aLeak: Privacy Leakage through Context-Free Wearable Side-Channel

Yang Liu, Zhenjiang Li
Department of Computer Science, City University of Hong Kong, Hong Kong, China
yliu562-c@my.cityu.edu.hk, zhenjiang.li@cityu.edu.hk

***Abstract*—We revisit a crucial privacy problem in this paper — can the sensitive information, like the passwords and personal data, frequently typed by user on mobile devices be inferred through the motion sensors of wearable device on user's wrist, *e.g.,* smart watch or wrist band? Existing works have achieved the initial success under certain context-aware conditions, such as 1) the horizontal keypad plane, 2) the known keyboard size, 3) and/or the last keystroke on a fixed "enter" button. Taking one step further, the key contribution of this paper is to fully demonstrate, more importantly alarm people, the further risks of typing privacy leakage in much more generalized context-free scenarios, which are related to most of us for the daily usage of mobile devices. We validate this feasibility by addressing a series of unsolved challenges and developing a prototype system *aLeak*. Extensive experiments show the efficacy of *aLeak*, which achieves promising successful rates in the attack from more than 300 rounds of different users' typings on various mobile platforms without any context-related information.**

I. Introduction

Rich sensors on electronic devices, *e.g.,* wearables, could generate valuable sensory data [14], [21], for designing a corpus of useful applications, *e.g.,* healthcare [16], [20], driving monitoring [6], fitness guidance [4], authentications [17], [5], mobile social networks [19], etc. We have thus nowadays witnessed their increasing popularity and high penetration into our daily life. However, studies, like [14], [13], [8], also unveil the *double-edged* fact of these sensory data recently, which could form side-channels and leak aspects of people's vital information. For instance, most our personal accounts now can be on-line accessed. It is hence inevitable for people to type private information explicitly [7], *e.g.,* passwords, personal particulars, security codes, etc., on various mobile platforms, *e.g.,* smart phones, POSs, ATMs, door entrance panels, etc. Therefore, one crucial and specific piece of the problem, which may relate to most of us, is — *whether such sensitive yet frequently typed information can be inferred through the motion sensors of wearable, like a smart watch or wrist band, on user's wrist?* If so, the consequence is serious, as the barrier to launch this side-channel attack is trivial [13], [8].

This paper, of course, is not the first attempt at this problem. Instead, we aim to comprehensively unveil the potential possibility of using wearable sensors to sacrifice user's typing safty, and fully demonstrate (more importantly alarm people) the further privacy leakage risks that may not be viable before, by addressing a series of unsolved challenges.

Challenges. To launch this side-channel attack, attackers need to precisely recover each piece of wearable's moving displace-

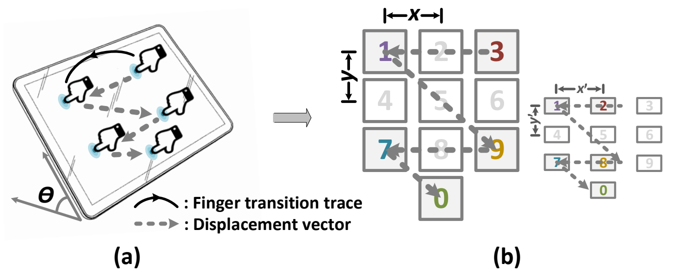

Fig. 1: **Illustration of wearable side-channel attack.** (**a**) Recover all displacement vectors and the keypad plane posture angle θ. (b) Match the moving trajectory on the keyboard with correct (left) and incorrect (right) keyboard sizes, incurring different results, *e.g.,* "31970" (correct) yet "21870" (incorrect).

ment *vectors*, which capture user's finger transitions between two keystrokes, as in Fig. 1(a). These vectors are expected to be used to reconstruct the keypad plane first, since the keypad plane can be in an *arbitrary* posture, *e.g.,* the device, such as smart phone or POS machine, is held in user's hand during the typing. As these vectors are not strictly co-plane due to various errors, they are then projected onto the keypad plane to obtain wearable's moving trajectory along this plane. By further matching the recovered trajectory to the keyboard layout, the typed information can finally get "decoded", *e.g.,* "31970" in Fig. 1(b-left). During this process, following three challenges will be encountered.

Inaccurate motion recovery. Wearable's displacement vectors are derived from the accelerometer data from wearables, while the result is naturally inaccurate as the double-integral could rapidly accumulate and amply acceleration errors. Recent remedy methods, *e.g.,* the mean-removal [14], become much less effective, since the zero velocity is not guaranteed at both ends of each vector and finger's transition follows a curved trace (illustrated in Fig. 1 and detailed in §II-B). Hence, the keypad plane (with an arbitrary posture) cannot be reliably reconstructed in the first place, and erroneous vectors cannot truthfully reflect wearable's moving trajectory neither — it is non-trivial to launch above side-channel attack.

Unknown keyboard size. Even the keypad's posture was precisely derived finally, the typed information is still not immediately decodable, since the keyboard size, *e.g.,* x and y values in Fig. 1(b), is unknown, lacking the correct decoding (or matching) reference. The keyboard size can vary quite differently cross platforms, *e.g.,* smart phones, tablets, POSs, door entrance panels, APPs, etc. Using any default keyboard size, we can always derive a most likely result (w.r.t. this size), which however tends to be wrong, *e.g.,* the wrong result "21870" in Fig. 1(b-right) when an incorrect keyboard is used. Thus, without this meta information, the attack stagnates again.

Information ambiguity. After the keyboard size, *e.g.,* x and y values in Fig. 1(b), could be eventually figured out, wearable's displacement trajectory may still be embedded into the keyboard layout in different ways, leading to different results. Such an ambiguity also needs to be effectively removed.

To overcome above issues, existing works [13], [8] achieve the initial success under certain known **contexts** about user's typing: 1) the horizontal keypad plane, 2) the known keyboard sizes, *e.g.,* ATM or POS panels, 3) and/or the last keystroke on a fixed "enter" button. However, without explicitly addressing above challenges, it is unclear about the potential periphery of this side-channel attack. One concrete concern, for instance, is whether our typing privacy on variant mobile platforms, which violate above context-aware assumptions (*e.g.,* unknown keypad postures and keyboard sizes), can still be compromised. We believe that this investigation is more critical, as most our personal accounts now can be accessed on-line, and people are likely to type private information, *e.g.,* passwords, personal data, security codes, etc., in such scenarios.

Contributions. In this paper, we present *aLeak* to explore the possibility of the context-free side-channel attack through the motion sensors from wearables, which could be smart watches or bands worn on user's wrist. Smart watch is designed for user's right or left either hand, while many people now wear it on the right hand, without the concern that it is easier to adjust time for traditional watches on the left hand. In addition, many people also tend to wear a smart band on the right hand while a common watch is on the left hand [13]. Therefore, when wearables and the finger move concurrently during the typing, by accessing the data from wearable motion sensors, *e.g.,* accelerometers and gyroscope, (attack model is detailed in §II), adversary could launch the wearable side-channel attack.

