aLeak: Context-Free Side-Channel from Your Smart Watch Leaks Your Typing Privacy

Yang Liu, *Student Member, IEEE* and Zhenjiang Li, *Member, IEEE*

Abstract—We revisit a crucial privacy problem in this paper — can the sensitive information, like the numeric 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, and/or 3) 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 500 rounds of different users' typings on various mobile platforms without any context-related information.

Index Terms—Privacy leakage, wearable device, wearable sensing, side-channel

1 Introduction

Rich sensors on electronic devices, e.g., wearables, could generate valuable sensory data [27], [34], for designing a corpus of useful applications, e.g., activity recognition [29], driver tracking [12], fitness [8], authentication [9], [30], social networks [32], localization [23], etc. We have thus nowadays witnessed their increasing popularity and high penetration into our daily life. However, studies, like [14], [26], [27], 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 [13], e.g., numeric passwords, personal particulars, etc., on various mobile platforms, e.g., 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 the user's wrist?* If so, the consequence is serious, as the barrier to launch this side-channel attack is trivial [14], [26].

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 technical challenges.

• *The authors are with the Department of Computer Science, City University of Hong Kong, Hong Kong, China.*
E-mail: yliu562-c@my.cityu.edu.hk, zhenjiang.li@cityu.edu.hk.

Manuscript received 28 May 2018; revised 16 Apr. 2019; accepted 13 May 2019. Date of publication 20 May 2019; date of current version 1 July 2020. (Corresponding author: Zhenjiang Li.)
Digital Object Identifier no. 10.1109/TMC.2019.2917659

Challenges. To launch this side-channel attack, attackers need to precisely recover each piece of wearable's moving displacement *vectors*, which capture user's finger transitions between two keystrokes, as in Fig. 1a. 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. 1b-left. During this process, the following three challenges will be encountered.

1) *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 amplify the acceleration errors. Recent remedy methods, e.g., the mean-removal [27], 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 Section 2.2). Hence, the keypad plane (with an arbitrary posture) cannot be reliably reconstructed in the first place, and the erroneous vectors cannot truthfully reflect wearable's moving trajectory neither — it is non-trivial to launch above side-channel attack.
2) *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. 1b, is unknown, lacking the

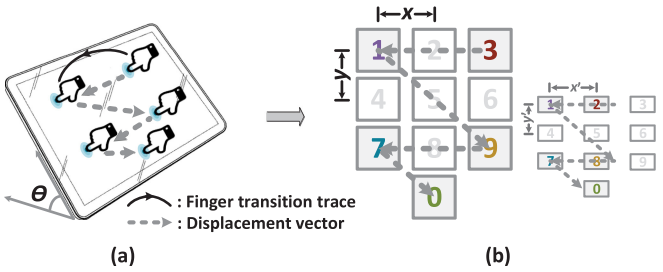

Fig. 1. Illustration of wearable side-channel attack. (a) Recover all displacement vectors and keypad plane posture angle θ. (b) Match the moving trajectory on the keyboard with correct (left) and incorrect (right) keyboard sizes, incurring different results: "31970" (correct) yet "21870" (incorrect).

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 keyboard size), which however tends to be wrong, e.g., the wrong result "21870" in Fig. 1b-right when an incorrect keyboard is used. Therefore, without this keyboard meta information, the attack stagnates again.

3) *Information inference*. After the keyboard size, e.g., x and y values in Fig. 1b, could be eventually figured out, wearable's displacement trajectory may still be embedded into the keyboard layout in different ways, leading to different inference results of the typed information. Such an ambiguity needs to be effectively removed. On the other hand, if the user's typed information is a long sequence, it could take an exponential computation complexity to derive the most likely results as the potential candidate amount can potentially increase due to motion data processing errors and inference ambiguity. Although we could assume sufficient computing resources from the attackers, a tractable attack (if possible) is definitely more severe in practice.

To overcome above issues, existing works [14], [26] 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., numeric 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 [26]. Therefore, by accessing the data from wearable motion sensors, e.g., accelerometers and gyroscope, (attack model is detailed in Section 2.1), adversary could launch the wearable side-channel attack.

We have conducted experiments to analyze the wearable sensory data and proposed a series of key techniques to address aforementioned challenges in *aLeak*. To demonstrate the efficacy of *aLeak*, we experiment with 5 users wearing the smart watch and act as the adversary to attack more than 500 rounds of users' password inputs on various platforms, e.g., mobile devices, door entrance panels, telephones and POS, with the keypad posture angles changing from 0 to 90 degree. Experiments show that without any context information, *aLeak*'s top-1 successful attacking rate is 45 percent and the top-5 accuracy increases to 94 percent, while the top-5 accuracy of the most recent RCCS [26] is 15 percent merely. Moreover, the average delay of *aLeak* is relatively stable under different password lengths, e.g., $64s$ to $89s$. For RCCS, short passwords can be inferred more efficiently (e.g., within $5s$ for 4 to 8 characters), but the delay grows rapidly for longer passwords (e.g., $58s$ and $1005s$ for 9 and 10 characters respectively).

In summary, this paper makes following contributions. 1) We demonstrate the possibility to leak the user's typing privacy in a context-free manner and unveil (more importantly alarm people) the further privacy leakage risks that may not be viable before. 2) We propose a set of key techniques to address, inaccurate motion recovery, unknown keyboard size and information inference, three major challenges to enable the side-channel attack in *aLeak*. 3) We develop a prototype system and conduct extensive experiments with 5 volunteers, by attacking their more than 500 rounds of inputs on a variety of mobile platforms with different keyboard sizes and keypad postures.

Roadmap. In the rest of the paper, we introduce the attack model and preliminary in Section 2, and elaborate the *aLeak* design in Section 3. The evaluation is in Section 4. Related works are reviewed in Section 5 before the discussion in Section 6 and the conclusion in Section 7.

2 Attack Model and Preliminary

Although biometric sensors, e.g., Touch ID [4], 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 [13], e.g., 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) numeric 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 Section 1 can be addressed, user's typing privacy will easily get compromised and scarified (which are not viable before). In this

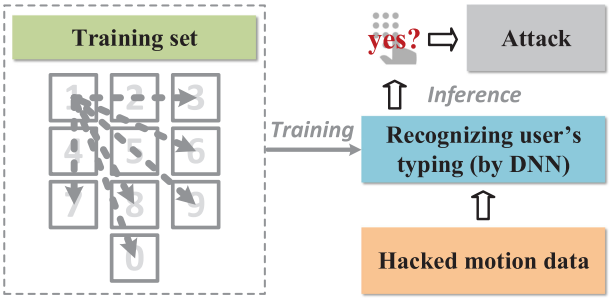

Fig. 2. Procedure to identify the typing piece within the motion data by a DNN-based design.

section, we will detail the attack model (Section 2.1) and challenges (Section 2.2).

2.1 Attack Model

Design Assumptions. We first introduce the assumptions in the current *aLeak* design as follows.

User's Typing Style. Similar as recent studies [14], [26], [27], the *aLeak* design assumes that the smart watch moves along with the user's finger as well, so that the smart watch's accelerometer data could reflect the finger's movement when the user is typing. If the user only moves his/her hand in the typing (wrist is not moving) or the user types using the thumb merely, the recovered displacement vectors may not precisely describe the finger's movement in these cases.

Advanced Keyboard Layout. Some advanced keyboards could randomize their buttons' layout to enhance the user's typing privacy, which is further discussed as one effective defense mechanism in Section 6. The current *aLeak* design mainly focuses on ordinal keyboards without the randomization feature, e.g., the widely used keyboards of physical buttons.

Content of Passwords. For some complex passwords, like the mix of digits, letters and special characters, the user may need to type cross different types of keyboards. Such an sophisticated setting is not considered in this paper and *aLeak* mainly focuses on the number input, typed on various numeric keyboards on the mobile device, telephone, ATM, POS, door entrance device, etc.

