# MoLe: Motion Leaks through Smartwatch Sensors

He Wang
University of Illinois at Urbana-Champaign
Champaign, IL, USA
hewang5@illinois.edu

Ted Tsung-Te Lai
University of Illinois at Urbana-Champaign
Champaign, IL, USA
tedlai@illinois.edu

Romit Roy Choudhury
University of Illinois at Urbana-Champaign
Champaign, IL, USA
croy@illinois.edu

## ABSTRACT

Imagine a user typing on a laptop keyboard while wearing a smart watch. This paper asks whether motion sensors from the watch can leak information about what the user is typing. While its not surprising that some information will be leaked, the question is *how much?* We find that when motion signal processing is combined with patterns in English language, the leakage is substantial. Reported results show that when a user types a word $W$, it is possible to shortlist a median of 24 words, such that $W$ is in this shortlist. When the word is longer than 6 characters, the median shortlist drops to 10. Of course, such leaks happen without requiring any training from the user, and also under the (obvious) condition that the watch is only on the left hand. We believe this is surprising and merits awareness, especially in light of various continuous sensing apps that are emerging in the app market. Moreover, we discover additional "leaks" that can further reduce the shortlist – we leave these exploitations to future work.

## Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection — Invasive software; C.3 [**Special-purpose and Application-based Systems**]: Real-time and embedded systems

## Keywords

motion leaks; smartwatch; side-channel attacks; accelerometer; gyroscope; security; malware; Bayesian inference; gesture

## 1. INTRODUCTION

Rich sensors on wearable devices are offering valuable data, enabling important applications in mobile health, user-interfaces, context-awareness, activity tracking, gaming, etc. Of course, such data are often "double edged swords" since they leak information about aspects of lives that are considered private. In our struggle to define what level of data exposure is appropriate, the core question often comes to: *what can be inferred from a given sensor data?* Every so often, we find that highly surprising inferences can be made from an apparently harmless data, forcing us to push back on information exposure. While this is a broad area of research, and immense work has been performed in this direction, new platforms and applications warrant a continuous vigil on information leakage. This paper looks into a narrow piece of this general problem. We ask: *can accelerometer and gyroscope data from smart watches be mined to infer the words that a user is typing.* In other words, given that a user's wrist moves in the granularity of few centimeters while typing, can the corresponding motion data be used to derive the keys that the user has typed? If so, the ramifications are serious – a smart watch app can be disguised as an activity tracker to heavily leak a user's emails, search queries, and other keyboard-typed documents. Unlike keystroke loggers that need to find loopholes in the operating system, the activity tracker malware can obtain the user's permission and easily launch a side channel attack.

Of course, this is not the first work that combines motion data processing with language structure to infer higher level semantics. Recent research have explored various systems and applications, including writing in the air [1], remote control [2], gesture-based signing and authentication [3, 4], smoking gestures [5], etc. While all these systems bear similarity in abstraction, unique challenges (and opportunities) emerge when a particular application is addressed end to end. In our case, we find that the absence of data from the right hand is a unique constraint, and so is the issue of inferring which finger executed the key-press. For a given position of the wrist watch, any one of 3 or 4 different keys could have been pressed, which could be further interspersed by unknown number of keys pressed by the right hand. Moreover, not all users type with equal dexterity – some use their little finger far less efficiently while others use specific fingers when it comes to digits or corner keys. Finally, detecting the typed key is also a function of where the finger was previously, injecting a notion of dependency between consecutive inferences. With these and more application-specific issues, global typing or motion models do not apply. While fundamentally new signal processing or learning algorithms may not be needed, modifying existing techniques and systematically integrating them into a whole is the crux of our contribution.

Importantly, the application of typing also offers a number of opportunities that should be leveraged to improve the inference capability of the watch. For instance, the watch motion is mostly confined to the 2D keyboard plane, in contrast to 3D gestures in air in other applications [3,6]. The orientation of the watch is relatively uniform across various users and, in many users, moves


*MobiCom'15,* September 7–11, 2015, Paris, France.
 ISBN 978-1-4503-3619-2/15/09 ...$15.00.
DOI: http://dx.doi.org/10.1145/2789168.2790121.

back to a reference position while typing (the "F" and "J" keys). Finally, knowing spelling priors from English dictionary further helps in developing Bayesian decisions. The combination of these challenges and opportunities motivates the research, with the aim of quantifying the degree of information leakage.

This paper develops *Motion Leaks* (*MoLe*), a completely functional system on Samsung Gear Live smart watches. Briefly, two of the authors pretend to be attackers and type 500 words each wearing the smart watch on their left wrist. The accelerometer and gyroscope data is used as training data, and processed through a sequence of steps, including key-press detection, hand-motion tracking, character point cloud computation, and Bayesian modeling and inference. Then, 8 different volunteers are recruited, and each asked to type 300 different English words from a dictionary. The smart-watch sensor data from the volunteers are transferred to our server, which then short-lists $K$ words, ranked in the decreasing order of probability (i.e., the first ranked word is considered the most probable guess). The actual words typed by volunteers are then revealed and each word's rank computed from the short-list.

We plot the distribution of rank across all the typed words. With this being the core of the evaluation methodology, we obviously test for various parameters and conditions, including different word lengths, sensor sampling rate, different keyboards, etc. Current limitations of this work include: (1) inability to infer non-valid English words, such as passwords; (2) scalability across different watch models; (3) inability to parse sentences due to difficulties in detecting the "space bar". We have also not tested with other wearable devices, such as *Fitbits* – we believe with some customization, the attacks can be launched on those platforms as well.

In light of these, the main contributions in the paper may be summarized as:

- *Identifying the possibility of leakage when users type while wearing a smart watch.* Developing the required building blocks through techniques in key-press detection, hand-motion tracking, cross-user data matching, and Bayesian inference.
- *Developing the system on Samsung Gear Live smart watches and experimenting with real users.* Performing experiments across 8 users and revealing how typed words can be inferred with reasonable accuracy. Individuals who came to know about our results expressed a sense of alarm and suggested that the findings be disseminated publicly.

The rest of the paper expands on these contributions, beginning with some groundwork and measurements, followed by system overview, assumptions, design detail, and evaluation.

## 2. SMART WATCH DATA: A FIRST LOOK

To understand the problem landscape, we take a first look into the data from smart watches. Basic questions pertain to the amount of wrist displacement for typed keys, whether displacements for nearby keys are even visually discernible, whether the displacements are consistent over time, etc. To this end, two of the authors wore a smart watch and recorded the accelerometer and gyroscope data as they typed each character one by one. The positive X axis of the watch is parallel to the arm and pointed towards the fingers, the positive Y axis is perpendicular and upward, and the positive Z axis pointed upwards from the plane of the arm. To capture ground truth, we placed a phone camera right on top of the keyboard and recorded video at 30 fps (Figure 1). A green and a yellow sticker placed on the watch helps with tracking the watch movement by using computer vision techniques. The watch, the phone camera, and the keyboard logger were all time–synchronized via the network time protocol (NTP). The synchronization offers precise correspondence between the sensor and visual data, extending semantic meaning to the motion signals.

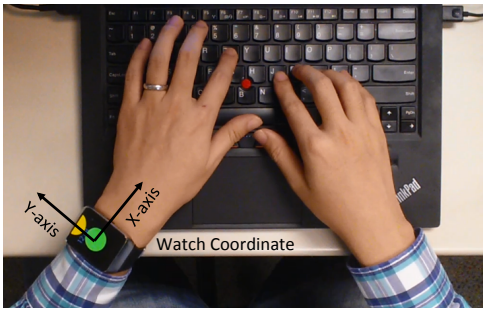


**Figure 1: Watch coordinate system and ground truth measurement by recording hand typing with a smartphone camera view.**

Figure 2 shows an example sequence of video frames capturing the process of typing the character "T". The left hand starts from a home position (i.e., the key "F"), moves along the +X direction to press "T", hits the key, and returns back to the home position. The yellow arrow on the arm shows the displacement of the green marker on the watch.

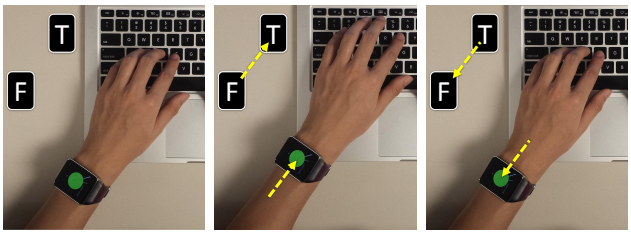


**Figure 2: 3 video frames show the process of typing "T" from "F".**

Figure 3 plots motion data from 20 different characters located on the left side of the keyboard. For each graph, the X axis is time and the Y axis is the displacement of the watch computed from the accelerometer's X axis data (the accelerometer's Y and Z axes are not shown). The light gray vertical bar in each graph marks the time of the key-press, obtained from the keyboard logger. Observe that the displacements align well with the keyboard's layout. The first row (12345) generates the largest positive displacement and the last row (zxcvb) produces negative displacement. The left fingers are initially placed on the third row (asdf), so nearly no displacement is detected for these characters. Although preliminary, these signals offer first indication of information leakage through watches.