We deeply analyze the wearable sensory data and propose a series of key techniques to address aforementioned challenges in *aLeak*. To demonstrate the efficacy of *aLeak*, we conduct extensive experiments with 5 users wearing the smart watch and act as the adversary to attack more than 300 rounds of users' password inputs on various mobile platforms, including smart phones, tablets, door entrance input panels, telephones and POS, with the keypad posture angles changing from $0°$ to $90°$. Experiments show that without any context information, *aLeak*'s top-1 successful attacking rate is 45% and the top-5 accuracy increases to 94%, while the top-5 accuracy of the most recent RCCS [13] is 15% merely (even partial context information is provided for RCCS otherwise its attack cannot proceed). In summary, the contributions of this paper are:

- Demonstrating the possibility to leak user's typing privacy in a context-free manner and unveiling (more importantly alarming people) the further privacy leakage risks that may not be viable before.
- Proposing a set of key techniques to address, inaccurate motion recovery, unknown keyboard size and information ambiguity, three major challenges in *aLeak*.
- Developing a prototype system and conducting extensive experiments with 5 volunteers, by attacking their more than 300 rounds of inputs on a variety of mobile platforms with different keyboard sizes and keypad postures.

Organization. In the rest of the paper, we introduce the wearable side-channel attack preliminary in §II and elaborate the *aLeak* design in §III. The evaluation are conducted in §IV. We review related works in §V before the conclusion §VI.

II. Preliminary and Challenges

Although biometric sensors, *e.g.,* Touch ID, are widely adopted by mobile devices to avoid a direct password input, they are not generic enough for many daily services on the device that require an explicit private information input [7], such as 1) PINs of many personal accounts for an on-line access, 2) personal particulars, *e.g.,* phone numbers, credit card security codes and date of birth, provided during transactions, and 3) passwords of mobile devices without biometric sensors. In addition, such an explicit typing can also commonly occur on many third-party terminals, like 4) depositions on ATMs, 5) transactions on POS machines, and 6) password typing on door entrance panels, etc.

For all these potential privacy leakage cases, the posture of the typing keypads can be arbitrary, *e.g.,* mobiles or POS machines are hold in user's hands, with variant keyboard sizes. Hence, if the design challenges stated in §I can be addressed, user's typing privacy will easily get compromised and scarified (which are not viable before). In the rest of this section, we will elaborate attack model (§II-A) and design challenges (§II-B).

A. Attack model

Similar as all existing studies [14], [13], [8], the adversary in *aLeak* could hack the same set of motion sensor data (accelerometers and gyroscope) from wearables on user's wrist to launch the side-channel attack. There are two common ways to illegally access such sensitives motion data [8], [13]:

Installing malicious applications. The adversary can trick a user (*e.g.,* victim) to install malicious applications on smart watch or phone without victim's notice [13], *e.g.,* embedding malicious codes in popular applications and making them free in the APP market [8]. The malicious APP runs in background and would send the motion sensor data to adversary's server when Wi-Fi is available.

Sniffing blue-tooth packets. A wearable device is usually paired with victims' mobile phone through blue-tooth and the wearable needs to report its sensor data to the phone for the data synchronization and logging purposes. Recent studies find that the adversary could overhear the transmitted blue-tooth packets in the vicinity of the victim using wireless sniffers [13], [11], [12] to recover the motion data.

With the harvested motion data from victim's wearable, the adversary needs to further cope with the following challenges to enable the context-free side-channel attack.

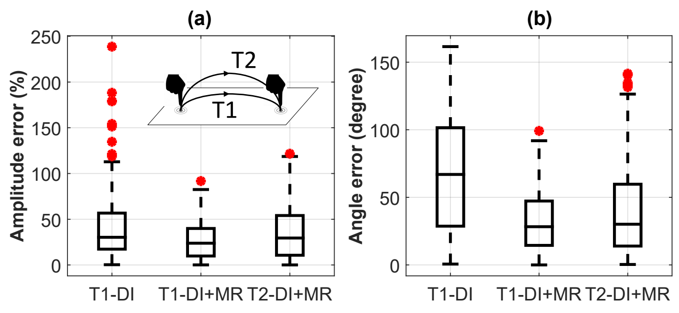

Fig. 2: **Accuracy of derived vectors.** (**a**) Amplitude and (**b**) angle errors, where "DI" and "MR" stand for double-integral and mean-removal, respectively, and the maximum heights of the two traces T1 and T2 are about $1cm$ and $3cm$, respectively.

B. Design challenges

1) Inaccurate motion recovery: The adversary needs to precisely recover wearable's displacement vectors in this attack. To understand the accuracy desired, considering the keyboard size x and y equal as an example, if we, in this case, want to reliably identify the two keystrokes covered by one vector as in Fig. 1, vector's amplitude (*e.g.,* length) and angle (between two consecutive vectors) errors should be less than 28% and 19°, respectively (the derivation detail is omitted here). Of course, errors could further accumulate cross vectors in a moving trajectory. Accuracy is thus expected to be even higher.

We conduct experiments to examine this feasibility. Fig. 2 shows that the direct double-integral (T1-DI) is highly inaccurate. The percentage of its amplitude error is 30.3% on average and can be up to 112.8% (w.o. outliers). Meanwhile, the angle error is 67.0° on average and can be up to 160°. In Fig. 2, we further apply the mean-removal technique [14] to improve the accuracy, where the amplitude and angle errors (T1-DI+MR) can be reduced to 23.8% and 28.3° on average, respectively.

The mean-removal becomes much less effective here due to the reason that the zero-velocity condition is not strictly satisfied at both ends of each vector. We also notice a more serious reason that the performance further deteriorates — user's finger (also the wearable) usually moves following a curve in the air between two keystrokes. From our experiment, we find as the moving trace becomes more curved, the accuracy keeps degrading. As our finger's moving trace normally has a curved level with a height between $1cm$ (T1) and $3cm$ (T2) as in Fig. 2(a) in the typing, *e.g.,* about 27% amplitude and 29° angle errors on average, it surely incurs an unsatisfactory motion recovery. In this case, the keypad plane may not even be reliably recovered in the first place and the moving trajectory can also be easily distorted.

2) Variant keyboard sizes: On the other hand, the keyboard size could vary remarkably cross different mobile platforms [1], [2] as Table I shows. From the table, we can see that the keyboard size could vary up to 6x and 5x times along the x and y directions, respectively.

To give some concrete examples, we measure the keyboard sizes for five popular mobile platforms, *e.g.,* Samsung Galaxy S6, iPhone 7 plus, a metal ATM keypad, a numeric keypad, and a door entrance panel. The inter-button distance could vary up to 2.3x, which is already highly diverged even from five examples only. In this attack, without knowing the keyboard size, even the moving trajectory was precisely derived finally, the typed information is still not decodable, because the correct decoding (or matching) reference is lacking.

Category	Range of x (horz.)	Range of y (vert.)
Mobile devices	$16 \sim 91$	$19 \sim 45$
Numeric keypads	$23 \sim 53$	$22 \sim 23$
ATMs / POSs	$29 \sim 35$	$18 \sim 38$
Door entrance panels	$15 \sim 28$	$30 \sim 97$

TABLE I: The feasible x and y ranges between two adjacent buttons cross different mobile platforms (unit: mm).

