, the user has to type it again to correct them. Figure 9(a) shows that all the passwords can be correctly obtained within the first two rounds of typing, especially, 84% of passwords are already correct after the first round, and all correct passwords are obtained after the second attempt in this experiment. Due to additional typing rounds, the effective latency to correctly input one character is increased. Figure 9(b) shows that it increases by $0.3s$ on average. In Figure 9, we also examine the efficacy of the joint decoder so that the input is successful only when an intact password is obtained. Figure 9(a) shows that for about 3% cases, at least 3 rounds of typing is needed and the per-character input delay is increased by $0.4s$ on average in Figure 9(b).

B. Micro-benchmark experiments

We then conduct micro-benchmark experiments to understand how variant factors could impact iType's performance.

Impact of input password sequences. In this trial of experiments, we first control the repetition of a same character in the password. For instance, the repetition for "1" in "...11..." is 2 and its input sequence is "...1" $\rightarrow$ virtual button $\rightarrow$ button "1...". We vary repetitions from 2 to 4. From Figure 10(a), we find that compared with purely random input sequences, there is no significant impact on the keystroke detection accuracy. However, as longer repetitions lead to more keystrokes on virtual button, Figure 10(b) indicates that the average latency per character is increased from $2.2s$ to $3.2s$. As passwords usually have few repeated characters due to the security concern in practice, *e.g.,* 2, the impact of input repetitions on iType is thus not substantial.

In Figure 10(c), we further study the impact of the *paging* function during typing. We observe similar performance as in the experiment for repetition, *e.g.,* no obvious influence on the accuracy and the per character latency increases from $2.2s$ to $3.0s$ on average.

Impact of device-to-eye distances. The device-to-eye distance is usually around $25cm$ to $35cm$ when people use mobile devices. In this experiment, we make $30cm$ as the benchmark and vary the distance from $25cm$ to $50cm$ to examine its impact on the typing performance. Figure 11(a) shows that compared with the benchmark, the performance exhibits no remarkable differences within $35cm$, but it drops significantly beyond this distance. As a result, the device-to-eye distance, within a common range like $25cm$ to $35cm$, has a minimal impact on the typing performance.

Impact of head movements. To understand the impact of head movements on the performance, we intentionally introduce three types of head movements during typing: perpendicular

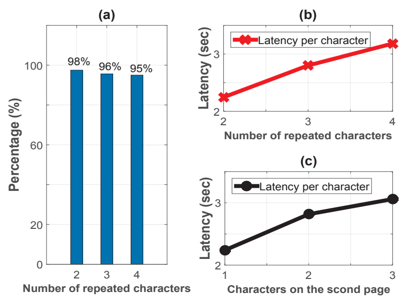

Fig. 10: Impact of various input sequences. (**a**) Keystroke accuracy. Latency due to different characters (**b**) repeated and (**c**) on the second page.

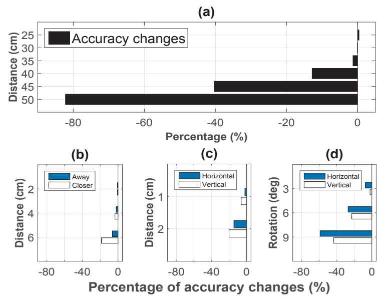

Fig. 11: Impacts of different factors. Accuracy changes due to (**a**) working distances, (**b**) perpendicular, (**c**) parallel and (**d**) rotated head movements.

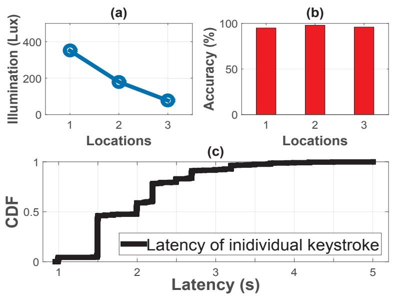

Fig. 12: Impacts of illuminations and user study. (**a**) Illumination levels. (**b**) Detection accuracy at each location. (**c**) Latency of different users.

to the screen in Figure 11(b), parallel to the screen in Figure 11(c), and rotation in Figure 11(d), and compare with the performance without such head movements. From the results, we observe that the first two types of head movements can be tolerated by our typing design to some extent, while iType is more sensitive to the head rotation, *e.g.,* when the rotation is large, *e.g.,* approximately 6° to 9°, the accuracy is dramatically degraded. We also find when head moves back, the performance can be recovered at a certain degree, but this experiment still inspires us to further enhance iType by better handling head movements in the future.