Figure 4 shows the watch displacement for the same 20 keys, but in 2D space (i.e., using the combined X and Y axes data from the accelerometer). Each color represents one row on the keyboard. While some keys (e.g., 1, t, r, 4, 5) are quite isolated, others overlap strongly – in particular, "asdf", "zxcv" and "q23" exhibit the strongest overlaps. This is not surprising. Cluster "asdf" is an

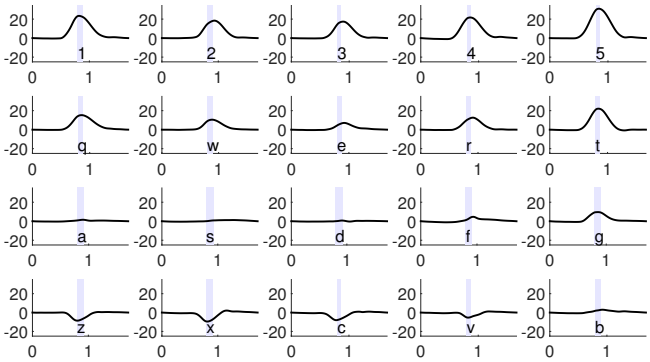


**Figure 3: The watch X axis displacements while a human types 20 characters. In the figures, X axis is time in seconds and Y axis is watch X axis displacement in millimeter. The gray bar shows the keystroke press and release time interval.**

outcome of the fingers being on these keys in the home position – the wrist hardly needs to move when typing these keys. Similarly, the fingers move uniformly downward for "zxcv" resulting in similarity between the keys. Finally, the hand movement for "q" is similar to "2" and "3" even though they are all not on the same row. This is because the little finger is shorter, and to type the character "q", it must move as much as the ring finger must move to type "2".

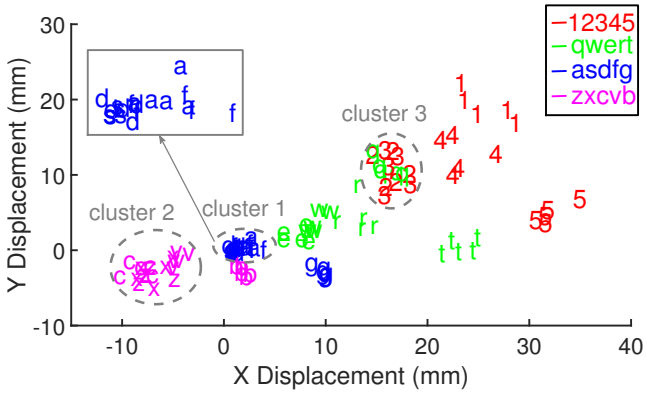


**Figure 4: Watch 2D displacements while a human types 20 characters with her left hand. Each character is typed repeatedly 5 times. (0,0) is the initial location when left hand fingers are placed on home position ("asdf"). Note that X and Y axes in the graph are in the watch's coordinate system.**

Decoding characters gets more complicated when the user types a word rather than just a single character. Figure 5 shows the sequence of hand displacements where the word "teacher" is typed. Obvious issues emerge: The wrist motion for each character is no longer aligned with the earlier observations since the motion is relative to the previous position of the key. Observe that "e", "a" and "c" are all far away from their respective clusters detected earlier in Figure 4. Moreover, we did not record "h" (pressed by the right hand), rather a small random motion of the left hand during this time. Finally, real world environments do not have cameras, and hence the data is completely unlabeled – a wrong decision about any of the keys can derail all subsequent decisions. In sum, while sensor data from smart watches indeed encode the human–typed information, decoding them reliably in real world conditions presents non-trivial challenges.

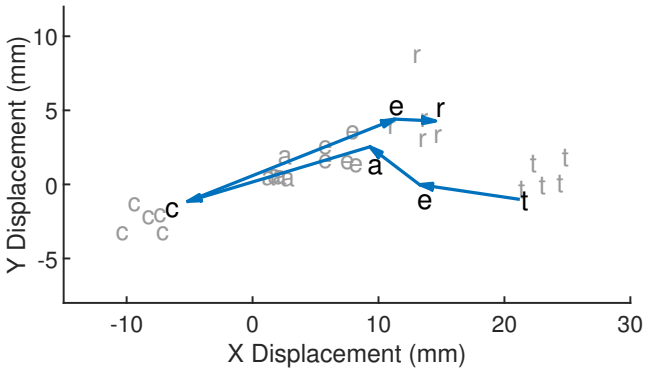


**Figure 5: Comparison of typing "teacher" continuously (in black) against each character separately (in gray). Note that the positions of "e", "a" and "c" are away from their original points due to sequential typing. Also, "h" is not captured due to right hand typing.**

## 3. SYSTEM OVERVIEW

This section presents a functional overview of *MoLe*; details of the technical building blocks will follow in Section 4. In the scenario of interest, we assume that the attacker has successfully installed the *MoLe* app in the user's smart watch and is receiving accelerometer and gyroscope data at the *MoLe* cloud server.

Figure 6 illustrates the flow of operations in the end to end *MoLe* system. At the backend server, the attacker types each character on a computer keyboard multiple times and computes a *character point cloud (CPC)* similar to the one in Figure 4. The operation is performed offline, and is stored separately for use later. The cloud can also be computed from multiple people (e.g., accomplices of the attacker) strengthening the robustness of the attack.

Now, when the raw sensor data from the user arrives, it is passed through a "Keystroke Detection" module, responsible for two tasks. (1) It detects the timing of each key stroke by analyzing the $Z$ axis of the sensor data – every time a user presses a key, the watch exhibits a discernible dip in the negative Z axis. (2) It computes the net 2D displacement of the watch by processing the signal through multiple steps, including gravity removal, mean removal, double integration, and Kalman Filtering. The output of the keystroke detection module is a set of $< location_i, time_i >$ tuples, where $location_i$ denotes the estimated location of the watch at $time_i$ when the key was pressed. When all the locations are plotted on the 2D plane, an *unlabeled point cloud (UPC)* emerges (note that the characters corresponding to each point in this cloud is not known). Figure 7 shows a comparison of the character point cloud developed offline by the attacker and an unlabeled point cloud computed from the attackee's data.

The UPC is forwarded to the "Cloud Fitting" module whose task is to assign approximate labels to the points in UPC. For this, the cloud fitting module obtains the CPC that was computed earlier, and scales and rotates the convex hull of the CPC to best fit the convex hull of the UPC. The output is a rotated and scaled CPC which serves as the *reference template* for decoding the unlabeled points in UPC.

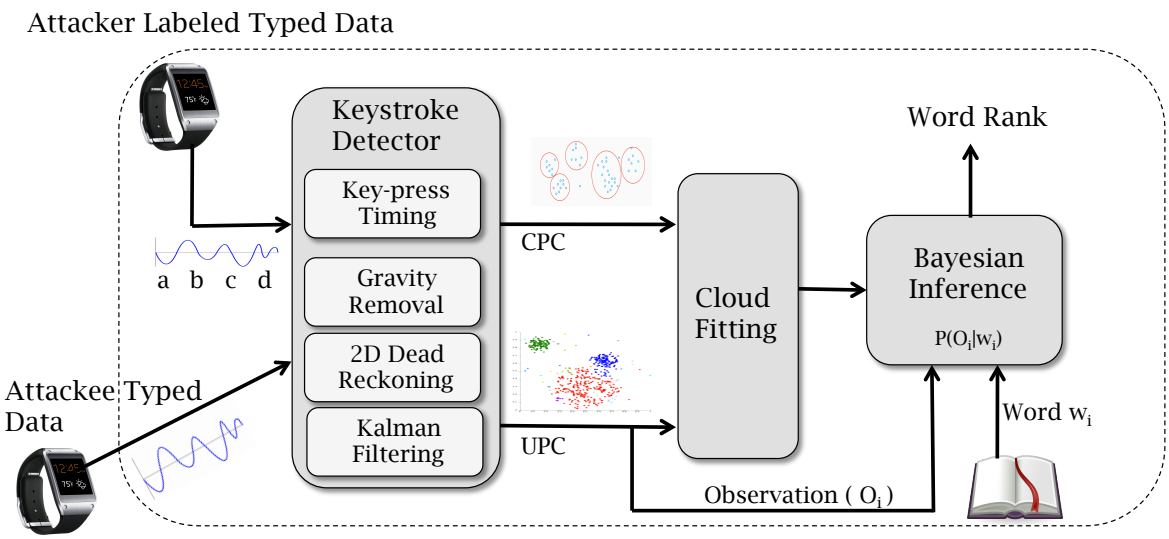


**Figure 6: System Overview: The typed data from users are pre-processed through gravity removal and timing analysis blocks, superimposed on the refitted typing templates, and passed through a Bayesian inference model that leverages the patterns and structures in English words to ultimately decode the typed words. Note, training is only required from the attacker's end; no training needed for the user.**