III. System Design

In this section, we elaborate three core component designs in *aLeak* to address above challenges, including 1) wearable's moving trajectory recovery in §III-A, 2) keyboard size derivation in §III-B, and 3) typed information inference in §III-C.

A. Moving trajectory recovery

This component converts wearable's motion data, *e.g.,* accelerometers and gyroscope, to its moving trajectory along the keypad plane, with three steps. We first divide the highly dynamic-varying motion data into segments for deriving wearable's displacement vectors (in §III-A1). Then we migrate the motion data inaccuracy issue to precisely derive keypad's unknown posture, such that wearable's moving trajectory on the keypad can finally get recovered (in §III-A2).

1) Motion data segmentation: Acceleration data[1] $\{a_i\}$ sequence should be first divided into segments, and each segment corresponds to one piece of wearable's displacement vectors. In other words, $\{a_i\}$ needs to be partitioned at the moments of keystrokes. However, due to high dynamic acceleration varyings and inevitable jitters, we find how to first *automatically* and *reliably* identify these delimiters becomes not so trivial.

Keystrokes could incur prominent acceleration changes [13], *e.g.,* $\{||\vec{a}_i||_2\}$, as the (green) circles marked in Fig. 3(a). Therefore, one natural attempt is the adoption of thresholds, but we find that the keystroke strength is highly user-dependent, which may even vary substantially for the same user, *e.g.,* typing on different devices or with different keypad postures. Moreover, some acceleration peaks due to dynamics and jitters could also have very large strengths, *e.g.,* the triangle after "Click 3" in Fig. 3(a). A universal cutting off is thus hardly to be determined using the threshold.

To cope with this issue, we find that the password typing is normally fluent and rapid (*e.g.,* the user is familiar with the password), which leads to a relatively stable typing *pace*. This inspires us to look at the set $\{a_i\}$ in the frequency domain to identify the dominating frequency f_{type}, as Fig. 3(b) depicts,

[1]The hacked raw accelerations $\{acc_i\}$ are in wearable's coordinate system. They are converted to a global coordinate system by referring to the concurrent gyroscope readings [14] and we denote the converted accelerations as $\{a_i\}$. This global coordinate system may have a fixed offset along the horizontal plane with the earth coordinate system, but it has no impact to *aLeak* design.

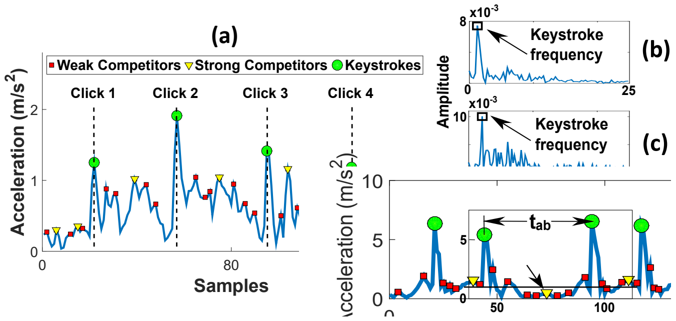

Fig. 3: **Motion data segmentation.** (**a**) Instrumental example. (**b**) FFT on $\{a_i\}$ to extract user's typing pace. (**c**) Typing pace extraction from 5 finger transitions with 2 being prolonged. (**d**) Acceleration peaks are small when the finger hangs in the air.

which corresponds to the average time interval t_{pace} between two keystrokes, *e.g.,* the typing pace. As a result, we have:

$$t_{pace} = 1/f_{type}. \tag{1}$$

From Fig. 3(a), we can see that all keystrokes occur at the acceleration peaks and they are also greater than their neighboring peaks (from both left- and right-hands). These two criteria can exclude many weak "competitors", *e.g.,* the (red) dot peaks in Fig. 3(a), and we denote the remaining peaks as a set $\{p_j\}$, which contains both the accelerations for keystrokes, *e.g.,* the circles in Fig. 3(a), and the strong "competitors", *e.g.,* the triangles in Fig. 3(a). Now, the interval t_{pace} in Eqn. (1) could further provide a proper temporal duration to filter out these strong competitors by comparison in Algorithm 1, since they are *relatively* small compared with adjacent keystrokes. As t_{type} is an average typing pace, in line 6 of the algorithm, we provide a margin α for the comparison, where α is set as 0.75 empirically in our current *aLeak* implementation.

Even the user may suddenly stop during the typing to recall the next password character (in case it is forgotten), as long as the typing pace still dominates, these prolonged delays could not impair the overall frequency behavior, as Fig. 3(c) shows, where 2 out of 5 finger's movings are prolonged. However, in this case, some peaks within the prolonged typing intervals might be mis-detected as keystrokes, *e.g.,* the marked yellow triangle within t_{ab} in Fig. 3(d). Nevertheless, fortunately, the accelerations within such a prolonged duration are very small, as the finger is relatively stable when it is hanging in the air. We can thus filter out all the peaks near zero (*e.g.,* less than 5% of the largest peak) in $\{p_j\}$ first before using Algorithm 1.

After Algorithm 1 completes, we denote its returned accelerations as $\{s_j\}$ and these items correspond to the keystrokes. By viewing each s_j as the delimiter, we can divide the original acceleration sequence $\{a_i\}$ into segments. By applying double-integral with mean-removal to all a_i from each segment, we can derive its displacement vector, $\vec{v}_l$. Note that all vectors $\{\vec{v}_l\}$ are in the same absolute coordinate system as $\{s_j\}$.

2) Wearable moving trajectory: With the segmented motion data, the next task is to further cope with its inaccuracy issue to precisely derive keypad's unknown posture, *e.g.,* the *angle* between the keypad and horizontal planes as the θ shown in Fig. 1(a), such that wearable's moving trajectory along the keypad plane can be recovered.

Algorithm 1: Keystroke Identification

```
1 input: peak set $\{p_j\}$; calculated $t_{type}$ from Eqn. (1);
2 output: keystroke set $\{s_j\}$;
3 while the size of $\{p_j\}$ changes do
4     for each item in $\{p_j\}$ do
5         calculate the time interval $t$ to the next item;
6         if $t < \alpha \cdot t_{type}$ then
7             remove the item with a smaller amplitude;
8 return $\{s_j\} \leftarrow \{p_j\}$;
```

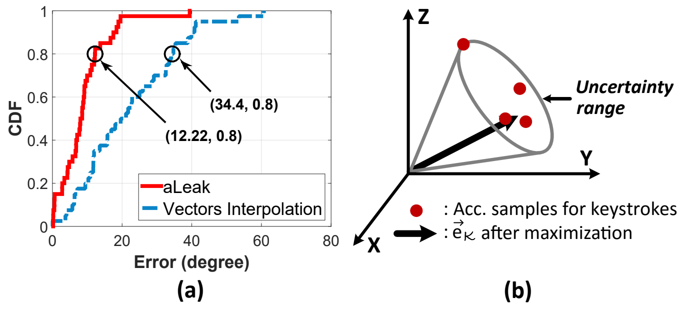

Fig. 4: **Keypad plane reconstruction.** (**a**) CDF of keypad plane reconstruction errors. (**b**) Estimation of the $\vec{e}_{\mathcal{K}}$ direction.