Hacking Motion Data from Devices. Similar as the existing studies [14], [26], [27], the adversary could (not must) hack the same set of motion sensor data (accelerometers and gyroscope) from the wearable device on the user's wrist to launch the side-channel attack through two possible ways:

Installing Third-Party Applications. The adversary can trick a user (e.g., victim) to install a malicious application on the smart watch without victim's notice to leak the motion sensor data [14]. However, with an anti-virus application, this approach may become more challenging and less effective. Recently, the studies find that such a third-party application can be further designed as a legitimate application, such as games [16], [26]. The motion sensor data can be first used by these applications legally to fulfill their functions, while the permitted access of the motion sensors [1] also provides the application designer (e.g., the designer could be an adversary) an opportunity to collect the motion sensor data for malicious purposes [16], [20]. The third-part application can leak the motion data when wireless [11] is available.

Sniffing Blue-Tooth Packets. A wearable device is usually paired with the victim's mobile phone through blue-tooth and the wearable needs to constantly report its sensor data to the phone for 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 [22], [25], [26] to recover the motion data. For instance, the adversary stealthily places a wireless sniffer close to the mobile platforms, e.g., ATM, POSs, door entrance panels, and captures the blue-tooth packets transmitted from the wearable on victim's wrist to his/her phone to extract the motion sensor data.

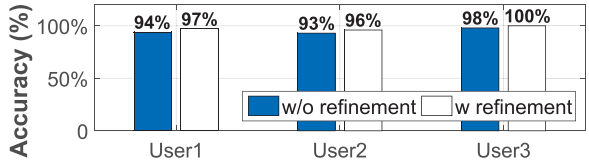

Fig. 3. Accuracy to identify the data sequence corresponding to the typing for three different users.

Identifying the Data Piece for Typing. After collecting the user's motion data, prior studies [14], [26], [27] all assume that the attacker can identify the piece within the data stream that corresponds to the user's typing. In this section, we investigate this assumption with a deep neural network (DNN) design and find that this attack model assumption is indeed valid, and the procedure is depicted in Fig. 2.

During a user's typing, the basic motion data unit is the transition between two consecutive button typings, e.g., Fig. 2 depicts all the transitions starting from button "1". Therefore, in the training phase, the attacker can first use his/her own motion data during the typing to train a DNN, where the detailed network structure with the Long Short-Term Memory (LSTM) [10], [19] is introduced in Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TMC.2019.2917659. In the experiment, we collect more than 600 such data units from one person acting as an attacker, covering all possible transitions on the keyboard under various keypad postures and typing behaviors, as well as the motion data from many other daily activities. After training, we apply the trained DNN to identify the typing data piece for other different users,[1] whose data are not used in the training.

In particular, with the hacked motion sensor data from one user, the attacker can divide the data stream by a time window, e.g., 1s. The configuration of the time window is investigated in Appendix, available online, and Fig. 3 shows the accuracy for using the DNN to identify the typing sequence on three new users (whose data are not used in the DNN training). In particular, we define a successful identification as: 1) the true typing sequence is included in the identified one, and 2) the identified sequence's boundary (beginning or end) should be no longer than 50 samples, e.g., around 1s, compared with the true sequence. Fig. 3 shows that the identification accuracy ("w/o refinement") is from 93 to 98 percent. Moreover, after the user's password is hacked, the attacker can utilize the "decoded" information as labels and the corresponding motion data to refine (e.g., retrain) the DNN for

1. As we assume that the smart watch moves along with the user's finger (Section 2.1), the motion data units, corresponding to the same finger's transition from different users, could share many similarities, providing the opportunity that this DNN can be used for other different users.

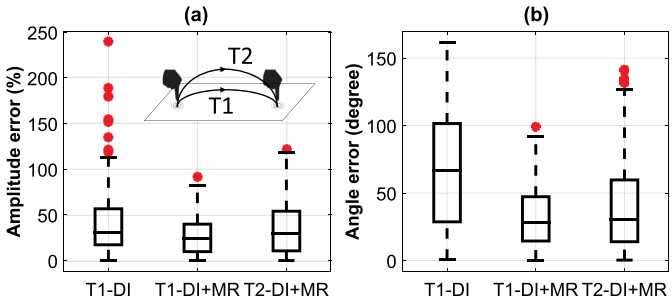

Fig. 4. Accuracy of derived vectors. (a) Amplitude and (b) angle errors, where "DI" and "MR" stand for double-integral and mean-removal, respectively, and the maximum heights of the two traces T1 and T2 are about $1cm$ and $3cm$.

further improving the performance, to hack user's other passwords. With this refinement, the attacker could maintain different DNNs for different users, and the accuracy ("w refinement") is improved from 96 to 100 percent in Fig. 3 for three users.

After identifying the motion data of user's typing, the adversary then can infer his/her typed information. The same as the attack model adopted in existing studies [14], [26], [27], we also focus on the scenarios that user's finger and the wearable do not have significant mutual movements during the typing, e.g., wearable's motions could represent finger's motions during this process, which widely occur for the typing on the mobile phone, ATM and door entrance panel. Even under the attack model above, the adversary still needs to further cope with the following challenges to enable the context-free side-channel attack in practice.

2.2 Attack Challenges

2.2.1 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 percent and 19 degree, respectively (the derivation detail is omitted here). Of course, errors could further accumulate cross the vectors in a moving trajectory. The accuracy is thus expected to be even higher.

We conduct experiments to examine this feasibility. Fig. 4 shows that the direct double-integral (T1-DI) is highly inaccurate. The percentage of its amplitude error is 30.3 percent on average and can be up to 112.8 percent (w.o. outliers). Meanwhile, the angle error is 67.0 degree on average and can be up to 160 degree. In Fig. 4, we further apply the mean-removal technique [3] to improve the accuracy, where the amplitude and angle errors (T1-DI+MR) can be reduced to 23.8 percent and 28.3 degree, 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. 4a in the typing, e.g., about 27 percent amplitude and 29 degree 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.

TABLE 1
The Feasible x and y Ranges Between Two Adjacent Buttons Cross Different Mobile Platforms (Unit: mm)

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	$15 \sim 28$	$30 \sim 97$

2.2.2 Variant Keyboard Sizes

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

3 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 Section 3.1, 2) keyboard size derivation in Section 3.2, and 3) typed information inference in Section 3.3.

3.1 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 Section 3.1.1). 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 Section 3.1.2).

3.1.1 Motion Data Segmentation

Acceleration data[2] $\{a_i\}$ 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.

2. 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 [1] 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.

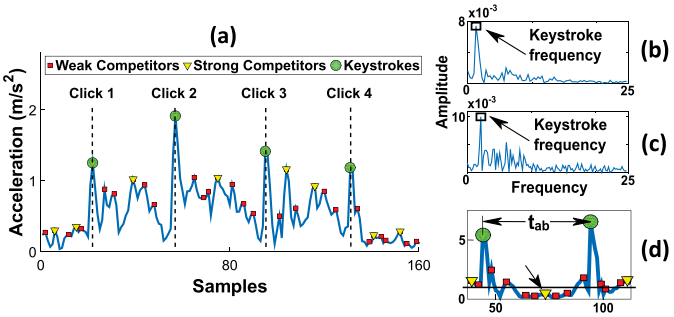

Fig. 5. Motion data segmentation. (a) Instrumental example. (b) FFT on $\{a_i\}$ to extract user's typing pace. (c) Typing pace extraction from 5 finger transitions with 2 being prolonged. (d) Accelerations are small when user's finger hangs in air.

