Permission-free Keylogging through Touch Events Eavesdropping on Mobile Devices

Luca Bedogni, Andrea Alcaras, Luciano Bononi
Department of Computer Science and Engineering, University of Bologna, Italy
{luca.bedogni4,andrea.alcaras,luciano.bononi}@unibo.it

***Abstract*—Mobile devices are carried by many individuals in the world, which use them to communicate with friends, browse the web, and use different applications depending on their objectives. Normally the devices are equipped with integrated sensors such as accelerometers and magnetometers, through which application developers can obtain the inertial values of the dynamics of the device, and infer different behaviors about what the user is performing. As users type on the touch keyboard with one hand, they also tilt the smartphone to reach the area to be pressed. In this paper, we show that using these zero-permissions sensors it is possible to obtain the area pressed by the user with more than 80% of accuracy in some scenarios. Moreover, correlating subsequent areas related to keyboard keys together, it is also possible to determine the words typed by the user, even for long words. This would help understanding what user are doing, though raising privacy concerns.**

I. Introduction

Mobile devices are carried by many individuals, which rely on them to access personalized services, work and communicate with friends and relatives. By tilting the device and interacting with it, applications can annotate user data, and personalize services for a more optimized user experience. However, the possibility to automatically infer what the user is doing or typing also raises privacy concerns. The majority of mobile devices are enabled with integrated sensors such as accelerometers, which report values useful to determine the dynamics of the device. The whole paradigm of Context Aware Computing builds on the ability to understand what the user is doing, or where she is, made possible by the use of the integrated sensors. This enables applications such as activity recognition and transportation mode detection [1] [2], which enable personalized services tailored to the current context of the user, and empowers developer to automatically label different activities and contexts. For instance, analyzing the tilting of the device may lead to understand where the user is clicking. On the one hand this could help in understanding how the user types, which may also lead to understand whether the user is in a hurry or relaxed. On the other hand, they may also be exploited to understand private information about the user, such as words typed, hence sentences.

Our aim is to show that the different tilting positions of the smartphones can lead to understand which area the user is pressing on the screen. Clearly this information can then be used to annotate meaningful user data, such as typing speed, frequency, and particular words. However, as these sensors can be currently used without requiring any permission on the Android system, a malicious application knowing the layout of the applications in use may actually understand what items the user is clicking. This problem is particularly important in case of applications which handle sensitive data, or those requiring passwords and secret information in order to access private information about the user. Clearly, this task is inherently challenging, as there are a number of variables which affect the tilting position, not only related to the touched area, such as the position in which the user is while typing. Moreover, recognizing bigger areas is certainly more feasible, as tilting positions differ more among each other. For this reason, we will perform our evaluation accounting for two possible scenarios, one in which the initial orientation of the device is known a priori, and the other one in which it has to be derived by analyzing the sensors values. However, many applications use a virtualized keyboard which typically appears on the lower part of the screen, through which the users can click and type words, passwords and numbers. Clearly, recognizing the areas which the user touches would also enable applications to eavesdrop the words the user is typing, potentially stealing private information.

In this work we present two novel contributions: at first, we develop a model which enables a mobile application to recognize the areas which the user is touching on the screen, by leveraging on the tilting perceived by the sensors and comparing it with previous measurements. We then focus our analysis on the recognition of areas related to the keyboard, which appears on the lower part of the touchscreen. By recognizing single touch events and building a tree of possibilities, also accounting for neighbor areas, and correlating subsequent recognitions, we show that it is possible to obtain an accuracy of more than 80% for short words, while longer words can be recognized with almost perfect accuracy. The rationale is that longer words, which benefit from longer sequences of touch events, are easier to be recognized than shorter words which instead are more prone to inaccurate touch event recognitions.

The rest of this paper is organized as follows: Section II discusses related works from literature, particularly regarding systems which exploit wearables and mobile devices to eavesdrop information about the user; Section III presents our framework to recognize touch events on smartphones, based on integrated sensors; Section IV correlates together results from Section III to eventually understand typed words, and evaluates its performance; Section V concludes this paper and presents future work on this topic.