As we have derived each vector $\vec{v}_l$, one natural solution is to construct an interpolated plane, by minimizing the average distance to each $\vec{v}_l$ by the least square, as the keypad plane. However, due to the inaccuracy of $\vec{v}_l$ (for both the amplitude and angle errors as unveiled in Fig. 2), the plane reconstruction performance is not satisfactory, *e.g.,* the 80^{th} percentile error is 34.4° and it can be up to 60.3° in Fig. 4(a).

To migrate the inaccuracy from $\vec{v}_l$, we observe a keystroke involves two consecutive but *opposite* finger movements along the *perpendicular* direction ($\vec{e}_{\mathcal{K}}$) of the actual keypad plane ($\mathcal{K}$), *e.g.,* touching on and then releasing from the keypad. Thus, wearable's acceleration changes are maximized at keystrokes, *e.g.,* the moments of each item in $\{s_j\}$ returned from Algorithm 1, along the $\vec{e}_{\mathcal{K}}$ direction. Hence, instead of figuring out $\mathcal{K}$ from the less accurate $\{\vec{v}_l\}$, we can leverage $\{s_j\}$ to approximate $\vec{e}_{\mathcal{K}}$ first. As each acceleration reading s_j is read from the sensor directly without any integral operations to cumulate errors, the derived $\vec{e}_{\mathcal{K}}$ from $\{s_j\}$ is more reliable and accurate than $\mathcal{K}$ directly from $\{\vec{v}_l\}$. With a high-quality $\vec{e}_{\mathcal{K}}$, precise keypad plane $\mathcal{K}$ can be trivially obtained as they are perpendicular to each other.

Fig. 4(b) shows that the items in $\{s_j\}$ could be diverged within a small cone-like uncertainty range. We can thus find the best $\vec{e}_{\mathcal{K}}$ by maximizing the accelerations at the moments of keystrokes along the final $\vec{e}_{\mathcal{K}}$:

$$max_{\vec{e}_{\mathcal{K}} \in cone} \sum_l ||g(s_j, \vec{e}_{\mathcal{K}})||_2,$$

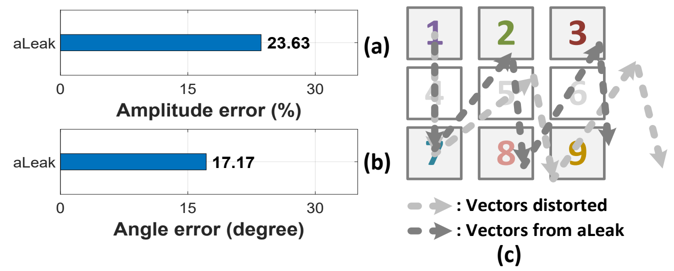

Fig. 5: **Wearable moving trajectory recovery.** (**a**) Amplitude and (**b**) angle errors of the vectors in $\{\vec{o}_l\}$. (**c**) Recovered moving trajectories distorted by different vectors' angle errors.

where $g(s_j, \vec{e}_{\mathcal{K}})$ represents to project s_j to the $\vec{e}_{\mathcal{K}}$ direction. In Fig. 4(b), we see that the 80^{th} percentile and maximum errors of this design can be reduced to $12.2°$ and $17.9°$, respectively.

After the posture of keypad plane is determined, for each vector $\vec{v}_l$, its displacement along keypad's perpendicular direction $\vec{e}_{\mathcal{K}}$ should be zero. However, due to the sensing error, the obtained result is usually non-zero, *e.g.,* d_{res}. Hence, for each vector $\vec{v}_l$, we can calibrate all its accelerations by a factor c, such that the double-integral of $\{c\}$ generates "$-d_{res}$" along the $\vec{e}_{\mathcal{K}}$ direction, to cancel out the non-zero residual displacement. This essentially applies the mean-removal again for the $\vec{e}_{\mathcal{K}}$ direction merely. Using these calibrated accelerations, the newly derived vectors $\vec{o}_l$ will have an improved accuracy and automatically become co-planed on $\mathcal{K}$. The set $\{\vec{o}_l\}$ is thus the wearable's moving trajectory, where each $\vec{o}_l$ is one of wearable's displacements along the keypad plane $\mathcal{K}$.

In Fig. 5(a) and (b), we examine the accuracy of derived $\{\vec{o}_l\}$. The result shows that both the amplitude and angle accuracies are improved, especially for the angular performance, *e.g.,* $17°$ error on average. We find the angle errors are more crucial and sensitive to the end performance, as they can easily distort the trajectory's shape, *e.g.,* as Fig. 5(c) depicts.

Summary. By solving the inaccurate motion recovery issue, this component converts wearable's motion data to its moving trajectory, represented by vectors $\{\vec{o}_l\}$ along keypad plane $\mathcal{K}$.

B. Keyboard size derivation

This component derives the unknown keyboard size, which is represented by the x and y values. We take the most widely adopted 4×3 grid layout as a main instrument to elaborate our design, which in fact can be extended to other grid layouts.

1) The key observation: Although the keyboard size may vary cross platforms, we find that once a user types on one keyboard, the recovered moving trajectory implicitly owns the keyboard size information, based on the following observation.

Observation. If one vector in trajectory $\{\vec{o}_l\}$ is supposed to be parallel to one of keyboard's axises, *e.g.,* if no errors, vector $\vec{o}_1$ is supposed to be parallel to keyboard's y-axis in Fig. 6(a), after we project all other vectors to this direction (nearly the y-axis) and its perpendicular direction (nearly the x-axis), their projected lengths should be approximately *integral multiples* of the keyboard size y and x values, respectively. Fig. 6(c) displays all possible integral multiples in principle when one vector is projected to these two directions.

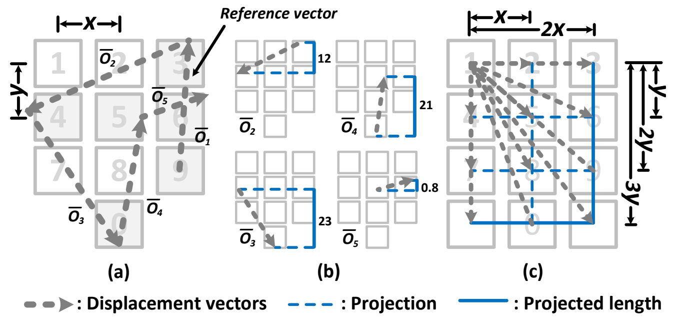

Fig. 6: **Reference vector in keyboard size derivation.** (**a**) $\vec{o}_1$ is selected as reference vector. (**b**) Other vectors are projected to $\vec{o}_1$'s own direction. (**c**) Possible integral multiple values.