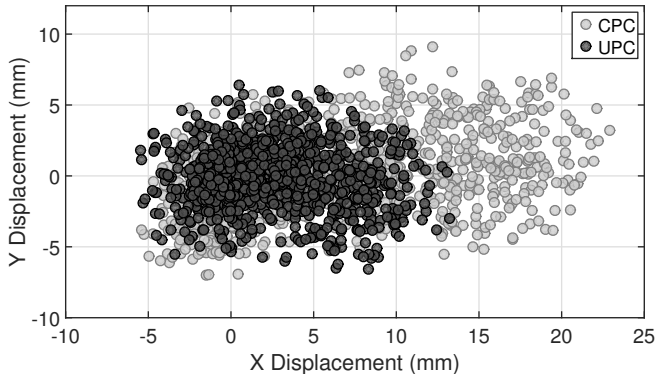


**Figure 7: (a) Character point cloud computed from attackers data; (b) Unlabeled point cloud computed from user's data.**

A "Bayesian Inference" module now accepts three items as input: (1) the template output from Cloud Fitting, (2) the unlabeled points from the UPC, and (3) a dictionary $W$ of valid English words, $w_i$[1]. Briefly, for each valid word $w_i$, the Bayesian Inference module (BIM) computes the *a posteriori* probability that the unlabeled points form $w_i$. For instance, if $w_i$ is the word "dear", BIM computes the probability that the first unlabeled point is "d", the second unlabeled point is "e", and so on. The product of the probabilities is the final probability that the unlabeled points is the word "dear". BIM computes this probability for each word $w_i$, and outputs a ranked list of $< word, probability >$ tuples as a guess of the user-typed word. If its a password, the attacker can now try out all the guesses above some probability threshold; if its an email or a search query, the attacker could manually try to decode the text from the possible sets of words. Even though *MoLe* does not offer a single suggestion, the probability estimate associated to each guess dramatically reduces the search space for the attacker. Results in Section 5.2 will quantify this reduction from the attacker's point of view.

[1]For practical purposes, $W$ contains the 5000 most frequently used English words, available from [7].

## 3.1 Assumptions

Before moving forward, we intend to enumerate a number of assumptions we make. These assumptions make *MoLe* inadequate for launching a real life attack, however, we believe that the assumptions are not fundamental and can be relaxed with some more work.

- The evaluation is performed in a controlled environment where volunteers type one word at a time (as opposed to free-flowing sentences).
- We assume valid English words – passwords that contain interspersed digits, or non-English character-sequences, are not decodable as of now.
- We have used the same Samsung smart watch model for both the attacker and the user – in reality the attacker can generate the CPC for different watch models and use the appropriate one based on the user's model.
- We assume the user is seasoned in typing in that he/she roughly uses the appropriate fingers – novice typists who do not abide by basic typing rules may not be subject to our proposed attacks.

Under these assumptions, the design details and evaluation of *MoLe* are presented next.

# 4. DESIGN DETAILS

We describe the main techniques executed by each module in Figure 6.

## 4.1 Keystroke Detector

Given sensor signals as input, this module is responsible for computing the time and location of each key-press present in the signal. The location is essentially a 2D vector with the origin as the "F" key on the keyboard. Aggregating all the key-press locations will yield the point cloud as discussed earlier.

## Key-press Timing

The intuition to detect key presses is rooted in the hand's motion in the vertical direction. When the finger dips while typing a key, the wrist also undergoes a partial dipping motion, expected to reflect in the Z axis of the watch. Figure 8 shows an example of the Z axis motion when the user types the word "administrative". Using ground truth, we observe that the actual key presses generally produce prominent peaks, however, false positives and false negatives occur. False positive occur mainly during transition from one key to another – the hand moves up slightly to make the movement, which manifests in Z-axis motion. False negatives typically arise due to subtle Z axis motion for keys like "asdf" that can go undetected.

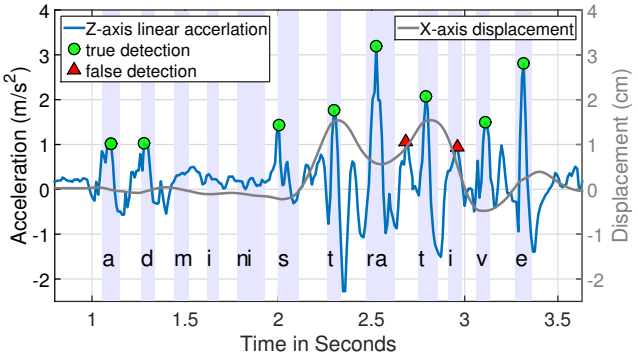


**Figure 8: A simple peak detection scheme to detect keystrokes. The left Y-axis represents acceleration and the right Y-axis indicates displacement. Note that, for "a","d","s" keystrokes, lower Z-axis acceleration is generated because of left hand's initial position. At time 2.7 and 3 seconds, there are two false detections due to the left hand moving from "a" to "t" and from "t" to "v".**

To cope with these issues, we use *bagged decision trees* to classify keystrokes. A bagged decision tree is a ensemble classifier that trains multiple decision trees by selecting different subsets of feature and training examples. The classifier improves the stability and accuracy by letting each subtree learn on the attacker's labeled data, apply the learning to the unlabeled data, and then compute the final results via voting. To obtain the labeled data, we first apply a simple threshold-based peak detection method [8] on the $Z$ axis acceleration, and label true/false detection on the attacker's template. We purposely set the peak detection threshold to be low so that we do not miss true keystrokes. Then we extract features within a time window around the labels and train the classifier.

The feature set includes: the width, height, prominence of the $Z$ axis peak; the mean, variance, max, min, skewness, and kurtosis for each of the 3-axis displacement, velocity, acceleration, and gyroscope rotation; the magnitude of acceleration/gyroscope; and finally the correlation of each pair between acceleration and gyroscope vectors. When the attackee's sensor data arrives, we apply the same peak detection scheme and obtain many candidate keystrokes and their features. Then the classifier identifies the validity of the keystroke and selects the $max$ value of $Z$ axis acceleration to denote the timing of the key-press. Figure 9 shows an example of the classification result of the word "administrative".

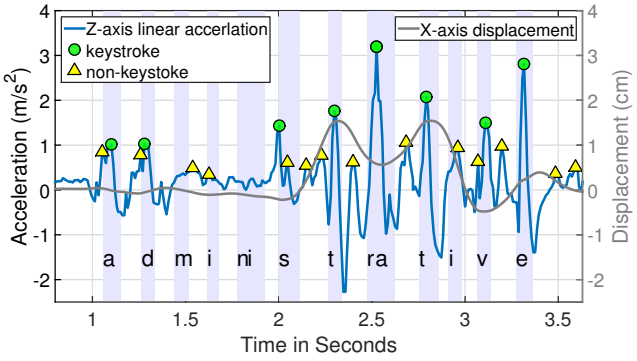


**Figure 9: Bagged decision classification results: A peak detection tool with low thresholds is first applied to the Z-axis acceleration data and marks potential keystrokes (both yellow triangles and green circles). The classifier then identifies whether the peaks are keystrokes or not. Note that for the first "a" and "d", since two peaks are too close, the classifier would identify only one peak with highest Z-axis acceleration within a time window.**

Figure 10 shows the detection rate for each key press when using one author's template model to test on 8 different volunteer recruited in section 5.1 (and ground truth recorded by the keyboard logger software). Expectedly, the keys pressed by the right hand are largely undetected.