978-1-5386-9151-9/19/$31.00 ©2019 IEEE

II. Related work

Since their introduction in smartphone and tablets, sensors such as accelerometers, magnetometers and gyroscopes have been used to understand and model the environment in which the mobile device was currently used, to eventually provide more appropriate content and to perform automatically specific actions. This has been the case of activity recognition [1], through which mobile devices can understand what the user is currently doing, such as walking, running, or how she is moving, through techniques known as transportation mode detection [2]. Another field is that related to E-health [6], where devices are leveraged to monitor patients over time, or used to acquire precise knowledge of their dynamics. Many works which aim to acquire a better understanding of the mobile device current context typically leverage on machine learning models, which are trained to recognize future similar events [2]. However, as we already stated, acquiring information about the patterns of the user may also provide knowledge on user's private and sensitive information, such as typical visited places, hours in which she performs specific activities and so on and so forth. Closer to the topic of this study is the possibility to eavesdrop information on the areas the user is interacting with on the smartphone. More recently, with the advent of wearable devices, studies have been also performed on the user's interaction on the smartphone by exploiting data obtained through similar sensors installed on the wearable device.

We can classify studies focusing on testing the privacy boundaries of users in three different classes: *(i)* identifying the user among a set of human beings; *(ii)* forecasting future actions of the user; *(iii)* understand how the user is interacting with the device.

Understanding the specific user can be performed through a multitude of different techniques. For instance, by analyzing how users walk, [7] shows that it is indeed possible to identify a specific human beings, by analyzing different user patterns, which are distinct among each one according to the step length, speed and frequency. While on the one hand this may enable more automated authentication procedures or tools to identify outlaws, it may also expose the user to being identified even when not requesting it, thus raising privacy issues.

Forecasting future actions, locations and relationships the user may have in the future has been studied in the past to improve resource allocation and to deliver more appropriate content, such as giving in advance maps and information related to the future venues. This falls under the general umbrella of Anticipatory Mobile Computing [12], which predicts future user activities by analyzing past actions. However, it has been shown that providing apparently harmless information on the internet may lead to expose private information such as home location [3] [4] [8].

To identify what the user is doing on a mobile device, there are different studies which tackle the problem of understanding movement patterns extracted by the data coming from wearable devices worn by the user [9] [10] [13] [16] . In [17] the authors use instead the audio traces collected by mobile devices to understand what the user is writing. Specifically, [16] uses a malicious app installed on a user worn smartwatch to infer handwritten words.

Interesting the study presented in [5], where the authors show that using accelerometers and vibration patterns produced by smartphones, private communication can be achieved for two close devices, which has a similar aim as [14], in which the focus is to exchange cryptography keys securely.

When relying on smartphones accelerometers, we have discussed how these sensors can be hacked to collect private user data. The work presented in [15] shows that by carefully introducing acoustic noise, it is possible to achieve certain levels of uncertainty in the accelerometer measurements, potentially reducing privacy problems raised by their malicious use. Closer to the work of this paper is that presented in [11], where authors use accelerometers to understand users passwords. However, we focus on longer sentences rather than just passwords, and we also show that by using more sensors such as the magnetometer improves the overall accuracy.

III. Touch event recognition

In this section we describe the model we developed to recognize touch events on smartphones. We assume users perform touch events on a smartphone using one hand. As the user types in different areas of the screen, also the tilt of the smartphone will change. Our aim is to read data coming from the sensors, and recognize different orientations, so they can be mapped to specific areas of the screen.

In Section III-A we describe the mobile application used to gather the data and to provide performance evaluation of our proposed system. Section III-B presents the performance evaluation of the touch event recognition.

A. Mobile application

To obtain the data and test the effectiveness of our proposal, we developed an Android application capable of recording the raw measurements of the accelerometer and the magnetometer, and the touch events performed on the screen, based on different number of areas available on the screen, hence of variable sizes. In particular, one of the sizes which we investigated resembles the normal key size of the smartphone keyboard, paving the way to the analysis we will present in Section IV.