We name such a baseline vector for projection, *e.g.,* $\vec{o}_1$, the *reference* vector. In fact, every vector in the moving trajectory could be a reference, while we call a reference to be *qualified* if it is supposed to be parallel to one of keyboard's axises. Fig. 6(b) shows the projected lengths of all other vectors to the qualified reference $\vec{o}_1$'s own direction (y-axis)[2]. However, from each projected length, we cannot derive the keyboard size y yet, because each exact integral multiple value is unknown, the projected lengths and the reference vector itself both have errors. In the following, we first address this issue, and postpone the discussions: 1) unawareness of which references are qualified, 2) even no qualified references exist, afterwards.

Solution. We essentially view all the projected lengths as the constraints and then search for the most likely keyboard size x and y pair with a best match to these constraints.

For a reference vector, *e.g.,* $\vec{o}_1$ in Fig. 6(a), after all other vectors are projected to its own or perpendicular direction, we can form a matrix-like structure. Fig. 7(a) depicts the structure for $\vec{o}_1$'s own direction (y-axis), where columns correspond to all projected vectors, following a decreasing order based on their projected lengths, and each row represents one possible integral multiple value. The last row of "$.1y$" handles the case that a vector itself, *e.g.,* $\vec{o}_5$, is (nearly) perpendicular to the reference, with a very small projected length. Fig. 7(a) is for deriving the keyboard size y, and a similar structure can also be built for the projection of these vectors to $\vec{o}_1$'s perpendicular direction (x-axis) for deriving keyboard size x.

In Fig. 7(a), we adopt t_j^i to indicate the circle in row i and column j. Between any two adjacent columns, we can draw an edge, $e(t_j^i, t_{j+1}^{i'})$, to connect t_j^i and $t_{j+1}^{i'}$. In addition, we can further form a path from the first column to the last using connected edges, and each path indicates one allocation of the integral multiple values for each projected length. For instance, for the path in Fig. 7(a), "2y" is allocated to both $\vec{o}_3$ and $\vec{o}_4$. Therefore, the keyboard size y derived from $\vec{o}_3$ and $\vec{o}_4$ are

[2]In a real attack, we are not aware how large each vector's error is at this moment. So once a vector is selected as the qualified reference, we view it to be error-free, *e.g.,* along x- or y-axis. Inaccuracy from this approximation will be finally reflected from the quality of derived keyboard x and y values.

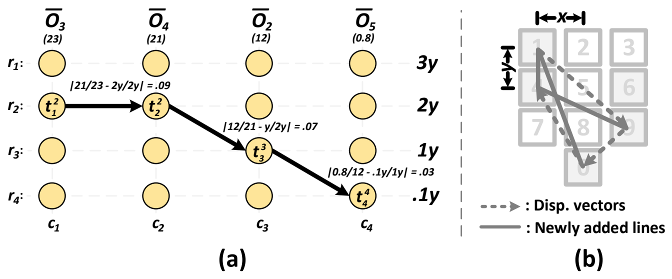

Fig. 7: **Keyboard size derivation design.** (**a**) Path search structure for Fig. 6(b). Edge weights are 0.09, 0.07 and 0.03, leading to 0.18 total path weight. (**b**) New line supplement.

$y_3 = \frac{23}{2} = 11.5$ and $y_4 = \frac{21}{2} = 10.5$, respectively, and the $\vec{o}_2$ leads to $y_2 = 12$. Therefore, the keyboard size y determined from this path is $y = \frac{y_3+y_4+y_2}{3} = 11.3$. Note that the vectors allocated with ".1y" are not used in deriving y as the factor "0.1" is an approximation merely, which is sufficient for the path selection (since it brings all paths the same inaccuracy) but not accurate enough to derive the value of y.

Since different paths lead to different allocations, our target is naturally to select the path achieving the best allocation. To quantify their differences, we define a weight for each edge:

$$w(t_j^i, t_{j+1}^{i'}) \quad = \quad |\frac{len(c_{j+1})}{len(c_j)} - \frac{len(r_{i'})}{len(r_i)}|, \tag{2}$$

where $len(c_j)$ is the projected length of the vector at column j and $len(r_i)$ is the represented allocation for row i.

A path weight can be further defined by adding the weights over all its edges. The path weight measures the **consistence** between the derived vectors' lengths and one allocation of the integral multiple values. A smaller weight indicates a higher likelihood of this allocation's (path's) correctness. Hence, we can use the dynamic programming to derive the best allocation.

With above design, for any reference vector $\vec{o}_r$, where $\vec{o}_r \in \{\vec{o}_l\}$, we can define a $pathSearch(\vec{o}_r)$ function as follows:

Step 1: Assume $\vec{o}_r$ along x-axis, *e.g.,* denoted as $\vec{o}_r(x)$, and project all other vectors to both this direction (x-axis) and its perpendicular direction (y-axis) for deriving the best keyboard values: x_1 and y_1 (with the minimal path weight for each).

Step 2: Repeat Step 1 by assuming $\vec{o}_r$ along the y-axis, *e.g.,* denoted as $\vec{o}_r(y)$, and obtain another pair: x_2 and y_2.

Step 3: Return (x_1, y_1) with $\vec{o}_r(x)$, and (x_2, y_2) with $\vec{o}_r(y)$.

Although one x and y pair must be less accurate or even wrong, we do not examine its correctness at this stage. Instead, we adopt both x and y pairs to infer the typed information and they are differentiated automatically by our design in §III-C.

2) Reference selection and generation: So far, we introduce how to derive keyboard size x and y from a qualified reference vector. However, in a recovered moving trajectory $\{\vec{o}_l\}$, we are not aware each vector is qualified or not at this moment. Therefore, we need to apply $pathSearch(\cdot)$ for each $\vec{o}_l$ vector. If some of them are indeed qualified, they will generate the most likely x and y pairs. Again, their correctness will be automatically differentiated after each x and y pair is applied for the typed information inference in §III-C.

On the other hand, it is also possible no qualified references exist in the moving trajectory, like Fig. 7(b). We find that in this case, if we connect the starting point of each vector to the end point of the next adjacent vector (forming a triangle), and also connect the starting point of the first vector to the end point of the last vector, qualified references will appear for sure in these newly added lines when at least 4 characters exist in the password. However, triggering this new line supplement mechanism will double the computation overhead. To wisely make the decision, the "keyboard size derivation" component works as follows:

1) We first apply the $pathSearch(\cdot)$ function for each $\vec{o}_l$ in the moving trajectory, and select the x and y pair with the *minimum* path weight, *e.g.,* the most likely allocation.

2) For either its x- or y-axis, if half of the x or y values, derived from the projected vectors at each row or column individually (*e.g.,* the three y values derived from $\vec{o}_3$, $\vec{o}_4$ and $\vec{o}_2$ in Fig. 7(a)), exhibit a large difference, *e.g.,* their mutual differences all $> 50\%$, the new line supplement is triggered.

3) All newly added lines are denoted as $\{\vec{n}_k\}$ and we apply the $pathSearch(\cdot)$ function for each $\vec{n}_k$ to generate more keyboard size x and y pairs.

Summary. This component outputs a series of x and y pairs and each pair's reference vector $\vec{o}_r(u)$ with direction u, where u indicates x- or y-axis.

C. Typed Information Inference

In this component, we match the moving trajectory $\{\vec{o}_l\}$ with each guessed keyboard layout size to infer user's typing.