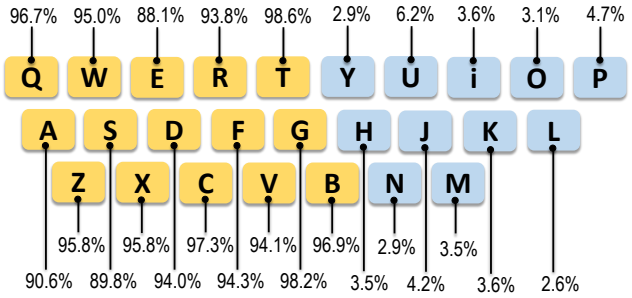


**Figure 10: Keystroke detection rate for each character.**

## Key-Press Location Estimation

The core challenge here pertains to tracking the hand motion as it moves from one key to another, and therefrom, infer the location of each key-press. Tracking over time is non-trivial since the required accuracy is high (in the granularity of key sizes); moreover, incorrect detection of one key will affect subsequent results. The hope we have is that the the left index finger periodically moves back to the home key "F", and hence, its an opportunity to recalibrate the tracking process at the start and end of a sequence of typed characters.

As a first cut, we used the linear acceleration (offered by the native Android API) to compute displacement – we applied established double integration and mean removal techniques. Figure 11 compares the $X$ displacement computed by Android API and *MoLe*, against ground truth (available from the camera). The errors proved inadequate for our purposes. Hence, we developed an improved tracking technique tailored to the *MoLe* application. We define each of the steps below.

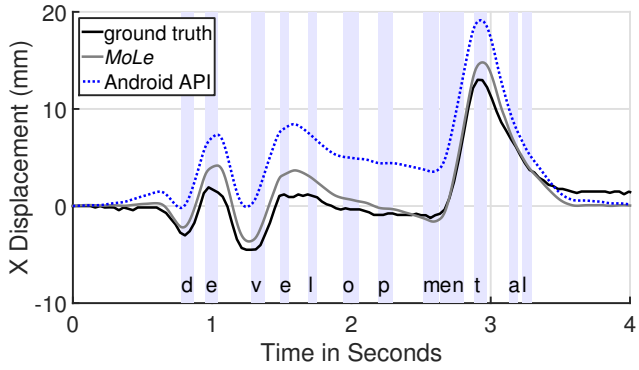


**Figure 11: Comparison of *MoLe* and Android API x-axis displacement results. Y-axis has similar results but is omitted due to space.**

**(1) Find gravity to define an absolute coordinate system.** Before the attacker (or attackee) starts to type, his/her hand is stable – we use this opportunity to estimate the direction of gravity in the watch's coordinate system; we can then estimate the orthogonal plane, which is the absolute horizontal plane. We then project the watch's $X$ axis to the absolute horizontal plane to get the absolute $X$ axis. Since the absolute $Z$ axis is essentially along the direction of gravity, the cross product of $Z$ and $X$ axes yields the absolute $Y$ axis. $C = (X, Y, Z)$ is the absolute coordinate system (represented by the watch's coordinate system). Of course, since the wrist orientation changes during typing, this representation changes as well. Thus, at the starting point, we represent the absolute coordinates as $C(0) = (X(0), Y(0), Z(0))$.

**(2) Estimate and remove gravity.** From the gyroscope, we estimate the rotation matrix over time $R(t)$ and use it to estimate the variation in watch's gravity $g(t)$, in the watch's coordinate system. We sample acceleration $a(t)$ in watch's coordinate system and it is polluted by gravity. Now we remove gravity and get $a_{rg}(t) = a(t) - g(t)$.

**(3) Estimate $C(t)$ and calculate projected acceleration.** Note that directly integrating $a_{rg}(t)$ has no physical meaning even if gravity has already been removed. This is because $a_{rg}(t)$ is along the watch's axes and watch rotates overtime. Ideally, we want to integrate along fixed directions of the absolute coordinate system. Those directions are $X(t)$, $Y(t)$ and $Z(t)$ mentioned before in $C(t)$ – we can get $C(t)$ with the help of the rotation matrix $R(t)$. Thus, $a'_{rg}(t)$ can be obtained by projecting $a_{rg}(t)$ to $X(t)$, $Y(t)$ and $Z(t)$. Integrating $a'_{rg}(t)$ now yields the speed $v'(t)$, and integrating $v'(t)$ ultimately returns the displacement $s'(t)$.

**(4) Calibrate by mean removal (speed and displacement).** Of course, this is erroneous, however, if we know that at time $T$, $v'(T) = 0$ and $s'(T) = 0$ (i.e., the watch has come to a stop), we can refine the estimates of speed and displacement by mean removal.

**(5) Kalman smoothing.** The displacement estimation is still not stable – occasionally the result becomes poor. We carefully checked the data and detected that gravity estimation is not entirely reliable. Thus, we apply a Kalman smoothing to $a'_{rg}(t)$ to estimate the gravity estimation error $g'_e(t)$. The idea is to think about $a'_{rg}(t)$ as $g'_e(t)$ plus noise, where noise is generated due to the act of typing. We set a large noise parameter in Kalman smoothing such that Kalman smoothing output does not closely follow $a'_{rg}(t)$, but it shows the underlying shifting trend hidden in $a'_{rg}(t)$. Now, we refine $a'_{rg}(t)$ to $a'_{rg}(t) - g'_e(t)$. The performance improves consistently.

Figure 12 plots the final result comparison between *MoLe* and original Android API method. It is clear that *MoLe* provides better watch displacement estimation.

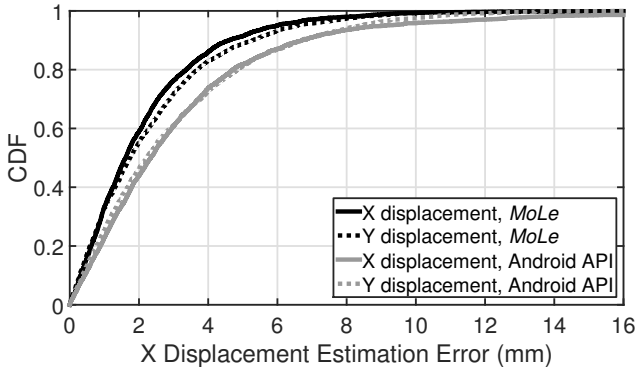


**Figure 12: Comparison of *MoLe* and Android API displacement result.**

## 4.2 Point Cloud Fitting

From the estimated displacements for each key, *MoLe* generates a *unlabeled point cloud (UPC)* for the attackee. Since the points in this UPC are not labeled, we fit the attacker's *character point cloud (CPC)* to the UPC. The key intuition is that the relative motions between keys (reflected in the relative locations of the point clouds) should bear similarity across all users. To achieve this fitting, we compute the convex hulls for the CPC and the UPC.

Observing that the fitting parameters for up and down hand displacements can be different, we compute 2 convex hulls for the CPC – one for all the positive $X$ displacements (denoted by $H^{CPC}_{pos}$) and the other for all the negative $X$ displacements (denoted by $H^{CPC}_{neg}$). Similarly, we compute 2 convex hulls for the UPC – $H^{UPC}_{pos}$ and $H^{UPC}_{neg}$. We fit $H^{CPC}_{pos}$ to $H^{UPC}_{pos}$ and $H^{CPC}_{neg}$ to $H^{UPC}_{neg}$ respectively.

To fit a convex hull $H_1$ to another convex hull $H_2$, we first calculate their cords $C_1$ and $C_2$ which originate from the origin and pass through their centroids. Then, we (1) rotate $H_1$ such that $C_1$ aligns with $C_2$, (2) scale $H_1$ in the direction of $C_1$ such that $C_1$ equals to $C_2$, (3) scale $H_1$ in the orthogonal direction of $C_1$ such that the area of $H_1$ equals to that of $H_2$. Figure 13 shows an example of point cloud fitting.

The metric for fitting is defined by the degree of overlap between the CPC and UPC's convex hulls. More precisely, we compute the ratio of the intersection and union of the convex hulls.

An attacker might be able to generate multiple CPCs, perhaps from her accomplices. *MoLe* performs the fitting for each of these CPCs and selects the one that maximizes the intersection/union ratio. The rotated and scaled CPC is now superimposed on the UPC and a framework is ready to estimate labels for each point in the UPC.