Impact of illuminations. In Figure 12(a), we evaluate the impact of ambient illuminations at different locations, where the illumination varies from more than 1000 to less than 10 Lux. In the experiment, we observe that when the illumination is sufficiently bright, *e.g.,* from 77 to 351 Lux, the keystroke detection is accurate, as depicted in Figure 12(b). When the illumination is excessively strong or weak, many eye details might be lost due to overexposure or underexposure, which could significantly degrade the gaze tracking performance, as the consequence of the typing performance.

User study. To further examine the usability and the security of iType, we conduct a user study. We recruit 5 participants. Each participant is given $15min$ to familiarize with iType and they then test the performance. Throughout the experiment, we find users can achieve good typing performance using iType. Figure12(c) further details their typing speeds of individual keystrokes, where the average latency is also about 2 seconds.

VI. Related Work

Gaze tracking. Gaze tracking techniques fall into two main streams [19]: model based detection and appearance based matching. The former category detects the pupil-iris boundary that works best with near-infrared illumination sources [20]. The latter category requires no special hardware and works with common cameras [19]. iType falls into the second category. Gaze tracking is mature for wearable devices already, like iShadow [19], iGaze [30] and CIDER [20]. This technique, however, still suffers low accuracy on mobile devices. Although some existing efforts have been made to develop gaze tracking using mobile devices, *e.g.,* [14, 15], this crucial issue is not explicitly addressed yet. Some commercial products, *e.g.,* Tobii [3], can handle this issue yet incurs additional hardware, solution cost and relatively high energy consumption [20]. iType thus requires effective and generic solutions to precisely infer the typed information atop the limited accuracy of gaze tracking on mobile devices.

There are also some software-based gaze tracking SDKs for mobile platforms from companies [2, 4], while Snapdragon SDK [2] supports a limited set of Android-based devices using the Snapdragon processors merely and Umoove's EyeMovementSDK is not open to the general public yet [4].

Privacy leakage and protection. Recent studies show serious privacy leakage issues on mobile devices [18, 21, 29], through acoustic or acceleration signals. To prevent such privacy leakage, existing efforts mainly leverage human's biometrics, *e.g.,* voices, faces [9], fingerprints [24], iris and its move [7, 26], phone usage features [11], etc., and also user's behavioral features, like the user's gestures on touch screen [23], phone usage patterns [27], keystroke patterns [22] and the multimodal using both gaze and touch [16]. These solutions usually focus on some dedicated services, *e.g.,* device authentication, and/or require specialized hardware. Different from existing works above, iType provides a secure channel for many conventional services that require direct textual input of privacy.

Typing on mobile devices. Recent studies have various novel typing approaches, *e.g.,* acoustic signals [28], Wi-Fi [6, 8], accelerometers [5], etc. These designs focus on improving the typing efficiency or exploring alternatives for physical keyboards on mobile devices, which however are not for solving the privacy leakage issue of the text-entry.

VII. Conclusion

We have described iType that uses eye gaze to enhance typing privacy on commodity mobile devices. We have devised effective techniques to address a series of design challenges, covering accuracy, latency and mobility several aspects. We have consolidated our techniques and implemented iType on the iOS platform. Experiments show iType achieves high typing accuracy within reasonable short latency in variant environments. A user study further verifies the efficacy of the iType design. In the future, we plan to conduct a larger-scale field study to obtain more feedbacks from different users.

Acknowledgment

This work is supported in part by the ECS grant from Research Grants Council of Hong Kong (Project No. CityU 21203516), City University of Hong Kong (Project No. 7200480), NSFC Grant No.61572396, National Basic Research Program of China (973 Program) under Grant No.2015CB351705, Singapore MOE AcRF Tier 1 MOE2013-T1-002-005, NTU CoE grant M4081879 and NTU NAP grant M4080738.020.

References

[1] https://github.com/tiendan/OpenGazer.

[2] Snapdragon. https://developer.qualcomm.com/software/snapdragon-sdk-android.

[3] Tobii. http://www.tobii.com/.

[4] Umoove. http://www.umoove.me/technology.html.

[5] S. Agrawal, I. Constandache, S. Gaonkar, R. Roy Choudhury, K. Caves, and F. DeRuyter. Using mobile phones to write in air. In *ACM MobiSys*, 2011.

[6] K. Ali, A. X. Liu, W. Wang, and M. Shahzad. Keystroke recognition using WiFi signals. In *ACM MobiCom*, 2015.