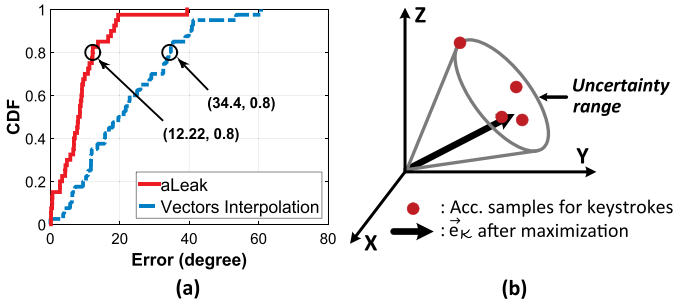

Fig. 6. Keypad plane reconstruction. (a) CDF of keypad plane reconstruction errors. (b) Estimation of $\vec{e}_{\mathcal{K}}$ direction.

However, due to high dynamic acceleration varyings and inevitable jitters, we find that how to first *automatically* and *reliably* identify these delimiters becomes not so trivial.

Keystrokes could incur prominent acceleration changes [26], e.g., $\{||\vec{a}_i||_2\}$, as the (green) circles marked in Fig. 5a. 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. 5a. 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. 5b depicts, which corresponds to the average time interval t_{pace} between two consecutive keystrokes, e.g., the typing pace. As a result, we have

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

From Fig. 5a, we can observe 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. 5a, and we denote the remaining peaks as a set $\{p_j\}$, which contains both the accelerations for keystrokes, e.g., the circles in Fig. 5a, and the strong "competitors", e.g., the triangles in Fig. 5a. 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. 5c 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. 5d. 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 percent of the largest peak) in $\{p_j\}$ first before using Algorithm 1.