## 4.3 Bayesian Inference

Even if the keystroke detection and point cloud fitting are perfect, *MoLe* still does not know the characters typed by the right

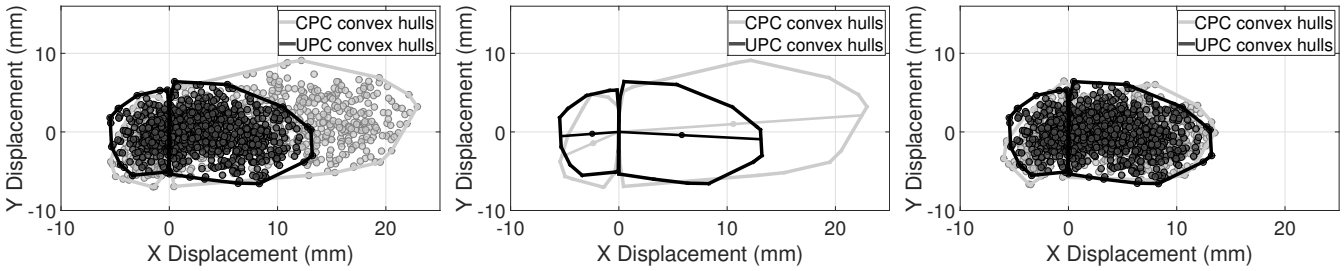


**Figure 13: Point Cloud Fitting. Black points are the CPC attacker template and gray points are UPC from attackee. (a) Finding each convex hull (b) Calculate the centroids and perform rotate and scale. (c) Point cloud fitting result.**

hand. Thus, as a first step, we attempt to fill in these "holes" to infer the complete word. The rather obvious step is to calculate the *posterior probability* of each word in the English dictionary, given the motion inferences from the left hand. The words corresponding to the top-K highest probabilities can be enumerated as candidates. We apply Bayes' theorem formulated as:

$$P(W \mid O) = \frac{P(O \mid W)P(W)}{P(O)} \tag{1}$$

where

- $W$ is a candidate word from the dictionary and $O$ is the observation motion data
- $P(W|O)$ is the posterior probability of the word given the observed motion data
- $P(O|W)$ is the likelihood function that estimates the probability of the word $W$ based on the observed motion data
- $P(W)$ is the prior probability which captures the word's occurrence frequency
- $P(O)$ is the probability of the observation

Since $P(O)$ is the same for all possible words, we are only interested in calculating $P(O|W)$ and $P(W)$. That is,

$$P(W \mid O) \quad \propto \quad P(O \mid W) \quad \times \quad P(W) \tag{2}$$

$P(W)$ can be obtained from a contemporary English corpus. In the current experiment, we assume $P(W)$ is equal among words, meaning each word has same occurrence frequency. The key goal translates to obtaining the maximum (or high) values of the likelihood $P(O|W)$. In the following, we present a few opportunities to refine the likelihood function and the posterior probability.

## Step I: Using the Number of Keystrokes

Our first intuition is simply to employ the number of detected keystrokes as observations to match the word. The keystroke detector gives us the number of keys typed by the left hand and this number is used to match each word. For example, when two keys detected, matching the word "the" produces higher likelihood than the word "teacher", because the number of peaks generated while typing "the" is much closer to 2 than "teacher". Now, for each of the detected keystroke, we would like to match them with the characters in the word. Since the keystroke could be caused by any characters in the word, we need to consider all possible assignments.

To calculate P(O | W), we can write

$$P(O \mid W) = P(N \mid W) = \sum_{(\alpha_1,\ldots,\alpha_N)} P((c_{\alpha_1},\ldots,c_{\alpha_N}) \mid W) \tag{3}$$

where $N$ is the number of keystrokes and $(\alpha_1,\ldots,\alpha_N)$ represents one possible $N$-element combination from $\{1,2,\ldots,L\}$ and L is the word length. The summation adds up all possible combinations. $c_i$ is the $i_{th}$ character in $W$; $P((c_{\alpha_1},\ldots,c_{\alpha_N}) \mid W)$ is probability that $N$ peaks are generated by $c_{\alpha_1},\ldots,c_{\alpha_N}$.

For instance, let's assume two keystrokes are detected and we want to calculate the likelihood of the word "the" by using the keystroke detector result in Figure 10.

$$\begin{aligned} P(O=2 \mid W=\text{"}the\text{"}) &= 0.986 * 0.035 * 0.119 \ (c_{\alpha_1}=t; c_{\alpha_2}=h) \\ &+ 0.986 * 0.965 * 0.881 \ (c_{\alpha_1}=t; c_{\alpha_2}=e) \\ &+ 0.014 * 0.035 * 0.881 \ (c_{\alpha_1}=h; c_{\alpha_2}=e) \\ &= 0.84 \end{aligned}$$

In above equation, 0.119 means the probability that "e" is not detected and equals to $(1-0.881)$. In a similar way, we calculate the probability that "t" is not detected (0.014) and "h" is detected (0.035).

Thus, by iterating over all words in the dictionary, we can obtain the likelihood for each word. Of course, further refinements are possible.

## Step II: Consecutive Characters

In some cases, our key-press timing module detects only one keystroke for two consecutive characters such as "er", "sa" or "re". These keys are adjacent on the keyboard and the watch dips in so close succession that they are not separable. Therefore, we treat these character pairs as one key. Figure 14 shows the experimental results that these common character–pairs are detected as zero, one, or two key-presses. Evidently, treating them as a single key-press should be appropriate in a majority of the cases.

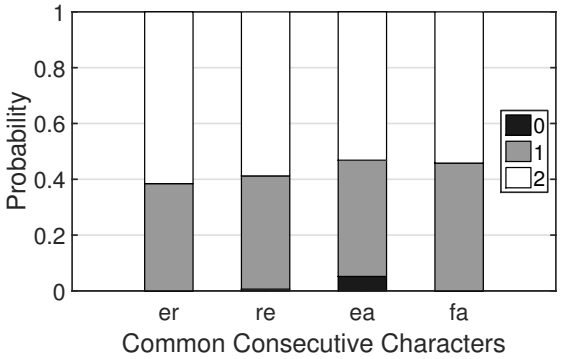


**Figure 14: The probability of number of keystroke detections for consecutive characters.**

## Step III: Adding Watch Displacement

*MoLe* is now ready to leverage the actual watch displacement. Assuming fingers are placed over the home position ("F" and "J"), recall that the key-press location estimation module computes the location of each key press (Figure 4). Of course, the estimated location is not accurate due to the noise in the hardware, minute differences in the hand motions, minute differences in hand's 2D orientation, etc. However, given that the CPC has been fitted to the user's UPC, it is now possible to better predict the word by taking displacement into consideration. Thus, equation 3 can be rewritten as:

$$\begin{aligned} &P(O \mid W) \\ &= P(N \cap d_i, i = 1,2,...,N \mid W) \\ &= \sum_{(\alpha_1,...,\alpha_N)} P((c_{\alpha_i},...,c_{\alpha_N}) \mid W) p((d_1,...,d_N) \mid (c_{\alpha_i},...,c_{\alpha_N}), W) \end{aligned} \tag{4}$$

where $p((d_1,...,d_N) \mid (c_{\alpha_i},...,c_{\alpha_N}), W)$ is probability density of typing $c_{\alpha_1},...,c_{\alpha_N}$ of W at character displacements $d_1,...,d_N$, and $d_i$ is the $i_{th}$ character displacement in $W$.

*MoLe* models each character's location as a Gaussian distribution. Assuming the distribution of displacement $d_i$ only depends on current character $c_{\alpha_i}$, we simplify Equation 4 as:

$$P(O \mid W) = \sum_{(\alpha_1,...,\alpha_N)} P((c_{\alpha_1},...,c_{\alpha_N}) \mid W) \prod_{i=1}^{N} p(d_i \mid c_{\alpha_i}) \tag{5}$$

where $p(d_i \mid c_{\alpha_i})$ is probability density of $d_i$ given character $c_{\alpha_i}$.

## Step IV: Character Transitions

We assumed above that each character displacement is independent. However, typing a word consists of sequential movements and the current displacement is indeed influenced by the location of the previous character. Figure 15 illustrates an example – we compare the displacement of "a" when previous char is "v" versus "r". Clearly, the distributions are different. For "ra", the displacement of typing "a" is shifted towards the position of character "r" because the little finger types "a" right after "r", before returning to home position. For "va", the displacement of "a" is clear close to "v", because of the same reason.