[7] W. X. C. Song, A. Wang and K. Ren. Eyeveri: A secure and usable approach for smartphone user authentication. In *IEEE INFOCOM*, 2016.

[8] B. Chen, V. Yenamandra, and K. Srinivasan. Tracking keystrokes using wireless signal. In *ACM MobiSys*, 2015.

[9] S. Chen, A. Pande, and P. Mohapatra. Sensor-assisted facial recognition: an enhanced biometric authentication system for smartphones. In *ACM MobiSys*, 2014.

[10] X. Chen, X. Wu, X.-Y. Li, X. Ji, Y. He, and Y. Liu. Privacy-aware high-quality map generation with participatory sensing. *IEEE Transactions on Mobile Computing*, 2016.

[11] M. Conti, I. Zachia-Zlatea, and B. Crispo. Mind how you answer me!: transparently authenticating the user of a smartphone when answering or placing a call. In *ACM ASIACCS*, 2011.

[12] O. Ferhat, F. Vilarino, and F. J. Sanchez. A cheap portable eye-tracker solution for common setups. *Journal of Eye Movement Research*, 2014.

[13] C. Forbes, M. Evans, N. Hastings, and B. Peacock. Student's t distribution. *Statistical Distributions*, 2010.

[14] C. Holland, A. Garza, E. Kurtova, J. Cruz, and O. Komogortsev. Usability evaluation of eye tracking on an unmodified common tablet. In *ACM CHI*, 2013.

[15] Z. Jiang, J. Han, C. Qian, W. Xi, K. Zhao, H. Ding, S. Tang, J. Zhao, and P. Yang. VADS: Visual attention detection with a smartphone. In *IEEE INFOCOM*, 2016.

[16] M. Khamis, F. Alt, M. Hassib, E. von Zezschwitz, R. Hasholzner, and A. Bulling. Gazetouchpass: Multimodal authentication using gaze and touch on mobile devices. In *ACM CHI*, 2016.

[17] S. Kullback. *Information theory and statistics*. Courier Corporation, 1968.

[18] 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*, 2015.

[19] A. Mayberry, P. Hu, B. Marlin, C. Salthouse, and D. Ganesan. iShadow: design of a wearable, real-time mobile gaze tracker. In *ACM MobiSys*, 2014.

[20] A. Mayberry, Y. Tun, P. Hu, D. Smith-Freedman, D. Ganesan, B. M. Marlin, and C. Salthouse. CIDER: Enabling robustness-power tradeoffs on a computational eyeglass. In *ACM MobiCom*, 2015.

[21] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. R. Choudhury. Tapprints: your finger taps have fingerprints. In *ACM MobiSys*, 2012.

[22] F. Monrose, M. K. Reiter, and S. Wetzel. Password hardening based on keystroke dynamics. *Springer International Journal of Information Security*, 2002.

[23] N. Sae-Bae, K. Ahmed, K. Isbister, and N. Memon. Biometric-rich gestures: a novel approach to authentication on multi-touch devices. In *ACM SIGCHI*, 2012.

[24] M. Shahzad, A. X. Liu, and A. Samuel. Secure unlocking of mobile touch screen devices by gestures: You can see it but you can not do it. In *ACM MobiCom*, 2013.

[25] R. Szeliski. Handbook of mathematical models of computer vision, 2005.

[26] S. Thavalengal, P. Bigioi, and P. Corcoran. Evaluation of combined visible/nir camera for iris authentication on smartphones. In *IEEE CVPR Workshops*, 2015.

[27] G. Wang, T. Konolige, C. Wilson, X. Wang, H. Zheng, and B. Y. Zhao. You are how you click: Clickstream analysis for sybil detection. In *USENIX Security*, 2013.

[28] J. Wang, K. Zhao, X. Zhang, and C. Peng. Ubiquitous keyboard for small mobile devices: harnessing multipath fading for fine-grained keystroke localization. In *ACM MobiSys*, 2014.

[29] 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.

[30] L. Zhang, X.-Y. Li, W. Huang, K. Liu, S. Zong, X. Jian, P. Feng, T. Jung, and Y. Liu. It starts with iGaze: visual attention driven networking with smart glasses. In *ACM MobiCom*, 2014.

[31] Y. Zheng, G. Shen, L. Li, C. Zhao, M. Li, and F. Zhao. Travi-Navi: Self-deployable indoor navigation system. In *ACM MobiCom*, 2014.