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 < α · t_type then
7             remove the item with a smaller amplitude;
8 return {s_j} ← {p_j};
```

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 $\vec{v}_l$, in the same absolute coordinate system as $\{s_j\}$. As the mean-removal may not be able to fully remove sensor data errors, we further propose a design to find the keypad plane without accumulated errors by integral operations in Section 3.1.2.

3.1.2 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. 1a, such that wearable's moving trajectory along the keypad plane can be recovered.

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. 4), the plane reconstruction performance is not satisfactory, e.g., the 80th percentile error is 34.4 degree and it can be up to 60.3 degree as depicted in Fig. 6a.

To migrate the inaccuracy from $\vec{v}_l$, we observe that 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 $\{s_j\}$

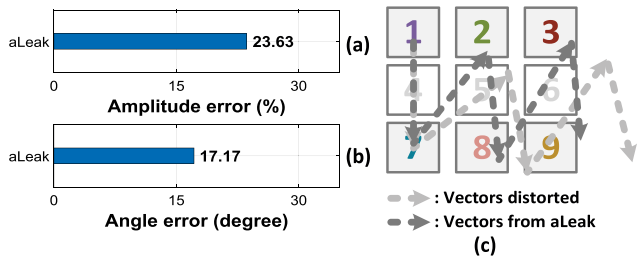

Fig. 7. Wearable moving trajectory recovery. (a) Amplitude and (b) angle errors of vectors in $\{\vec{o}_l\}$. (c) Recovered moving trajectories distorted by various vector angle errors.

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. 6b 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 $\vec{e}_{\mathcal{K}}$

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

where $g(s_j, \vec{e}_{\mathcal{K}})$ represents to project s_j to the $\vec{e}_{\mathcal{K}}$ direction. In Fig. 6a, we see that the 80th percentile errors of this design can be reduced to 12.2 degree.

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 Figs. 7a and 7b, we examine the accuracy of derived $\{\vec{o}_l\}$. The result shows both the amplitude and angle accuracies are improved, especially for the angular performance, e.g., 17 degree 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. 7c 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 user's typed keypad plane $\mathcal{K}$.

3.2 Keyboard Size Derivation

This component derives the unknown keyboard size, which is represented by the x and y values as depicted in Fig. 8a. 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.

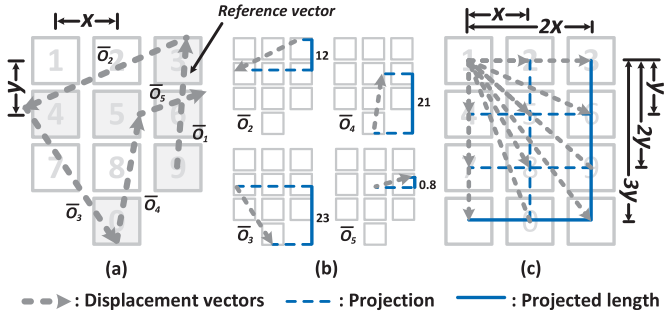

Fig. 8. Reference vector in keyboard size derivation. (a) $\vec{o}_1$ is selected as reference vector. (b) Other vectors are projected to $\vec{o}_1$'s own direction. (c) Possible integral multiple values.

3.2.1 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. 8a, 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 y and x values, respectively. Fig. 8c shows all possible integral multiples in principle when one vector is projected to these two directions.

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. 8b shows the projected lengths of all other vectors to the qualified reference $\vec{o}_1$'s own direction (y-axis).[3] 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. 8a, after all other vectors are projected to its own or perpendicular direction, we can form a matrix-like structure. Fig. 9a 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 maximum integral multiples are 2 and 3 for x and y directions respectively as Fig. 8c shows). 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. 9a is for deriving the keyboard size y, and a similar structure can also be built for the

3. 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.

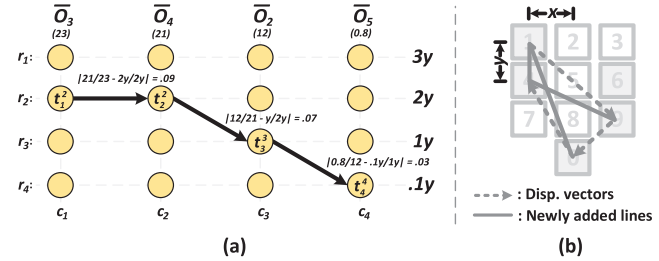

Fig. 9. Keyboard size derivation design. (a) Path search structure for Fig. 8b. Edge weights are 0.09, 0.07, and 0.03, leading to 0.19 total path weight. (b) New line supplement.

projection of these vectors to $\vec{o}_1$'s perpendicular direction (x-axis) for deriving the value of x.

In Fig. 9a, we adopt t^i_j to indicate the circle in row i and column j. Between any two adjacent columns, we can draw an edge, $e(t^i_j, t^{i'}_{j+1})$, to connect t^i_j and $t^{i'}_{j+1}$. 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. 9a, "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 $y_3 = \frac{23}{2} = 11.5$ 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 keyboard size y.

Because 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 as follows:

$$w(t^i_j, t^{i'}_{j+1}) = |\frac{len(c_{j+1})}{len(c_j)} - \frac{len(r_{i'})}{len(r_i)}|, \quad (2)$$

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. Therefore, 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 that works as follows with 3 steps:

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 axis).

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 the derived (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 from step 3 above, 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 Section 3.3.

3.2.2 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 Section 3.3.

On the other hand, it is also possible that no qualified references exist in the moving trajectory, like Fig. 9b. To address this issue, we propose to further 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. Then by enumerating all the combinations, we observe that when the number of distinct characters is small, e.g., three, the qualified reference vector may not be covered by the original moving trajectory and these newly added lines, wherein the coverage ratio is 93.4 percent. However, when the distinct character number is equal to or greater than four,[4] the coverage ratio achieves 100 percent. 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. 9a), 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 the direction u, where u indicates x- or y-axis.

3.3 Typed Information Inference

This component finally matches moving trajectory $\{\vec{o}_l\}$ with each guessed keyboard layout size to infer user's typing.

3.3.1 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. 10a, 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

4. Simple passwords with identical numbers, like "1111", can challenge the keyboard size information derivation, but such a password itself is not safe enough. We thus do not suggest to use them to defend this attack. Potential defense mechanisms are discussed in Section 6.

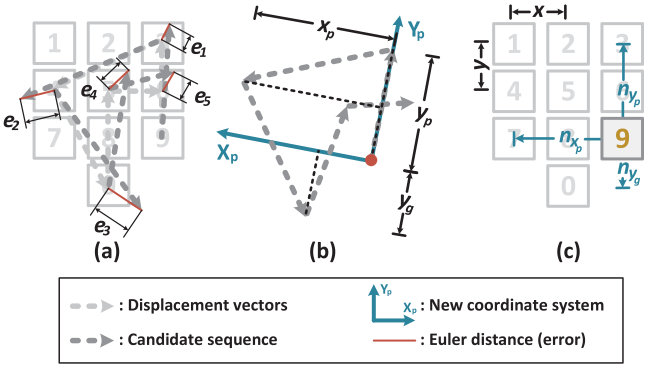

Fig. 10. Typed information inference. (a) Errors calculation. (b) Finding reference vector's position. (c) Final position.

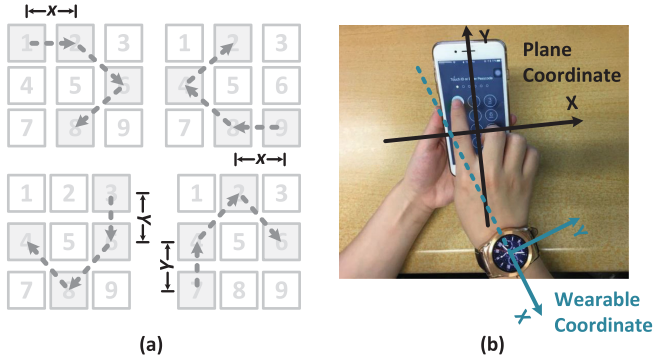

Fig. 11. (a) Ambiguous results with the same $E_{\mathcal{S}}(b)$ error. (b) Using the coordinates' similarity to differentiate.

button "9" in Fig. 10a, we can infer sequence $\mathcal{S} = "934056"$ as the result, as it minimizes the average error $E_{\mathcal{S}}$ compared with other candidates

$$E_{\mathcal{S}}(b) = \min_{\mathcal{S}} \left\{ \frac{1}{L} \times \sum_{i=1}^{L} e_i \right\}, \tag{3}$$

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. 10a, and e_i is the distance between the ending point of each vector $\vec{o}_i$ and the center of one button. To improve the "decoding" performance in Eqn. (3), we leverage the method proposed in [26] with two useful techniques. First, for each $\vec{o}_l$, e_i records the distance between its end point and the center of the closest button. To continue to decode the next vector $\vec{o}_{l+1}$, the end point of vector $\vec{o}_l$ will be enforced to be placed at the center of its closest button, and we can then calculate e_{i+1} for the next vector $\vec{o}_{l+1}$. This will not accumulate the error contained in each vector [26]. Second, for each vector $\vec{o}_l$, we can calculate the error for multiple nearby buttons (we calculate for the top-two closest buttons in *aLeak*) on the keyboard and finally output the button sequence, which minimizes Eqn. (3), as the result, because in case that the closest button to one vector turns out to be an incorrect answer (the error is relatively large in this vector), the decoding for all the following vectors could be wrong.

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 in the information inference.

3.3.2 Ambiguity Removal

We leverage the lengths from the projected vectors in the keyboard size derivation (in Section 3.2) as a hint to restrict b within a very limited range on keyboard.

For a reference vector, e.g., $\vec{o}_1(y)$ in Fig. 8a, 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. 10b 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 then calculate:

- $n_p = round(y_p/y)$, which indicates that n_p rows are above $\tilde{b}$ on the keyboard, e.g., $\tilde{b}$ is *at least* on the $(n_p + 1)$th row.
- $n_g = round(-y_g/y)$, which indicates that 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. 10c, $n_p = round(y_p/y) = 2$ 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. 11a-up-left, another sequence, by rotating the keyboard 180 degree, has the same $E_{\mathcal{S}}(b)$ value, e.g., Fig. 11a-up-right. In addition, if $x = y$, two more sequences in Fig. 11a-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 degree) has a similar orientation as keypad's as in Fig. 11b. Our key idea is to use the four sequence candidates in Fig. 11a 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 through comparing the difference between each moving trajectory from four inferred sequences and the recovered trajectory from the wearable respectively.

3.3.3 Handling the Long Sequences

The design in Section 3.3.2 can remove the inference ambiguity and derive user's typed information, however, its computation complexity increases exponentially with the length of the information sequence, even for a single x and y pair. If we further consider the multiple x and y pair candidates and the computation burdens from other components (e.g., keyboard size derivation and moving trajectory recovery), the overall computation overhead in *aLeak* becomes non-negligible (shown in Section 4). Although we can assume very capable computing resources from attackers, a tractable attack (if possible) is definitely more severe in practice. In this section, we

find that the typed long sequences indeed can be handled efficiently with effective technical designs.

Computation Overhead Analysis. To this end, we first analyze the computation overhead of the current *aLeak* design. Supposing there are L vectors in the recovered moving trajectory $\{\vec{o}_l\}$, e.g., $L+1$ keystrokes, the overall computation overhead $N_{overall}$, in terms of the floating-point operations, is the summation of the following three items:

- N_{infer}: the overhead from the information inference (Section 3.3). It can be detailed as $2 \times b \times n_k \times L \times n_b^L$, where b is the number of starting button positions, n_k is the number[5] of x and y pair candidates and n_b is the number of buttons on the keyboard.
- $N_{keyboard}$: the overhead from keyboard size derivation (Section 3.2). It can be detailed as $4 \times L^2 \times (4^L + 3^L)$, where 4 and 3 stand for the 4×3 keyboard layout.
- N_{traj}: the overhead from the moving trajectory recovery (Section 3.1), which is relatively stable and much less than the first two items.

According to the three items above, the overall computation overhead $N_{overall}$ thus can be expressed as

$$N_{overall} = \mathcal{O}(n_k \cdot L \cdot n_b^L + L^2 \cdot (4^L + 3^L)), \tag{4}$$

which inspires us to reduce $N_{overall}$ from the numbers of buttons (n_b), keyboard size pairs (n_k) and vectors (L) these three aspects as follows.

Button Amount n_b. In Section 3.3 for the typed information inference, when we calculate the distance between the ending point of each vector $\vec{o}_i$ and the button centers in Eqn. (3), we in principle need to calculate for all n_b buttons on the keyboard. As n_b is the base of an exponential item in Eqn. (4), we propose to reduce its value by merely calculating for certain most likely (i.e., closest) buttons in Eqn. (3).

In the current *aLeak* design, we empirically select the first two closest buttons in the calculation and $N_{overall}$ thus reduces to $\mathcal{O}(n_k \cdot L \cdot 2^L + L^2 \cdot (4^L + 3^L))$. To understand the computation overhead after this reduction, we conduct experiments on a desktop with Intel Core i7-6700 CPU (detailed in Section 4). Before we reduce the value for n_b, we observe that the execution of *aLeak* is not finished after one hour even when the number of characters in the typed information is merely 4. After this reduction, *aLeak* consumes less than 25 minutes for short sequences, i.e., 4 or 5 characters. However, when the character number is equals to or greater than 6, *aLeak* is still running after several hours, which needs a further improvement.

Keyboard Size Pairs n_k. For the ambiguity removal in Section 3.3.2, after we calculate the parameters n_p and n_g, its summation $n_p + n_g$ indicates the total amount of rows or columns on the keyboard layout using the current keyboard size x and y values. For the y direction, ideally, it should equal to 3 for the 4×3 keyboard layout, but if the result is excessive large, e.g., larger than 7 in the current *aLeak* design, we will discard this keyboard pair and omit its inference to avoid unnecessary computations. Similarly, we set the threshold for the x value as 5, which ideally should equal to 2.

5. In *aLeak* implementation, for each reference vector along x- or y-axis, five x or y values are returned from the dynamic programming in Section 3.2 to improve system robustness due to the unknown errors of $\{\vec{o}_l\}$.

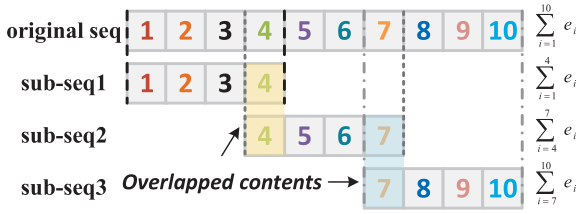

Fig. 12. An instance of the divide-and-combine design for long sequences. The original 10-character sequence is divided into three sub-sequences with one character overlapped.

With this reduced $\overline{n}_k$, we further examine the new computation overhead $\mathcal{O}(\overline{n}_k \cdot L \cdot 2^L + L^2 \cdot (4^L + 3^L))$ with invariant setting. The result shows that it can successfully improve execution efficiency, i.e., reducing the several hours to 43 minutes when the character number is 6. However, the latency for longer sequences is still large, as $N_{overall}$ is essentially an exponential function of L, i.e., dominated by the terms 2^L and $(4^L + 3^L)$.

Vector Number L. To further improve the efficiency, we propose a *divide-and-combine* approach to divide the original typed sequence into a series of *overlapped* small sub-sequences using Algorithm 2. By doing so, we can "decode" each small sub-sequence first (from line 3 to line 7) and then leverage their overlapped contents to combine the derived sub-sequences together (from line 8 to line 11).

For instance in Fig. 12, supposing the error between the ending point of each vector in the sequence and its closest button is e_i. The total error for the whole sequence is $E_{original} = \sum_{i=1}^{10} e_i$. The search space to obtain the top-i, e.g., $i = 1, 2, \ldots, 5$, sequence candidates as the result is thus exponential to L. From experiments, we find that when the sequence length is relatively long, e.g., seven characters, the computation cannot be completed even after four hours.

To further improve the design efficiency, we introduce the *divide-and-combine* approach as shown in Fig. 12, wherein the original sequence (of 10 characters) is divided into 3 sub-sequences (with 1 character overlapped). With the sequence division, we can first decode each sub-sequence independently. Then, for any two consecutive sub-sequences, e.g., for sub-seq1 and sub-seq2 in Fig. 12, only the sub-sequence candidates (sharing a common overlapped character) can be combined to form a longer sub-sequence, and this design aims to minimize $E_{sub} = E_{sub1} + E_{sub2} + E_{sub3}$, where $E_{sub1} = \sum_{i=1}^{4} e_i$, $E_{sub2} = \sum_{i=4}^{7} e_i$ and $E_{sub3} = \sum_{i=7}^{10} e_i$.

For this divide-and-combine approach, if we check all the combinations for every sub-sequence candidate pair, the equivalent search space over all three sub-sequence search spaces will be the same as the direct decoding. In this case, the computation overhead is still high. Therefore, in *aLeak* we introduce the following heuristic approach. For all the candidates for one sub-sequence j, we sort the E_{subj} of each sub-sequence candidate in a decreasing order and only keep the top-n candidates, e.g., $n = 20$. Then we only combine sub-sequences from these top-n candidates.[6]

6. In theory, it is possible that there are no sub-sequence candidates share a common character cross two consecutive with this heuristic approach. However, in our experiments over 100 long password sequences, we have not observed such a phenomenon yet. In case that it happens, we can either gradually increase the number of candidates kept for the combination or terminate this password decoding.