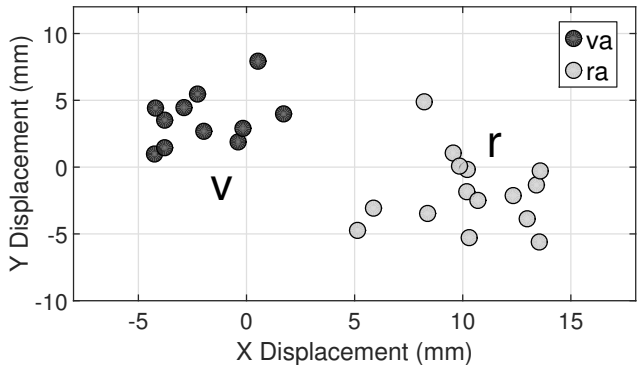


**Figure 15: Comparison of "a" displacement while previous character is "v" or "r". The key locations of v and r are marked in the figure.**

Given this observation, we extend the likelihood function to the following:

$$P(O \mid W) \approx \sum_{(\alpha_1,...,\alpha_N)} P((c_{\alpha_1},...,c_{\alpha_N}) \mid W) \prod_{i=1}^{N} p(d_i \mid c_{\alpha_i}, c_{\alpha_{i-1}}) \tag{6}$$

Now the displacement probability density $p(d_i \mid c_{\alpha_i}, c_{\alpha_{i-1}})$ not only consider $c_{\alpha_i}$ but also the previous char $c_{\alpha_{i-1}}$.

## Step V: Keystroke Interval

Timing of the key-presses on the left hand should also encode information about missing keys. We ask, given the time detected interval between consecutive keystrokes, what is the probability that there are $N$ right hand characters between them? Correct guesses of $N$ can obviously help. For example, when typing the word "t̲ h a̲ n k s̲" (characters typed on the left hand are underlined), the observed interval between "t" and "a" may be expectedly shorter than between "a" and "s".

Figure 16 plots the distribution of time intervals for increasing lengths of character sequences. These sequences consists of right hand characters in the middle and are surrounded by two left characters. Unsurprisingly, the time interval generally increases with the number of keystrokes. However, we observe high variance. For example, even though the segment "-b̲ i l i t̲-" and "-t̲ i o n a̲-" both have three right hand characters in the middle, the average interval of "-t̲ i o n a̲-" is shorter than "-b̲ i l i t̲-" due to hand geometry and typing familiarity.

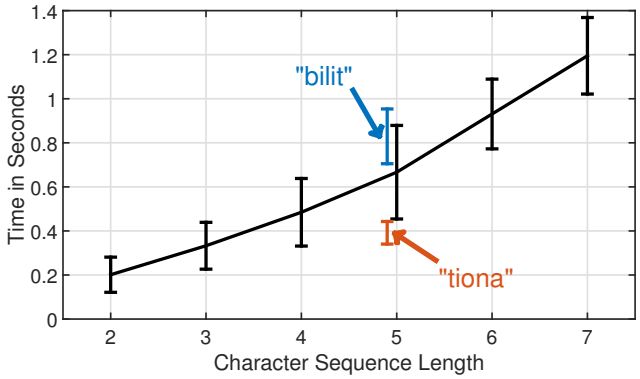


**Figure 16: Y-axis is the time interval between two detected keystrokes and X-axis is the number of sequence length.**

Therefore, for better timing observation, we should obtain the time interval distribution of every possible character-sequence that is preceded and followed by two left-hand characters, and use this distribution in the Bayesian model.
The observation can be written into

$$\begin{aligned} &P(O \mid W) \\ &= P(N \cap d_i, i = 1,2,...,N \cap t_j, j = 1,2,...,N-1 \mid W) \\ &\approx \sum_{(\alpha_1,...,\alpha_N)} P((c_{\alpha_1},...,c_{\alpha_N}) \mid W) \prod_{i=1}^{N} p(d_i \mid c_{\alpha_i}, c_{\alpha_{i-1}}) p((t_1,...,t_{N-1}) \\ &\mid (c_{\alpha_1},...,c_{\alpha_N}), (d_1,...,d_N), W) \end{aligned} \tag{7}$$

where $N-1$ interval distributions, $t_j$ with $j = 1,2,...,N-1$, are added into the observation. Note that the attacker and attackee are typing at slightly different speeds. *MoLe* compensates this speed bias with a factor $k$, calculated from the ratio of attacker to attackee's average typing interval.

# 5. EVALUATION

## 5.1 Data Collection and Methodology

*MoLe* has been implemented on the Galaxy Gear Live smart watch, which runs the latest Android Wear platform. When activated, the *MoLe* client on the watch continuously logs accelerometer and gyroscope readings at 200Hz, along with timestamps. The sensor data is stored locally during data collection and transferred to the backend (MATLAB) server for analysis.

*MoLe* is evaluated with 8 subjects, recruited by advertising about these experiments in the university campus. The subjects were offered an incentive of $10 per hour, and each subject invited to our lab for a 2 hour session. All subjects were familiar with English typing (5 of them are native English speakers, 3 of them were females). Each subject was asked to type 300 English words randomly selected from 5000 most frequently used words [7]. The word-length ranged from 1 to 14, and was equally distributed. In total, we test 2400 words across all users.

Each subject was seated at a desk in front of a Lenovo laptop. Our experiment GUI popped up one word at a time on the laptop screen – the volunteer's task was to type the same word in a text box on the screen. If any of the character is incorrectly typed, we discard the data and let the subject re-enter the word. Between each word recording, we ask the subject to initialize their hand position on "F" and "J". During the typing, the laptop was also programmed to record the timing of the keystroke – this will later serve as ground truth. To collect the offline training data, two of the authors (pretending to be attackers) performed the same procedure, but with Top-500 longest words in the dictionary. Long words help capture the diversity of the typing patterns. The whole data collection is done a Lenovo ThinkPad equipped with a regular full-sized keyboard.

For full ground truth recording, we mount an Android Samsung Galaxy S4 phone on top of the keyboard and use the front camera to capture the video of hand movement. We apply the camera calibration toolbox in Matlab [9] to calibrate the camera pixel, and measure the watch distance and location from each frame (Figure 1).

## 5.2 Performance Results

The following questions are of interest in this section:

- How well can *MoLe* guess each word (i.e., in an ordered list of guesses by *MoLe*, what is the rank of the actual word?)
- What factors affect this rank?
- How different opportunities contribute towards overall performance.
- How can we prevent the threat introduced by *MoLe*?
- Does keyboard matter?
- Can humans guess better by looking at a sequence of candidate guesses?

### (1) How well can *MoLe* guess each word?

Figure 17 plots the CDF of rank, computed from each of the 2400 words typed by the subjects of the experiments. The average across all 8 subjects is shown in black, while *MoLe*'s performance for each individual subject shown in gray. We observe that the median rank of a word is 24, while for 30 percentile, the rank is 5. Put differently, when a user types a word, there is a 30% chance that *MoLe* would narrow down the typed word to only 5 possibilities, and a 50% chance to only 24 possibilities. Given that 5000 words are possible, this is an appreciable reduction of the search space, and makes it amenable to brute-force attacks.

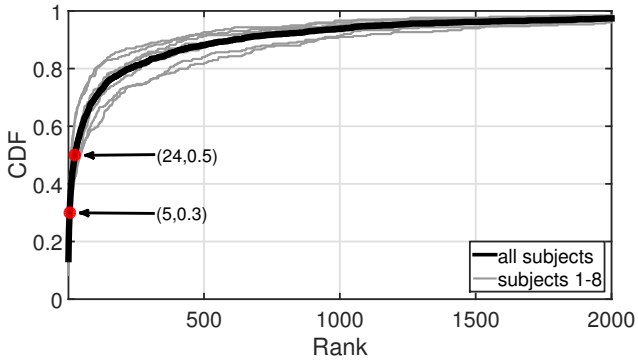


**Figure 17: CDF of rank computed for each of the 2400 words typed by 8 subjects.**

Figure 18 plots the ranks of typed words for each test subject. The ranks are generally higher (i.e., *MoLe*'s guesses are better) for subjects S2, S5, and S8. On examining the video and sensor traces for these subjects, we find that they have lower variance in their hand movements, probably because they type as per the prescribed guidelines. We also test the case with perfect key-press detection (i.e., using the actual number and timing of keystrokes only from the left hand, gathered from the ground truth timing information recorded during the experiments). Surprisingly, the $30^{th}$ percentile drops sharply to 1, meaning that *MoLe* can exactly guess the word; the $50^{th}$ percentile drops to 6. Clearly, further improvements in key-press detection is the key to improving *MoLe*.