1) Position of reference vector: Since the trajectory shape is determined by all the displacement vectors already and the orientation of the reference vector w.r.t. the keyboard is known, *e.g.,* $\vec{o}_1(y)$ is along the y-axis in Fig. 8(a), if we can further determine the right position of $\vec{o}_1(y)$'s starting point (*e.g.,* on which button), the entire trajectory can be correctly overlaid onto the keyboard. For instance, if we know the starting point of reference vector $\vec{o}_1(y)$ is on button "9" in Fig. 8(a), we can infer sequence $\mathcal{S}$ = "934056$''$ as the result, as it minimizes the average error $E_{\mathcal{S}}$ compared with other candidates:

$$E_{\mathcal{S}}(b) \quad = \quad \min_{\mathcal{S}}\{\frac{1}{L} \times \sum\nolimits_{i=1}^{L} e_i\}, \tag{3}$$

where b is the starting button position for the reference vector, L is the number of vectors, *e.g.,* $b = 9$ and $L = 5$ in Fig. 8(a), and e_i is the distance between the ending point of each vector $\vec{o}_i$ and the center of one button.

In fact, starting from any button b, we can always derive a sequence by minimizing Eqn. (3), which however tends to an incorrect result if b is wrongly selected. We of course could go through each possible b and then prioritize all the inferred results based on $E_{\mathcal{S}}(b)$s. This ambiguity however is expected to be alleviated in the first place; otherwise such an exhaust search will be performed for all keyboard size x and y pairs.

2) Ambiguity removal: We leverage the lengths from the projected vectors in the keyboard size derivation (in §III-B) as a hint to restrict b within a very limited range on keyboard.

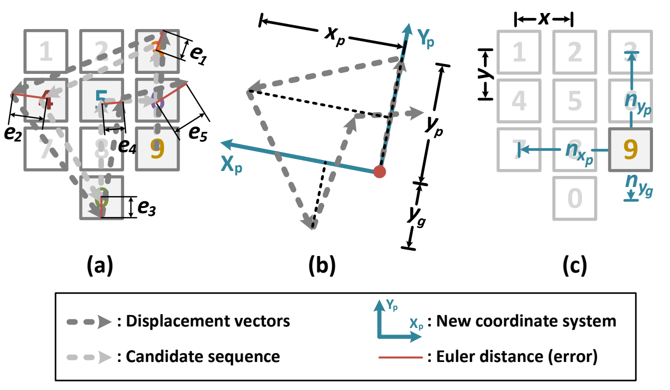

Fig. 8: **Typed information inference.** (**a**) Errors calculation. (**b**) Determining reference vector's position. (**c**) Final position.

For a reference vector, *e.g.,* $\vec{o}_1(y)$ in Fig. 8(a), we know its orientation w.r.t. the keyboard, which is along the y-axis. We propose to place $\vec{o}_1(y)$ along the same direction (y-axis) of another coordinate system and move its starting point to this coordinate system's origin, as Fig. 8(b) shows. We next project all the vectors, including the reference vector itself, to this y-axis, to first determine on which row the right b, denoted as $\tilde{b}$, should be. We write the largest positive (or zero) and smallest negative (or zero) projection values as y_p and y_g, respectively, and calculate:

- $n_p = round(y_p/y)$, indicating n_p rows are above $\tilde{b}$ on the keyboard, *e.g.,* $\tilde{b}$ is *at least* on the $(n_p+1)^{th}$ row.
- $n_g = round(-y_g/y)$, indicating n_g rows are below $\tilde{b}$ on the keyboard, *e.g.,* $\tilde{b}$ is *at most* on the $(n_g+1)^{th}$ last row.

For instance, in Fig. 8(c), $n_p = round(y_p/y) = 2$ and $n_g = round(-y_g/y) = 1$, which limits $\tilde{b}$ on the third row. Similar calculations are also performed for the x-axis, which further limits $\tilde{b}$ on the third column. Although $\tilde{b}$ sometimes cannot be uniquely identified, but it is already within a very limited range. In this case, we can utilize Eqn. (3) to prioritize the inferred results, *e.g.,* a smaller $E_{\mathcal{S}}(b)$ leads to a higher priority.

On the other hand, if the inferred sequence contains numbers from 1 to 9 only, *e.g.,* Fig. 9(a-up-left), another sequence, by rotating the keyboard 180°, has the same $E_{\mathcal{S}}(b)$ value, *e.g.,* Fig. 9(a-up-right). In addition, if $x = y$, two more sequences in Fig. 9(a-down) also have same $E_{\mathcal{S}}(b)$ values. Therefore, we need to further remove such ambiguity. We observe that when the user is typing, wearable's coordinate system (after rotating 90°) has a similar orientation as keypad's as in Fig. 9(b). Our key idea is to use the four sequence candidates in Fig. 9(a) to generate their own moving trajectories in the same coordinate system as the one recovered from the wearable. We can then conduct inner-production to identify the most likely one. Moreover, *aLeak* can also handle cases that there is no "enter" button on the typed keyboard (details are omitted).

IV. Performance Evaluation

Experiment setup. We develop a prototype system of *aLeak* based on previous designs and examine its performance in this

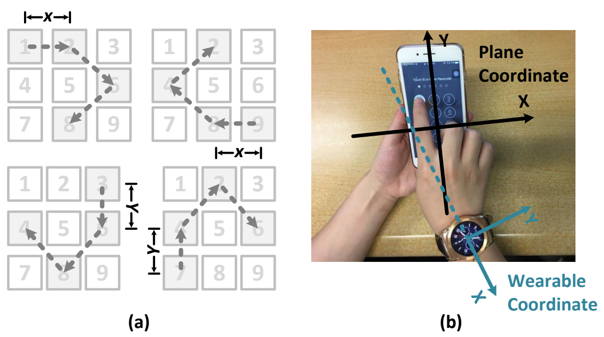

Fig. 9: **Ambiguity removal.** (**a**) Ambiguous results with same $E_{\mathcal{S}}(b)$ error. (**b**) Using coordinates' similarity to differentiate.

section. We experiment with 5 users wearing LG W150 smart watch and act as the adversary to attack more than 300 rounds of users' password inputs on four common types of keyboards, including: 1) an POS terminal, 2) an entrance guard panel, 3) a telephone dial pad and 4) the virtual keyboard on iOS. The keyboard size varies from $14mm$ to $21mm$ (for x) and $10mm$ to $21mm$ (for y), and their posture angle changes from 0° to 90° in the experiment. We also vary the sampling rates of the 3-axis accelerometer and gyroscope data from 30Hz to 200Hz.

To validate the efficacy of our design, we compare *aLeak* with the state-of-the-art approach RCCS [13]. The attack in RCCS assumes a *horizontal* keypad plane and it also requires 1) the *known* keyboard size and 2) the last keystroke on a *fixed* "enter" button to enable the attack. We explicitly provide such two pieces of information for RCCS, while *aLeak* is not aware of any context information. Some keyboards used in the experiment have no "enter" button. In this case, we provide the ground truth of the last keystroke as "enter" for RCCS.