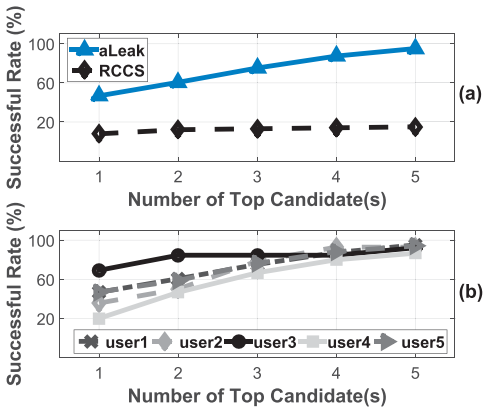

Fig. 13. Successful rates from top-1 to top-5 candidates in *aLeak* (a) compared with RCCS and (b) for different users.

In general, we denote the number of vector number in each sub-sequence as l with m overlapped characters, e.g., $l = 4$ and $m = 1$ in current *aLeak*. The original sequence of $L + 1$ characters will be divided into n sub-sequences, where $n = \lceil (L + 2)/4) \rceil$. Thus, the exponential item in $N_{overall}$ (w.r. t. L) is eliminated and the overall computation overhead can be further reduced to

$$N_{overall} = \mathcal{O}\left(\frac{L}{4} \cdot (\overline{n}_k \cdot l \cdot 2^{l+1} + l^2 \cdot (4^l + 3^l) + \beta)\right),$$
$$= \mathcal{O}\left(\frac{L}{4} \cdot (\alpha + \beta)\right),$$

where $\alpha = \overline{n}_k \cdot 4 \cdot 2^5 + 4^2 \cdot (4^4 + 3^4)$ and β is the additional overhead to combine the common part for two overlapped sub-sequences, which it is lightweight. In Section 4, we find that the execution overhead can be effectively reduced within 2 minutes when the character number changes from 4 to 10.

Algorithm 2. Divide-and-Combine Approach

input: recovered $\{\vec{o}_l\}$, where $L > 3$; $j = 1$;
output: inferred characters in ascending order of error;
while $j < L$ **do**
  $\{(\vec{s}_i, \vec{e}_i)\} = aLeak\,(\{\vec{o}_j, \vec{o}_{j+2}\})$;
  $j = j + 2$; $i = i + 1$;
  if $L - 2 < j < L$ **then**
    $\{(\vec{s}_i, \vec{e}_i)\} = aLeak\,(\{\vec{o}_{L-2}, \vec{o}_L\})$;
for $n = 1 : i$ **do**
  if $n = 1$ **then**
    $\{(\vec{o}_1, \vec{err}_1)\} = \{(\vec{s}_1, \vec{e}_1)\}$;
  $\{(\vec{o}_n, \vec{err}_n)\} =$
    $combine(\{(\vec{s}_n, \vec{e}_n)\}, \{(\vec{o}_{n-1}, \vec{err}_{n-1})\})$;
$sort(\{(\vec{o}_n, \vec{err}_n)\})$;
return $\{(\vec{o}_n, \vec{err}_n)\}$;

4 Performance Evaluation

4.1 Experiment Setup

We develop a prototype system of *aLeak* based on previous designs and examine its performance in this section. We experiment with 5 users wearing LG W150 smart watch and act as the adversary to attack more than 500 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 degree in the experiment. We also vary the sampling rates of the accelerometer and gyroscope data from 30 to 200 Hz.

To validate the efficacy of *aLeak*, we compare it with the state-of-the-art approach RCCS [26]. 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 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.

For *aLeak*, we do not assume the awareness of the last keystroke being on the "enter" button or not. We can virtually extend the keyboard layout with one area (Area 1), i.e., adding another column on the right side of keyboard. In addition, the button on the left and right hand of "0" (Area 2) is also treated as "enter". Thus, for the recovered moving trajectory $\{\vec{o}_l\}$, where $l = 1, 2, \ldots, L$ and L is the total vector number, we can first exclude the last vector $\vec{o}_L$ and apply all previous designs for the first $L - 1$ vectors. For each inferred typed result, we can then add the last vector back, e.g., the starting point of $\vec{o}_L$ is equal with the ending point of $\vec{o}_{L-1}$, and its ending point can be determined by minimizing the error E_S as in Section 3.3.1. If the last vector is determined as a keystroke on any button from these two areas, the last button is inferred as "enter"; Otherwise, the number on the closest button in the original 4×3 keyboard is selected as the password's last character.

4.2 Evaluation Results

4.2.1 Overall Performance

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. 13a, 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 degree with a step size of 10 degree. For each posture angle, we attack a similar amount of users' typings without any context information.

From the result, we can see that even top-5 successful rate of RCCS is 15 percent merely, which is mainly achieved when keypad plane is close to be horizontal (as it assumes). In contrast, *aLeak* can achieve 94 percent top-5 and even 45 percent top-1 successful rates without any context information. The improvements are gained from 4.8x to 5.3x. Fig. 13 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.

In Fig. 13b, we further plot the performance achieved by different users. From the result, we can see that their top-1 successful rates are slightly scattered around 50 percent, while their top-5 successful rates are all very high. Fig. 13b

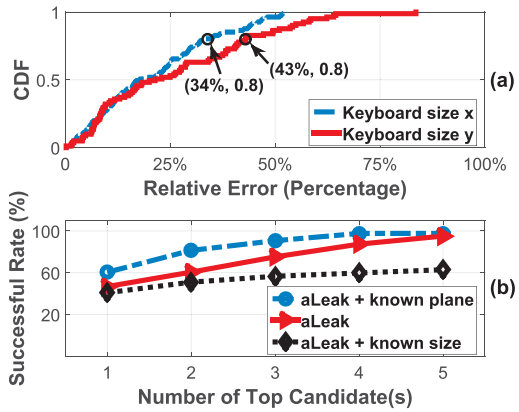

Fig. 14. (a) CDF of the relative errors of the keyboard size derived from *aLeak*. (b) Impact of the keypad posture and size recovery to *aLeak*'s performance.

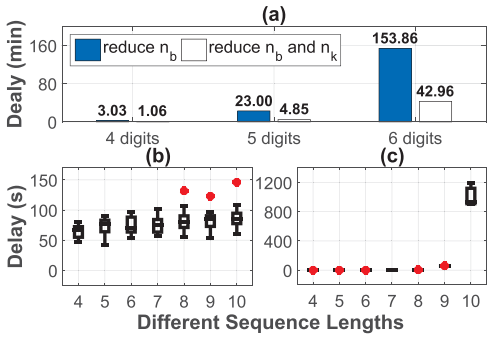

Fig. 15. (a) Inference delays of *aLeak* after reducing button amount n_b and a further reduction of keyboard size pairs n_k. Overall inference delays of (b) *aLeak* and (c) RCCS.

indicates that *aLeak* can achieve an effective wearable side-channel attack on different users.

Recovered Context Information. As *aLeak* does not assume context-related information, its performance highly depends on the accuracy of the recovered keypad postures and keyboard sizes. In Fig. 6 (on p. 5), we have demonstrated that *aLeak* can achieve a good keypad posture recovery, e.g., the average error is less than 7 degree. In this section, Fig. 14a further indicates that the accuracy of the derived keyboard sizes by *aLeak* is accurate as well. Compared with the measured ground truth, the median relative error of keyboard size derivation is 17.6 and 21.4 percent for x and y directions, respectively. 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. 14a. In summary, experimental results show the good performance achieved from both of these two aspects, which ensures *aLeak*'s high successful rates shown in Fig. 13.

In Fig. 14b, 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 percent 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, because we use the smart watch's movement to infer the finger's movement. The effective keyboard size (from smart watch's view) thus could be different from the actual keyboard size, as user's arm may not move in parallel exactly during the typing.

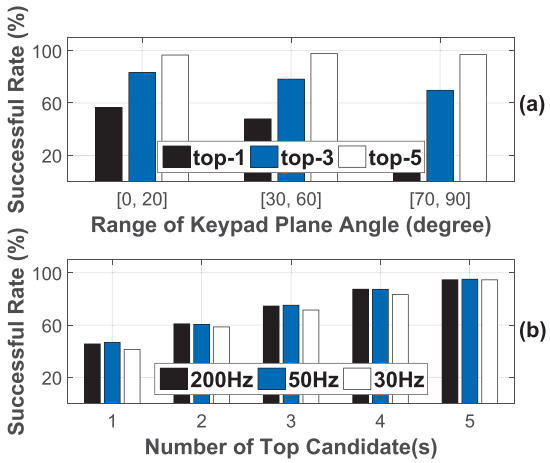

Fig. 16. Performance of *aLeak* (a) under different keypad postures, where posture angles divided into three ranges and (b) under different sampling rates.

Inference Time Complicity. In Section 3.3.3, we propose to reduce three factors, i.e., 1) button amount n_b, 2) keyboard size pairs n_k and 3) vector number L, to speedup the inference process for long sequences. Fig. 15a shows that with the reduction for n_b, the inference can be finished within 25 minutes for short sequences, i.e., 4 or 5 characters (digits), while it is still slow for a longer sequence of 6 characters. With a further reduction of n_k, the inference time can be dramatically improved as in Fig. 15a, but we find that it is still time consuming, e.g., several hours, when the sequence becomes even longer, as the time complicity is an exponential function of L.

With the divide-and-combine design to remove the exponential term of L, Fig. 15b shows that the delay can be effectively reduced within 2 minutes even when the character number is 10. In particular, the average inference time for the 6-character sequences is $75s$ on average and can only be up to $89s$ (w.o. outliers), which is much faster than the 43-minute delay in Fig. 15a without this reduction.

As we remove the last vector and determine its corresponding button separately to handle the "enter" button case (stated in 4.1), the sequences with 5 to 7 characters in Fig. 15a are divided into two sub-sequences and their inference delays are $74s$ on average, which is $10s$ larger than that of the sequence with 4 characters (without sequence division). When the sequence length further increases from 8 to 10, one more sub-sequence inference time contributes to the delay performance, i.e., near $85s$ on average. Overall, the average delay of *aLeak* is from $64s$ to $89s$ in Fig. 15b. In Fig. 15c, we further examine the delay performance of RCCS and find that when the password length is relatively short, e.g., no more than $5s$ for the sequences with 4 to 8 characters. But when the sequence becomes further longer, the delay gets close to or even exceeds *aLeak*, e.g., $58s$ and $1005s$ for the passwords of 9 and 10 characters, respectively.