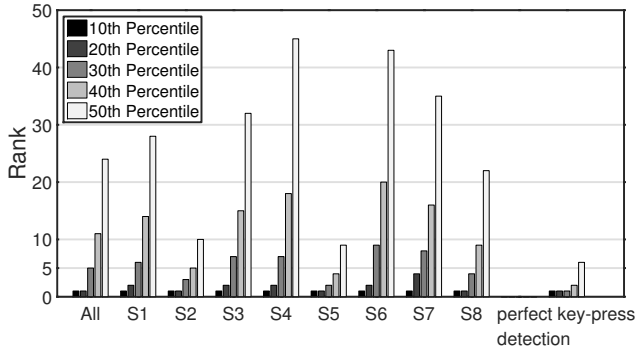


**Figure 18: Rank of average, across users and with perfect key-press detection**

### (2) What factors affect the rank?

Figure 19 plots the median rank of words for increasing word-lengths. The rank generally decreases with word length greater than 6, primarily because (1) there are greater number of keystrokes that get detected in a longer word, and (2) because the number of words of that length reduces, in turn reducing the number of words to be confused with. Words of length 4 – 7 on the other hand, have less keystrokes; also, there are many more words of such lengths, adding to the difficulty of detection.

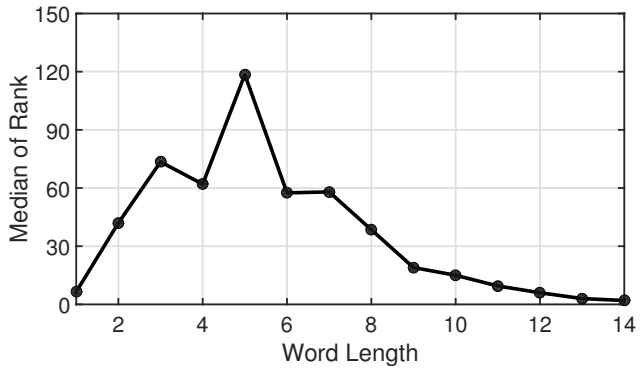


**Figure 19: Median rank plotted against increasing word length – 4 – 7 length words show worse performance due to fewer keys to be detected while such words occur in large numbers.**

Figure 20 shows the number of left hand characters in a word. With increasing number of left characters, *MoLe* naturally gains richer information about the word, ultimately improving its ability to guess. When a word contains more than 5 left characters, *MoLe* is able to bring down the rank below 20. When left character are 2 to 4, performance degrades because a large number of words have the same 2 – 4 left hand characters in them.


**Figure 20: Variation of rank against the number of characters typed by the left hand.**

## (3) Impact of Each Bayesian Opportunity

Section 4.3 leveraged a number of opportunities under the Bayesian model. Figure 21 shows the break-down of contributions from each of them. Using only the number of detected keystrokes (step 1 and 2 in section 4.3), *MoLe* performs rather poorly on the dataset. When the displacement information is added (step 3 and 4), *MoLe* improves the rank from 340 to 49 at $50^{th}$ percentile. Finally, when time interval is incorporated (step 5), the median rank improves from 49 to 24.

## (4) Impact of Sampling Rate

Figure 22 characterizes the impact of sensor sampling rate (200Hz, 100Hz, 50Hz, 20Hz) on the median rank of words for all of the subjects. Evidently, *MoLe*'s ability to guess degrades drastically with lower sampling rates – median ranking falls as 64, 141 and 1218. Perhaps this could be a way to mitigate the attack through smart-watches. A typing classifier could first

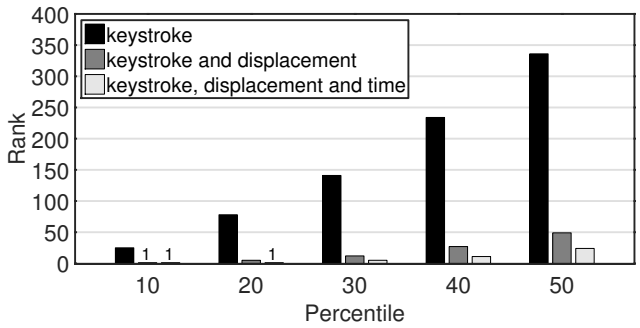


**Figure 21: Contribution of different opportunities towards the overall performance of *MoLe*.**

detect whether the user is typing, and if so, the sampling rate of the sensor data can be diminished to less than 50Hz.

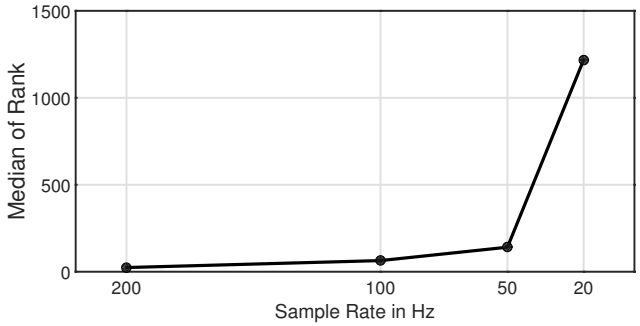


**Figure 22: Lower sensor sampling rate rapidly reduces the ability to guess the word, perhaps indicating a way to thwart *MoLe*'s attack.**

## (5) Keyboard Variant

The difference in the shape of desktop and laptop keyboards may translate to decoding errors with *MoLe*. To test this, subject S5 was asked to repeat the same data collection process on a desktop keyboard. Figure 23 plots a histogram of the rank differences of each word, when decoded from the two keyboards. We notice that 45.2% of rank differences are less than 10. More specifically, the laptop keyboard presents a median rank of 10 and 30 percentile rank of 4. The computer keyboard's median rank is 14 and 30 percentile rank is still at 4. The results show that the system performance is close between two keyboards, *even though the attackers used the laptop keyboard for training the system.* If the attackers generated models from various keyboards, and applied the best one during the keystroke detector and cloud fitting process, the results can be even better.

## (6) Recovery via Human Observation

Although *MoLe* is not able to detect spaces and separate the words at this moment, we are interested to know how the threat would become even worse if this limitation is relaxed. To this end, we ask subject *S*5 to enter (one-by-one) the words from an actual sentence. Table 1 shows *MoLe*'s end-to-end prediction result for each of the words in the sentence (which contains 8 words). For each word, the *Top-5* guesses are listed from top to down. As would be expected, the words in each column bear similarity in the character sequences embedded in them. For example, $W_6$ typically starts with "t" or "th" and $W_8$ contains many left hand characters. We present this table to colleagues in

| Rank | $W_1$ | $W_2$ | $W_3$ | $W_4$ | $W_5$ | $W_6$ | $W_7$ | $W_8$ |
|---|---|---|---|---|---|---|---|---|
| 1. | motor | pistol | profound | technology | angel | those | that | disappear |
| 2. | monitor | list | journalism | remaining | spray | today | tight | discourse |
| 3. | them | but | originally | telephone | super | third | tightly | secondary |
| 4. | the | lost | original | meanwhile | fire | through | thirty | adviser |
| 5. | then | most | profile | headline | shore | towel | truth | discover |

**Table 1: *Can you guess the correct sentence?* The words in each column are ranked in decreasing order of probability; also note that some words may not feature in the $top-5$ words presented in each column. The answer is made available at end of the paper[2].**

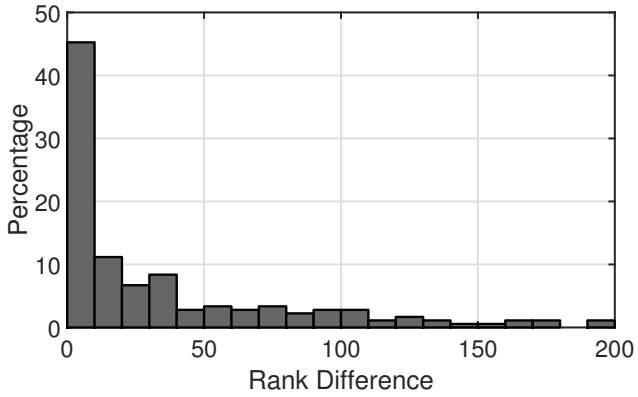


**Figure 23: A histogram shows the rank difference of each word between 2 keyboards. The trend indicates that most words have similar rank, so the difference is low.**

our department and most of them could recover the sentence in a few minutes. We encourage the readers to reconstruct sentence on their own – the answer is made available at end of the paper[2].

## 6. POINTS OF DISCUSSION

We discuss a few limitations and opportunities for improvement.

**(1) Confined to separate words.** *MoLe* is not yet a real-world attack since it is not able to detect the space bar and separate out words from a sentence. Also, without any priors on digit sequences, it is difficult to detect what digits users are typing. Additional work is necessary to further separate out these keystrokes. However, we believe there is opportunity. We have found early evidence that the magnetic field on the keyboard is quite telling of the position of the wrist. With some signal processing, the magnetic field may offer valuable hints on how the wrist is moving and when it is coming back to its original position. We leave this to future work.

**(2) Applying nature language processing.** To recover the whole sentence, techniques from nature language processing (NLP) may also apply. For example, we can apply N-gram language models which predict the $N^{th}$ word given a previous $N-1$ word sequence. Thus, even if a few words have low accuracy in the sentence, it may still be possible to infer the sentence, or even the broad semantic content.