Data is recorded with a frequency rate of 10 Hz, and stored with the following structure:

$$\Theta(t) =< t, a_{x,y,z}, m_{x,y,z}, \Omega >,$$

where t is the time, $a_{x,y,z}$ are the values corresponding to the 3 axis of the accelerometer, while $m_{x,y,z}$ store the same for the magnetometer. Finally, Ω reports the area pressed on the screen, if any.

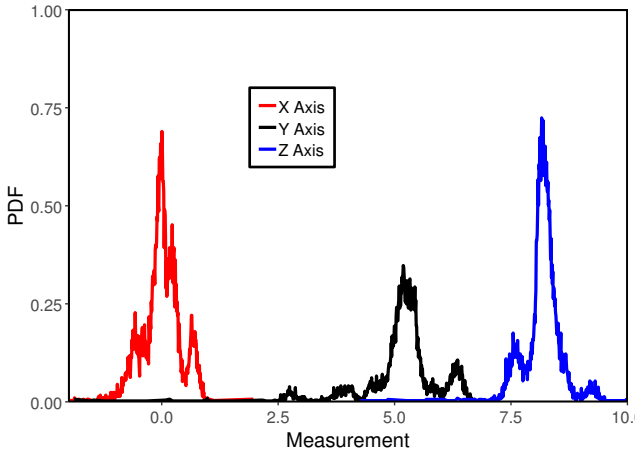


Fig. 1: Probability Density Function of the raw Accelerometer readings. Figure best viewed in colors.

Base tilting: We test two different scenarios, one where the base tilting of the smartphone is known a priori, and the other one in which it is not known. The former refers to the case in which the user keeps the smartphone in position before starting to touch the screen, while the former is modeled for scenarios in which the user writes on the touchscreen quickly, such as the case for instant messaging applications. For the first case, the base tilting of the devices is easily determined by analyzing the orientation of the device, obtained through the internal accelerometer, and computing the average values while the device is still.

The second scenario is instead more challenging, as different orientations obviously produce different tilting when touching areas on the screen. To understand the base tilting, we firstly assume that the base tilting of the device does not change during the recognition phase. The general idea is that even when moving the device to reach different areas of the screen to touch them, the smartphone rotates on a virtual axis, which is the base tilting. To recognize it, we leverage on the density values of the raw readings, as shown in Figure 1, where we plot the raw values of the accelerometer, which clearly show 3 different peaks. Let $\mathcal{P}_x^A$, $\mathcal{P}_y^A$ and $\mathcal{P}_z^A$ be the raw values of the X axis, Y axis and Z axis peak for the accelerometer, respectively, and $\mathcal{P}_x^M$, $\mathcal{P}_y^M$ and $\mathcal{P}_z^M$ those for the magnetometer. Having estimated the base tilting, we change the raw values of the two sensors as:

$$\Theta'(t) =< t, a_{x,y,z} - \mathcal{P}^A_{x,y,z}, m_{x,y,z} - \mathcal{P}^M_{x,y,z}, \Omega > .$$

In other words, when we do not know a priori the tilting of the device, we focus on the differences of tilting rather than on the absolute value itself. This enables comparing also different position of the user together.

Models: We build two different models, depending on the available sensors on the smartphone. While the accelerometer is available in almost every model on the market, the magnetometer is not. Hence, we build $\mathcal{M}^A$ which only accounts for measurements obtained from the accelerometer, and $\mathcal{M}^{A,M}$, which instead uses both the accelerometer and the magnetometer. As classifier, we used a Random Forest algorithm for both models, though with a different set of initial features.

B. Touch event results

In this section we present the results related to the touch event recognition. We test our proposals accounting for different scenarios, and with different numbers of areas to recognize. All the tests are performed on a ASUS Zenfone Max 2.