Successful rate. After attacking each piece of user-typed passwords, *aLeak* and RCCS both can provide a set of candidate inferences ordered by the likelihood. In Fig. 10, we examine their successful rates from top-1 (*i.e.,* the candidate with the highest likelihood is just the user-typed password) to top-5 candidates (*i.e.,* the password is in the first five candidates). In this experiment, keyboards' posture includes all possible angles from 0° to 90° with a step size of 10°. For each posture angle, we attack a similar amount of users' typings.

From the result, we can see that even top-5 successful rate of RCCS is 15% merely, which is mainly achieved when keypad plane is close to be horizontal (as it assumes). In contrast, *aLeak* can achieve 94% top-5 and even 45% top-1 successful rates without any context information. The improvements are gained from 4.8x to 5.3x. Fig. 10 essentially demonstrates the severity of user's typing leakage risk unveiled in *aLeak*, since when a user types on many mobile platforms in practice, *e.g.,* mobile phone or POS terminal, even the context-aware assumption does not hold usually, *e.g.,* with an arbitrary unknown posture angle, *aLeak* shows that user's typing can still be possibly leaked through wearable's motion sensors. This should draw our great attention for the typing on mobiles.

Recovered context information. As *aLeak* does not assume

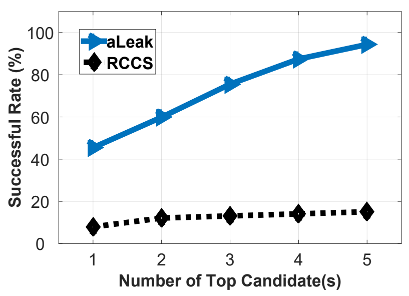

Fig. 10: Successful rates from top-1 to top-5 candidates of *aLeak* and RCCS.

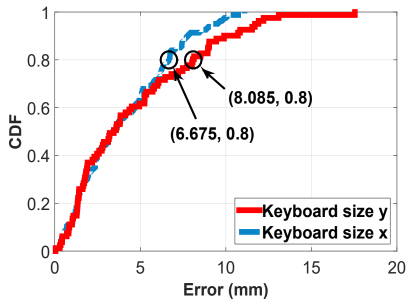

Fig. 11: CDF of the keyboard size errors derived from *aLeak*.

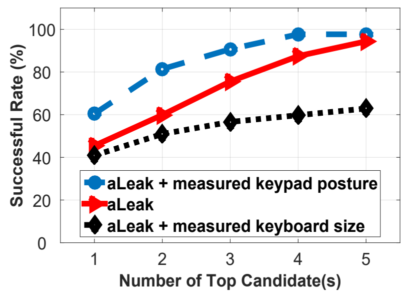

Fig. 12: Impact of keypad posture and size recovery to *aLeak*'s performance.

context-related information, its performance highly depends on the accuracy of the recovered keypad postures and keyboard sizes. In Fig. 4 (on p. 4), we have demonstrated that *aLeak* can achieve a good keypad posture recovery, *e.g.,* the average error is less than 7°. In this section, Fig. 11 further indicates that the accuracy of the derived keyboard sizes by *aLeak* is accurate as well. Compared with the measured ground truth, the average keyboard size derivation error is 4.0mm for x (4.6mm for y) and the 80th percentile error is 6.7mm for x (8.1mm for y). As the keypad posture error has a more significant impact on the y-axis, the error of y is slightly larger than x in Fig. 11. In summary, experiments show the good performance achieved from both of these two aspects, which ensures *aLeak*'s high successful rates in Fig. 10.

In Fig. 12, we further investigate *aLeak*'s performance loss due to the keypad posture and keyboard size recovery errors. We first repeat all previous attacks by providing the ground truth of the keypad posture (but the keyboard size is still derived by *aLeak*). The result shows for the top-1 accuracy, the keypad posture error leads to 15% performance loss, while its impact on the top-5 rate becomes minimal. This is another indication that *aLeak*'s keypad posture recovery is accurate.

We then repeat the experiment again with the ground truth of keyboard size only. We notice that the performance even degrades. Through our investigation, we find that this is because wearable's moving trajectory is slightly different from finger's moving. Thus, the derived result by *aLeak* is essentially the desired (and effective) size that matches wearable's movement.

Different keypad posture angles. In Fig. 10, we have shown *aLeak*'s overall performance. In Fig. 13, we further provide its detailed performance under different keypad posture angles. For the ease of illustration, we group these angles into three ranges: $0° \sim 20°$, $30° \sim 60°$, $70° \sim 90°$, and depict the performance for each range. From the result, we observe that when the posture angle is closer to 0°, the performance tends to be better, *e.g.,* 57% vs. 30% for the top-1 accuracy in the first and third ranges, respectively. However, the performance of *aLeak* in general is comparable in the three ranges, *e.g.,* successful rate differences between the first and third ranges reduce to 13.6% and 0.3% for the top-3 and top-5, respectively.

Different keypads. In Fig. 14, we plot *aLeak*'s performance achieved on four different (three physical and one virtual) keypads, where the keyboard size x and y varies from 14mm to 21mm and 10mm to 21mm, respectively. Overall, *aLeak* achieves similar performance cross different keypads. The top-1 successful rate varies from 36% to 57%, and there is no obvious difference for the successful rate from the top-5 candidates. The reason that the top-1 performance on the second keypad is slightly worse because this keypad is relatively aged, such that the buttons become soft and not very responsive. Hence, the signal-to-noise ratio of the accelerations observed from this keypad is relatively lower than the other three. Nevertheless, its (absolute) successful rates are still high.

Different motion data sampling rates. Although wearable device could sample motion data at a high rate, *e.g.,* 200Hz, the effective sampling rate achieved by the adversary might be much lower [13]. From Fig. 15(a), we observe that this side-channel attack is robust to the sampling rate reduction. Even the rate is reduced to 30Hz (by down-sampling), the achieved successful rates are comparable to the high sampling rates, *e.g.,* 200Hz and 50Hz, for all numbers of top candidates. Fig. 15(a) essentially implies that the barrier to launch this attack is trivial, consistent with the observations from prior studies [8], [13]. In this experiment, we have tried to further lower the sampling rate and found that after the rate is lower than 10Hz, the system cannot provide meaningful outputs.

Different users. In Fig. 15(b), we further plot the performance achieved on different users. From the result, we can see that their top-1 successful rates are slightly scattered around 50%, while their top-5 successful rates are all very high. Fig. 15(b) indicates that *aLeak* can achieve an effective wearable side-channel attack on different users.

V. Related Work

Privacy leakage by wearable motion sensors. Some existing efforts [13], [14], [8] demonstrate the initial success of the keystroke leakage through wearable's motion sensors. MoLe [14] leverages a linguistic model to infer the English word typing on computers keyboards. Liu *et al.* [8] report the password leakage on POS terminals and cope with the inaccurate motion data using a machine-learning based approach, which however requires the training and known keypad plane in prior. Wang *et al.* [13] further release the training requirement and propose a backward inference method to migrate the

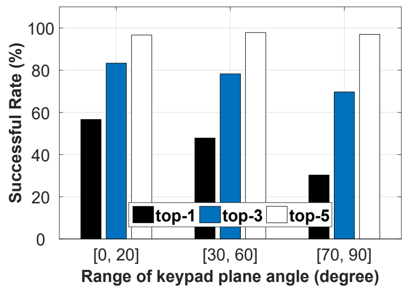

Fig. 13: Performance of *aLeak* under different keypad postures, where posture angles divided into three ranges.