**(3) Typing activity classifier.** In a real world attack, we would first need to subject the sensor data to a classifier which will output whether the user is typing or not. Only when the user is known to be typing, should *MoLe* be applied on the data. We have not developed a "typing or not" detector in this paper, but believe that it can be developed (perhaps from the orientation of the watch and the slight back and forth movement). Also, we need to be able to identify whether the watch is worn on the left or right hand, left to future work.

## 7. RELATED WORK

We categorize this section into inferring keystrokes on traditional computer keyboards and sensor information leaks on smart devices.

**Inferring keystrokes on computer keyboards**
Many researches have attempted to infer keystrokes on computer keyboards. Various modalities have been leveraged, such as acoustics signal [10, 11], input timing analysis [12–14], RF radio [15], and electromagnetic emanations [16]. These researches have successfully delivered high accuracy results. However, none of them use motion sensor data; also, to intercept physical signals and decode the data, these methods are typically required to install additional hardware or software, which make them somewhat difficult to widely deploy the attacks. In contrast, *MoLe* can be launched with ease on top of commericially available wearable devices. Marquardt et al. demonstrated the (Sp)iPhone [17] and show that it is possible to use the accelerometer on iPhone to recover text entered on a keyboard when the phone is placed nearby, on the same table surface. Both (Sp)iPhone and *MoLe* exploit side channels, however, *MoLe* uses the wrist motion data in 3D, different from the surface vibrations in (Sp)iPhone.

**Sensor Information leaks on smart devices**
Researchers have studied side channel attacks to infer keystrokes on smartphones and tablets [18] [19] [20] [21]. The core idea behind these works is that when typing on different locations on a virtual keyboard, the keystrokes cause distinct vibrations/rotations. The motion data on smartphones can thus be used to infer the tapped location. TouchLogger [19] and accessory [18] are the early works, and use accelerometer only to infer tap location on screen. Cai et al. [20] and Miluzzo et al. [21] advance the technique to infer keystrokes and show that gyroscope has better accuracy than accelerometer based inference. *MoLe* bears similarity in that it is also a side channel attack. However, there are two main differences. Firstly, above works are feature-based. They design features to capture distinct screen motions caused by the touches and train models with these features to infer the touch positions. In contrast, since smartwatch is on the user's wrist, we track the movement of the wrist to infer what the user has typed. Secondly, we are only able to sense partial keystrokes with one hand, as well as indirect data of the wrist. To recover the whole input word, *MoLe* must rely much more strongly on the Bayesian models. GyroPhone [22] presents a new type of threat to intercept human speech by using gyroscope on smartphone. The authors found that the MEMS gyro sensors are able to pick up air vibrations from sound at

low frequency. *MoLe* is still different since it attempts to extract semantic understanding of the human's hand motions.

# 8. CONCLUSION

This paper demonstrates that sensor data from smart watches can leak information about what the user is typing on a regular (laptop or desktop) keyboard. By processing the accelerometer and gyroscope signals, tracking the wrist micro-motions, and combining them with the structure of valid English words, reasonable guesses can be made about typed words. Given the excitement around a smart-watch app store, such an attack can be severely penetrating into the private lives of humans. While we find that diminishing the sampling rate of the accelerometer and gyroscope can alleviate the attack, we believe additional side channels like magnetic field variations need to be carefully investigated. Otherwise, wearable devices could soon become a "double edged sword" slowing down future innovations on this platform.

# 9. ACKNOWLEDGMENTS

We sincerely thank our shepherd and the anonymous reviewers for their valuable comments and suggestions. This research is partially funded by NSF research grant CNS-1430064, and by a fellowship by the Ministry of Science and Technology of Taiwan. We are also grateful to Google, Qualcomm, Samsung and Huawei for partially funding this research.

# 10. REFERENCES

[1] Sandip Agrawal, Ionut Constandache, Shravan Gaonkar, Romit Roy Choudhury, Kevin Caves, and Frank DeRuyter, "Using mobile phones to write in air," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*. 2011, MobiSys '11, pp. 15–28, ACM.

[2] Shahriar Nirjon, Jeremy Gummeson, Dan Gelb, and Kyu-Han Kim, "TypingRing: A wearable ring platform for text input," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. 2015, MobiSys '15, pp. 227–239, ACM.

[3] Jiayang Liu, Zhen Wang, Lin Zhong, J. Wickramasuriya, and V. Vasudevan, "uWave: Accelerometer-based personalized gesture recognition and its applications," in *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on*, March 2009, pp. 1–9.

[4] Chao Xu, Parth H. Pathak, and Prasant Mohapatra, "Finger-writing with smartwatch: A case for finger and hand gesture recognition using smartwatch," in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*. 2015, HotMobile '15, pp. 9–14, ACM.

[5] Abhinav Parate, Meng-Chieh Chiu, Chaniel Chadowitz, Deepak Ganesan, and Evangelos Kalogerakis, "Risq: Recognizing smoking gestures with inertial sensors on a wristband," in *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*. 2014, MobiSys '14, pp. 149–161, ACM.

[6] Sangki Yun, Yi-Chao Chen, and Lili Qiu, "Turning a mobile device into a mouse in the air," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. 2015, MobiSys '15, pp. 15–29, ACM.

[7] "Word Frequency Data Set," `http://www.wordfrequency.info/`.

[8] "MATLAB Peak Analysis Library," `http://www.mathworks.com/help/signal/examples/peak-analysis.html`.

[9] "Camera Calibration Toolbox for Matlab," `http://www.vision.caltech.edu/bouguetj/calib_doc/`.

[10] Li Zhuang, Feng Zhou, and J. D. Tygar, "Keyboard acoustic emanations revisited," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 1, pp. 3:1–3:26, Nov. 2009.

[11] Dmitri Asonov and Rakesh Agrawal, "Keyboard acoustic emanations," in *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, May 2004, pp. 3–11.

[12] Denis Foo Kune and Yongdae Kim, "Timing attacks on pin input devices," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*. 2010, CCS '10, pp. 678–680, ACM.

[13] Dawn Xiaodong Song, David Wagner, and Xuqing Tian, "Timing analysis of keystrokes and timing attacks on SSH," in *Proceedings of the 10th Conference on USENIX Security Symposium - Volume 10*. 2001, SSYM'01, USENIX Association.

[14] Kevin S. Killourhy and Roy A. Maxion, "Comparing anomaly-detection algorithms for keystroke dynamics," in *Dependable Systems Networks, 2009. DSN '09. IEEE/IFIP International Conference on*, June 2009, pp. 125–134.

[15] "Key Sweeper," `http://samy.pl/keysweeper/`.

[16] Martin Vuagnoux and Sylvain Pasini, "Compromising electromagnetic emanations of wired and wireless keyboards," in *Proceedings of the 18th Conference on USENIX Security Symposium*. 2009, SSYM'09, pp. 1–16, USENIX Association.

[17] Philip Marquardt, Arunabh Verma, Henry Carter, and Patrick Traynor, "(sp)iphone: Decoding vibrations from nearby keyboards using mobile phone accelerometers," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*. 2011, CCS '11, pp. 551–562, ACM.

[18] Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, and Joy Zhang, "Accessory: Password inference using accelerometers on smartphones," in *Proceedings of the Twelfth Workshop on Mobile Computing Systems and Applications*. 2012, HotMobile '12, pp. 9:1–9:6, ACM.

[19] Liang Cai and Hao Chen, "TouchLogger: Inferring keystrokes on touch screen from smartphone motion," in *Proceedings of the 6th USENIX Conference on Hot Topics in Security*. 2011, HotSec'11, pp. 9–9, USENIX Association.

[20] Liang Cai and Hao Chen, "On the practicality of motion based keystroke inference attack," in *Proceedings of the 5th International Conference on Trust and Trustworthy Computing*. 2012, TRUST'12, pp. 273–290, Springer-Verlag.

[21] Emiliano Miluzzo, Alexander Varshavsky, Suhrid Balakrishnan, and Romit Roy Choudhury, "Tapprints: Your finger taps have fingerprints," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*. 2012, MobiSys '12, pp. 323–336, ACM.

[22] Yan Michalevsky, Dan Boneh, and Gabi Nakibly, "Gyrophone: Recognizing speech from gyroscope signals," in *23rd USENIX Security Symposium (USENIX Security 14)*. Aug. 2014, pp. 1053–1067, USENIX Association.

[2] "The most profound technologies are those that disappear" - Mark Weiser, 1991