When users type, the base orientation of the device may change depending on their position, hence also the values of the internal sensors would give different values depending on it. We then evaluated two different scenarios, one in which we perform tests on a single run, with the user typing standing in the same position. The second test instead involves the user typing, then moving and then typing again, so that she will slightly change the base tilting of the device. Clearly, the second case is far more challenging than the first one, as similar tilts may refer to different areas, since the initial position of the user is also different.

Figure 2a shows the accuracy results when considering the scenario in which the position of the user does not change through the tests, while Figure 2b refers to the scenario in which the users records 5 minutes of data, then moves, then records again, hence changing her typing position. Both figures show the red bar referring to the use of $\mathcal{M}^{A,M}$, while the black bar refers to $\mathcal{M}^A$. Clearly, using also the magnetometer improves the results, regardless of the amount of areas to be recognized, as it may also give indication on whether the user is rotating the smartphone and how much.

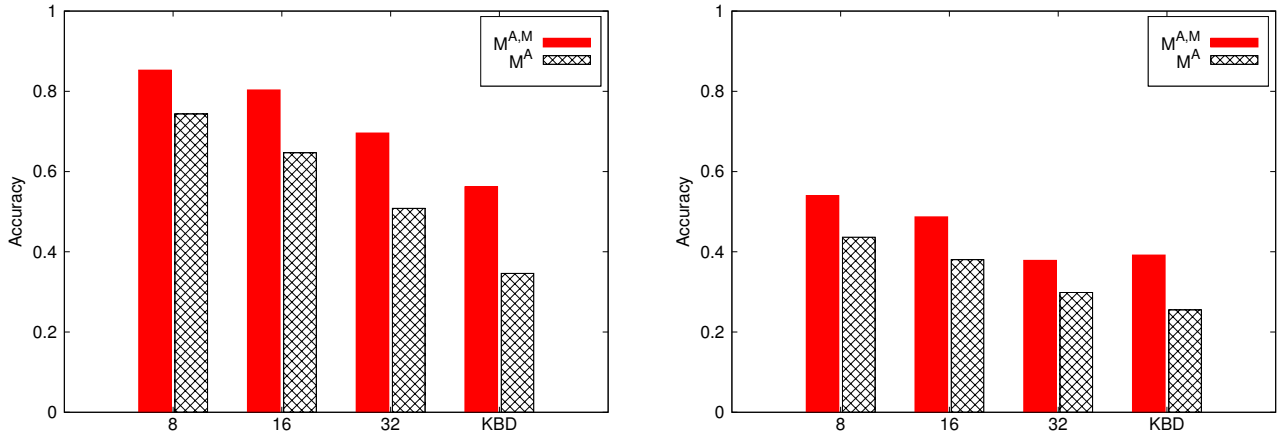
Naturally the accuracy results of Figure 2a are higher than those of Figure 2b, as the base tilt of the device does not change through the tests, hence making touch events on the same areas more similar among them, thus easier to be recognized. We present, for both scenarios, results for different area size, specifically for 8, 16 and 32 areas evenly distributed among the whole screen. Moreover, we also present a keyboard size area recognition test named `KBD`, which refers to screen areas placed in correspondence to keyboard keys, and with the same size. Compared to the other tests, the areas of `KBD` are 4 times smaller than the 32 areas size. For both scenarios, increasing the number of areas to be recognized obviously reduces the accuracy, as areas are smaller hence the differences between them are lower.

IV. Word Recognition

We perform word recognition by correlating subsequent touch events and building a tree, which provide the probability of a given word to be classified. To improve the overall accuracy of the word recognition phase, we leverage the contents of an English dictionary of more than 470.000 words, through which we analyze whether a sequence of characters may lead to a meaningful word or not.

A. Model

In this section we describe how we build the model which recognizes the words typed, by recognizing the touch events



(a) User position is not changing

(b) User is changing position

Fig. 2: Touched area recognition test for the two scenario. Figure 2a refers to the case in which the user is not moving between tests; Figure 2b shows the same analysis, though with the user which changes position, also the tilting of the mobile device.