4.2.2 Micro-Benchmark Performance

Different Keypad Posture Angles. In Fig. 13, we have shown *aLeak*'s overall performance. In Fig. 16a, we further provide its detailed performance under different keypad posture

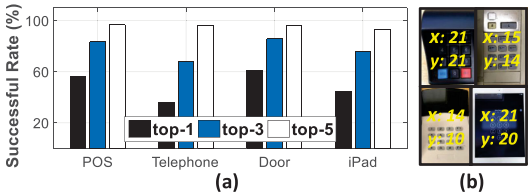

Fig. 17. Performance on different keypads. (a) Successful rates in *aLeak*. (b) Keypads used in the experiment.

angles. For the ease of illustration, we group these angles into three ranges: $0^\circ \sim 20^\circ$, $30^\circ \sim 60^\circ$, $70^\circ \sim 90^\circ$, and depict the performance for each range. From the result, we observe that when the posture angle is closer to 0 degree, the performance tends to be better, e.g., 57 percent versus 30 percent 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 percent for the top-3 and top-5, respectively.

Different Motion Data Sampling Rates. Although wearable device could sample motion data at a high rate, e.g., 200 Hz, the effective sampling rate achieved by the adversary might be much lower [26]. From Fig. 16b, we observe that this side-channel attack is robust to the sampling rate reduction. Even the rate is reduced to 30 Hz (by down-sampling), the achieved successful rates are comparable to the high sampling rates, e.g., 200 and 50 Hz, for all numbers of top candidates. Fig. 16b essentially implies that the barrier to launch this attack is trivial, consistent with the observations from prior studies [14], [26]. In this experiment, we have tried to further lower the sampling rate and found that after the rate is lower than 10 Hz, the system cannot provide meaningful outputs.

Different Keypads. In Fig. 17, we plot *aLeak*'s performance achieved on four different (three physical and one virtual) keypads, where the keyboard size x and y varies from 14 to $21mm$ and 10 to $21mm$, respectively. Overall, *aLeak* achieves similar performance cross different keypads. The top-1 accuracy varies from 36 to 57 percent, and there is no obvious difference for the accuracy 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.

Accuracy Under Different Sequence Lengths. To attack a relatively short sequence, e.g., 4 characters, we can directly derive the accuracies or successful rates for top-1 to top-5 candidates ordered by the calculated sequence errors (Eqn. (3)). However, it may provide less meaningful results if we directly apply it for longer sequences. For instance, if the top-1 accuracy for a short sequence, e.g., 4 to 6 characters, is 45 percent, a long sequence, e.g., with 3 times of shorter sequence's length, in principle can be successfully derived as the top-1 candidate with the likelihood no more than 9 percent ($\approx (45\%)^3$). Moreover, though the top-5 accuracy of the short sequence may be much higher, e.g., 94 percent, the true result within the top 125 candidates for the long sequence is 83 percent ($\approx (94\%)^3$). This is a general phenomenon to decode long sequences, i.e., if one character is wrong (even all the rest is correct), the entire result is wrong. In RCCS [26], the authors mainly consider short sequences, e.g., with 4 or 6 characters, while we find there is still a chance to restrict the inference results of long sequences within a small range, e.g., within top-5.

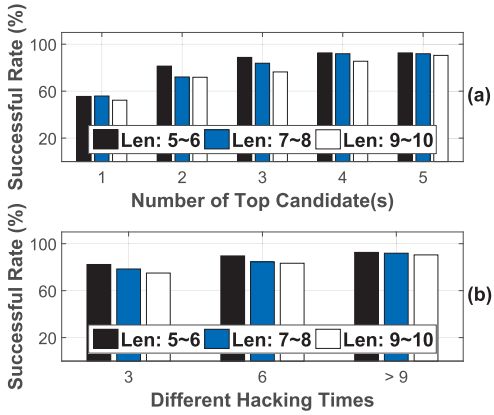

Fig. 18. (a) Successful rates for different sequence lengths. (b) Successful rates of top-5 with different hacking times.

A password can be typed by the user from time to time. After hacking the password multiple times,[7] the attacker can select the sequence appeared with the highest frequency (i.e., majority vote), which tends to obtain the true result, as no incorrect sequences should dominate the statistic results. In particular, we select the most frequently appeared inference result(s) as the former sub-sequence candidate(s). Then, the following sub-sequence will be derived with the same overlapped character as each previous sub-sequence candidate. Finally, we can select the top-5 derived sequences with the smallest errors (Eqn. (3)). Fig. 18a depicts the top-1 to top-5 accuracy for the sequences with 5 to 10 characters (the full password length is from 6 to 11) when the attacker hacks each password more than 9 times. The result shows that we can achieve above 50 percent top-1 and 90 percent top-5 successful rates for the longer sequences as well.

Moreover, we also examine how the successful rate is impacted by the hacking time, i.e., the number of times that the password is decoded (independently) before the majority vote is applied. When a password is obtained by the attacker 3 times, Fig. 18b shows that we can already achieve (by the majority vote) about 80 percent top-5 successful rate. When the hacking time increases to 6, the rate improves to around 85 percent. As Fig. 18a already indicates, when the time is sufficiently large, e.g., > 9, we can achieve above 50 percent top-1 and 90 percent top-5 successful rates for the long sequences.

5 Related Work

Privacy Leakage by Wearable Motion Sensors. Some existing efforts [14], [26], [27] demonstrate the initial success of the keystroke leakage through wearable's motion sensors. MoLe [27] leverages a linguistic model to infer the English word typing on computers keyboards. Liu et al. [14] report the password leakage on POS terminals and cope with the inaccurate motion data using a machine-learning based

7. A user may have multiple passwords, which usually have different lengths. The attack can use the value of L to group different passwords.

approach, which however requires the training and known keypad plane in prior. Wang et al. [26] further release the training requirement and propose a backward inference method to migrate the 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. [28] study such a camera-based attack on mobile phones and Ye et al. [31] 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 [18] 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 [7] and gyroscope can be used to unveil the fingerprints of user's typing [17]. Furthermore, acoustic signal can also be utilized to infer the typed characters on a keyboard [35]. These existing works are essentially orthogonal to this paper, which do not address the unique challenges in the *aLeak* design.

Motion Sensor Data Processing. Motion sensor data from wearables sniffed by *aLeak* are acceleration and angular speed captured by accelerometer and gyroscope. The accelerometer data is utilized to compute the lengths of moving vectors through double-integration and gyroscope is used to perform the coordinate transformation of acceleration. However, there are wide blames for their inaccuracy [6], [21], [24]. Existing work [27] addresses the noisy acceleration by calibrating each acceleration sample to refine the estimation of speed calculation. And [33] proposes a mechanism for accurate phone attitude detection through utilizing gyroscope carefully and intelligently. In addition, the Android API, TYPE_GAME_ROTATION_VECTOR, also provides an approach to perform coordinate transformation accurately through sensor fusion [1]. In *aLeak*, we design our system based on these existing works and then propose our designs to address the unique challenges in *aLeak*.

6 Discussion on Defense Mechanisms

We discuss the possible defense mechanisms in this section.

Keyboard Layout Randomization. As stated in Section 2.1, a randomized keyboard layout can be an effective defense mechanism against this wearable side-channel attack. Hence, if a keyboard (e.g., the virtual keyboard on phone or ATM) can support the layout randomization feature, it is suggested to be enabled to better protect the user's typing privacy. On the other hand, even for the keyboards of physical buttons without layout randomization, there are other two potential defense mechanisms as follows.