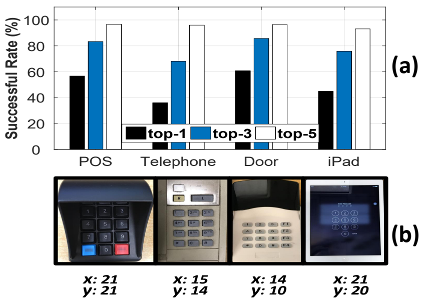

Fig. 14: Performance on different keypads. (**a**) Successful rates in *aLeak*. (**b**) Keypads used in the experiment.

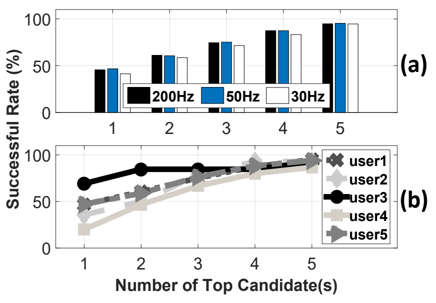

Fig. 15: Successful rates in *aLeak* (**a**) under different sampling rates and (**b**) with different users.

motion data inaccuracy. These recent successes however are achieved under certain known contexts about user's typing: 1) the horizontal keypad plane, 2) the known keyboard sizes, *e.g.,* the ATM or POS panels, 3) and/or the last keystroke on a fixed "enter" button. In this paper, we take one step further by addressing unsolved challenges to demonstrate the possibility that leaks user's typing privacy in much more general context-free scenarios, so as to unveil (more importantly alarm people) the further privacy leakage risks that are not viable before.

Privacy leakage by other side-channels. Some other side-channel attacks to user's typing privacy are also studied in the literature. In particular, the user's typing on mobile platforms can be compromised by ambient cameras, where Wu *et al.* [15] study such a camera-based attack on mobile phones and Ye *et al.* [18] report to crack the Android pattern lock in five attempts. In addition, recent studies find that the user's typing privacy can be leaked by the wireless mouse trajectory [10] as well. On the other hand, the user's typing on mobile devices can also be compromised by mobile's own sensors, *e.g.,* the keystrokes on touch screen can be inferred from motion sensors [3] and gyroscope can be used to unveil the fingerprints of user's typing [9]. These existing works are essentially orthogonal to this paper, which do not address the unique challenges in the *aLeak* design.

VI. Conclusion

This paper presents *aLeak*, which fully demonstrates, more importantly alarms people, a crucial typing privacy leakage risk in much more generalized context-free scenes that are not viable before. We address the inaccurate motion recovery, unknown keyboard size and inference ambiguity three unsolved challenges in the *aLeak* design and develop a prototype system to validate its feasibility. Extensive experiments by attacking more than 300 rounds of different users' typings without the context information indicate the efficacy of *aLeak*.

Acknowledgment

This work is partially supported by the GRF grant from Research Grants Council of Hong Kong (Project No. CityU 11217817), and the ECS grant from Research Grants Council of Hong Kong (Project No. CityU 21203516).

References

[1] Material design. https://material.io/devices/.

[2] Wired keyboard sale. http://cn.made-in-china.com/Computer-Products-Catalog/Keyboard.html.

[3] L. Cai and H. Chen. Touchlogger: Inferring keystrokes on touch screen from smartphone motion. *Proc. of USENIX HotSec*, 2011.

[4] X. Guo, J. Liu, and Y. Chen. Fitcoach: Virtual fitness coach empowered by wearable mobile devices. In *Proc. of IEEE INFOCOM*, 2017.

[5] J. Han, C. Qian, P. Yang, D. Ma, Z. Jiang, W. Xi, and J. Zhao. Geneprint: Generic and accurate physical-layer identification for uhf rfid tags. *IEEE/ACM Transactions on Networking*, 2016.

[6] C. Karatas, L. Liu, H. Li, J. Liu, Y. Wang, S. Tan, J. Yang, Y. Chen, M. Gruteser, and R. Martin. Leveraging wearables for steering and driver tracking. In *Proc. of IEEE INFOCOM*, 2016.

[7] Z. Li, M. Li, P. Mohapatra, J. Han, and S. Chen. iType: Using eye gaze to enhance typing privacy. In *Proc. of IEEE INFOCOM*, 2017.

[8] X. Liu, Z. Zhou, W. Diao, Z. Li, and K. Zhang. When good becomes evil: Keystroke inference with smartwatch. In *Proc. of ACM CCS*, 2015.

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

[10] X. Pan, Z. Ling, A. Pingley, W. Yu, N. Zhang, K. Ren, and X. Fu. Password extraction via reconstructed wireless mouse trajectory. *IEEE Transactions on Dependable and Secure Computing*, 2016.

[11] M. Ryan et al. Bluetooth: With low energy comes low security. *WOOT*, 2013.

[12] D. Spill and A. Bittau. Bluesniff: Eve meets alice and bluetooth. *WOOT*, 2007.

[13] C. Wang, X. Guo, Y. Wang, Y. Chen, and B. Liu. Friend or foe?: Your wearable devices reveal your personal pin. In *Proc. of ACM ASIACCS*, 2016.

[14] H. Wang, T. T.-T. Lai, and R. Roy Choudhury. Mole: Motion leaks through smartwatch sensors. In *Proc. of ACM MobiCom*, 2015.

[15] L. Wu, X. Du, and X. Fu. Security threats to mobile multimedia applications: Camera-based attacks on mobile phones. *IEEE Communications Magazine*, 2014.

[16] L. Xie, X. Dong, W. Wang, and D. Huang. Meta-activity recognition: A wearable approach for logic cognition-based activity sensing. In *Proc. of IEEE INFOCOM*, 2017.

[17] P. Xie, J. Feng, Z. Cao, and J. Wang. Genewave: Fast authentication and key agreement on commodity mobile devices. In *Proc. of IEEE ICNP*, 2017.

[18] G. Ye, Z. Tang, D. Fang, X. Chen, K. I. Kim, B. Taylor, and Z. Wang. Cracking android pattern lock in five attempts. In *Proc. of ISOC NDSS*, 2017.

[19] L. Zhang, X.-Y. Li, K. Liu, T. Jung, and Y. Liu. Message in a sealed bottle: Privacy preserving friending in mobile social networks. *IEEE Transactions on Mobile Computing*, 2015.

[20] X. Zheng, J. Wang, L. Shangguan, Z. Zhou, and Y. Liu. Design and implementation of a csi-based ubiquitous smoking detection system. *IEEE/ACM Transactions on Networking*, 2017.

[21] P. Zhou, Y. Zheng, and M. Li. How long to wait?: predicting bus arrival time with mobile phone based participatory sensing. In *Proc. of ACM MobiSys*, 2012.