Sensor Data Management. The sensor data management on wearable devices can be further improved to defend this attack. Nowadays, applications can easily obtain the permission to access motion sensors, e.g., this permission is completely open on Android OS, which thus provides attackers the opportunity to launch this attack (Section 2.1). With an improved motion sensor permission management, a user can grant this permission only to the applications that the user believe are indeed necessary to access motion sensors and/or are developed by well-known or trusted companies (minimizing the chance of the motion data usage for malicious purposes). On the other hand, even the permission can by granted to one APP, the APP (like fitness or game APP) may not need a high sampling rate, e.g., 50 Hz. In our experiment Fig. 16b, we find that when the motion sensor sampling rate is low, e.g., 10 Hz, the system cannot provide meaningful attacking outputs, while this reduced sampling rate may still fulfill the function of the application itself, e.g., step counting. Therefore, in addition to the sensor permission management, a fine-grained sampling rate control can also be investigated to migrate this side-channel attack.

Typing Styles and Methods. As stated in Section 2.1, the attack on the user's typing privacy is effective (in *aLeak* and recent studies) when the smart watch moves along with the user's finger during the typing. The user can thus intentionally change her typing style when inputting sensitive information. Another alternative is to adopt new input methods, e.g., gaze-based typing [13], which can also defend this attack since no hand motions are involved in such a typing design.

7 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 and develop a prototype system to validate the design feasibility. Extensive experiments by attacking more than 500 rounds of different users' typings without context information indicate the efficacy of *aLeak*.

Acknowledgments

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). A preliminary version of this study has been published in the *Proceedings of IEEE INFOCOM 2018* [15].

References

[1] Android sensor event. [Online]. Available: https://developer.android.com/reference/android/hardware/SensorEvent.html#values , Last accessed 2019.
[2] Material design. [Online]. Available: https://material.io/devices/, Last accessed 2017.
[3] Mean removal. [Online]. Available: https://wiki.multimedia.cx/index.php/Mean_Removal, Last accessed 2019.
[4] Touch ID. [Online]. Available: https://en.wikipedia.org/wiki/Touch_ID, Last accessed 2019.
[5] Wired keyboard sale. [Online]. Available: http://cn.made-in-china.com/Computer-Products-Catalog/Keyboard.html, Last accessed 2017.

[6] S. Agrawal, I. Constandache, S. Gaonkar, R. Roy Choudhury, K. Caves, and F. DeRuyter, "Using mobile phones to write in air," in *Proc. ACM Int. Conf. Mobile Syst. Appl. Serv.*, 2011, pp. 15–28.
[7] L. Cai and H. Chen, "TouchLogger: Inferring keystrokes on touch screen from smartphone motion," in *Proc. USENIX Conf. Hot Topics Secur.*, 2011, pp. 9–9.
[8] X. Guo, J. Liu, and Y. Chen, "FitCoach: Virtual fitness coach empowered by wearable mobile devices," in *Proc. IEEE INFOCOM*, 2017, pp. 1–9.
[9] 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 Trans. Netw.*, vol. 24, no. 2, pp. 846–858, Apr. 2016.
[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[11] X. Ji, Y. He, J. Wang, K. Wu, D. Liu, K. Yi, and Y. Liu, "On improving wireless channel utilization: A collision tolerance-based approach," *IEEE Trans. Mobile Comput.*, vol. 16, no. 3, pp. 787–800, Mar. 2017.
[12] 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. IEEE INFOCOM*, 2016, pp. 1–9.
[13] Z. Li, M. Li, P. Mohapatra, J. Han, and S. Chen, "iType: Using eye gaze to enhance typing privacy," in *Proc. IEEE INFOCOM*, 2017, pp. 1–9.
[14] X. Liu, Z. Zhou, W. Diao, Z. Li, and K. Zhang, "When good becomes evil: Keystroke inference with smartwatch," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2015, pp. 1273–1285.
[15] Y. Liu and Z. Li, "aLeak: Privacy leakage through context-free wearable side-channel," in *Proc. IEEE INFOCOM*, 2018, pp. 1232–1240.
[16] A. Maiti, M. Jadliwala, J. He, and I. Bilogrevic, "(Smart) watch your taps: Side-channel keystroke inference attacks using smartwatches," in *Proc. ACM Int. Symp. Wearable Comput.*, 2015, pp. 27–30.
[17] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. R. Choudhury, "TapPrints: Your finger taps have fingerprints," in *Proc. ACM Int. Conf. Mobile Syst. Appl. Serv.*, 2012, pp. 323–336.
[18] X. Pan, Z. Ling, A. Pingley, W. Yu, N. Zhang, K. Ren, and X. Fu, "Password extraction via reconstructed wireless mouse trajectory," *IEEE Trans. Depend. Secure Comput.*, vol. 13, no. 4, pp. 461–473, Jul./Aug. 2016.
[19] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proc. 30th Int. Conf. Int. Conf. Mach. Learn.*, 2013, pp. III-1310–III-1318.
[20] W. Qi, W. Ding, X. Wang, Y. Jiang, Y. Xu, J. Wang, and K. Lu, "Construction and mitigation of user-behavior-based covert channels on smartphones," *IEEE Trans. Mobile Comput.*, vol. 17, no. 1, pp. 44–57, Jan. 2018.
[21] A. Rai, K. K. Chintalapudi, V. N. Padmanabhan, and R. Sen, "Zee: Zero-effort crowdsourcing for indoor localization," in *Proc. ACM Annu. Int. Conf. Mobile Comput. Netw.*, 2012, pp. 293–304.
[22] M. Ryan et al., "Bluetooth: With low energy comes low security," in *Proc. 7th USENIX Conf. Offensive Technol.*, 2013, pp. 4–4.
[23] L. Shangguan, Z. Yang, A. X. Liu, Z. Zhou, and Y. Liu, "STPP: Spatial-temporal phase profiling-based method for relative RFID tag localization," *IEEE/ACM Trans. Netw.*, vol. 25, no. 1, pp. 596–609, Feb. 2017.
[24] G. Shen, Z. Chen, P. Zhang, T. Moscibroda, and Y. Zhang, "Walkie-Markie: Indoor pathway mapping made easy," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2013, pp. 85–98.
[25] D. Spill and A. Bittau, "BlueSniff: Eve meets Alice and bluetooth," in *Proc. 1st USENIX Workshop Offensive Technol.*, 2007, pp. 1–10.
[26] C. Wang, X. Guo, Y. Chen, Y. Wang, and B. Liu, "Personal PIN leakage from wearable devices," *IEEE Trans. Mobile Comput.*, vol. 17, no. 3, pp. 646–660, Mar. 2018.
[27] H. Wang, T. T.-T. Lai, and R. Roy Choudhury, "MoLe: Motion leaks through smartwatch sensors," in *Proc. ACM Annu. Int. Conf. Mobile Comput. Netw.*, 2015, pp. 155–166.
[28] L. Wu, X. Du, and X. Fu, "Security threats to mobile multimedia applications: Camera-based attacks on mobile phones," *IEEE Commun. Mag.*, vol. 52, no. 3, pp. 80–87, Mar. 2014.
[29] L. Xie, X. Dong, W. Wang, and D. Huang, "Meta-activity recognition: A wearable approach for logic cognition-based activity sensing," in *Proc. IEEE INFOCOM*, 2017, pp. 1–9.
[30] P. Xie, J. Feng, Z. Cao, and J. Wang, "GeneWave: Fast authentication and key agreement on commodity mobile devices," in *Proc. IEEE Int. Conf. Netw. Protocols*, 2017, pp. 1–10.
[31] G. Ye, Z. Tang, D. Fang, X. Chen, W. Wolff, A. J. Aviv, and Z. Wang, "A Video-based Attack for Android Pattern Lock," *ACM Trans. Privacy Secur.*, vol. 21, no. 4, pp. 19:1–19:31, 2018.
[32] 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 Trans. Mobile Comput.*, vol. 14, no. 9, pp. 1888–1902, Sep. 2015.
[33] P. Zhou, M. Li, and G. Shen, "Use it free: Instantly knowing your phone attitude," in *Proc. ACM Annu. Int. Conf. Mobile Comput. Netw.*, 2014, pp. 605–616.
[34] P. Zhou, Y. Zheng, and M. Li, "How long to wait?: Predicting bus arrival time with mobile phone based participatory sensing," in *Proc. ACM Int. Conf. Mobile Syst. Appl. Serv.*, 2012, pp. 379–392.
[35] L. Zhuang, F. Zhou, and J. D. Tygar, "Keyboard acoustic emanations revisited," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 1, 2009, Art. no. 3.

Yang Liu received the BE degree from Xi'an Jiaotong University, China, in 2016. She is currently working toward the PhD degree in the Department of Computer Science, City University of Hong Kong. Her research interests include mobile and wearable sensing, mobile computing, deep learning, and cyber security and privacy. She is a student member of the IEEE.

Zhenjiang Li received the BE degree from Xi'an Jiaotong University, China, in 2007, and the MPhil and PhD degrees from the Hong Kong University of Science and Technology, Hong Kong, in 2009 and 2012, respectively. He is currently an assistant professor with the Department of Computer Science, City University of Hong Kong. His research interests include wearable and mobile computing, cyber-security, deep learning, and distributed computing. He is a member of the IEEE.