Fig. 3: Example with $\mathcal{C}_i = a$ and $\mathcal{A}(\mathcal{C}_i) = \{q, w, s, z\}$.

on the keyboard. For any given character $\mathcal{C}_i$ at the i-th position of a sequence of characters Λ we define $\mathcal{A}(\mathcal{C}_i)$, which is the list of adjacent characters of $\mathcal{C}_i$ in a typical qwerty keyboard of a smartphone. We do this to assume a certain degree of uncertainty in the recognition, hence when our system recognizes $\mathcal{C}_i$ as the touched area, we also give a probability to the other adjacent characters in $\mathcal{A}(\mathcal{C}_i)$ of being touched. More formally, after identifying the area touched by the user, hence character $\mathcal{C}_i$, we define the probability $p(\mathcal{C}_i)$ as:

$$p(\mathcal{C}_i) = 1 - (|\mathcal{A}(\mathcal{C}_i)|) \cdot p_c(\mathcal{A}(\mathcal{C}_i)),$$

where $p_c(\mathcal{A}(\mathcal{C}_i))$ is the probability of character $c \in \mathcal{A}(\mathcal{C})$, and $|\mathcal{A}(\mathcal{C}_i)|$ is the total number of adjacent characters. Hence, $p_c(\mathcal{A}(\mathcal{C}_i))$ is defined as:

$$p_c(\mathcal{A}(\mathcal{C}_i)) = \frac{1}{|\mathcal{A}(\mathcal{C}_i)| + 2}.$$

Basically, the area identified by the system presented in Section III is considered with a higher probability $(p(\mathcal{C}_i))$ than that of every other character in its adjacency list $(p_c(\mathcal{A}(\mathcal{C}_i)))$. This is to assume possible uncertainty in the recognition, and to consider as if closer areas may have been clicked instead of the identified one. In other words, it is similar to have a lower number of areas to be discovered, making them bigger hence easier to be recognized, as Figure 2a and Figure 2b show. An example of this behavior is shown in Figure 3, where the area of the character A is identified, hence all the adjacent characters are also considered, though with a lower probability.

We then correlate different characters $\mathcal{C}_i$ identified, along with their adjacency list $\mathcal{A}(\mathcal{C}_i)$, in a sequence Λ. We then build a tree $\mathcal{T}_\Lambda$ in which level i is composed with the nodes $\mathcal{C}_i$ and all that in $\mathcal{A}(\mathcal{C}_i)$. The transition probability from any node in level $i-1$ to $\mathcal{C}_i$ is $p(\mathcal{C}_i)$, while for any given character c at the same level of the tree is for all of them $p_c(\mathcal{A}(\mathcal{C}_i))$. Once the tree is built, we select the word with the highest probability, accounting for words which do not exist in the dictionary. Basically, we visit $\mathcal{T}_\Lambda$ and remove each node n, and hence all its subtree, if no words exists from the beginning up to n.

Figure 4 shows an example of $\mathcal{T}_\Lambda$ for the word *app*, though For sake of readability we select a rather short word.

As it is possible to see, initially recognizing the character a also puts at the same level in the tree the characters w, z, s and q (i.e. $\mathcal{A}(a) = \{w, z, s, q\}$). It is then possible to see that all the subsequent characters are taken from the following character which is p and all its adjacency list $\mathcal{A}(p) = \{o, l\}$. Finally, we add the last character p once more and $\mathcal{A}(p) = \{o, l\}$. At each step we remove the paths that do not lead to any meaningful word, identified in *red* in Figure 4, and we do not explore deeper levels of such subtree. All the orange paths are possible words, which have however lower probabilities (not shown here to improve readability) than the recognized word which is shown in *green*.

As we will show later, the tree of longer words eventually have less leafs at the end of the recognition than those of shorter words, as the possibility of having an impossible word is raised, reducing the possibilities for choosing the final word.

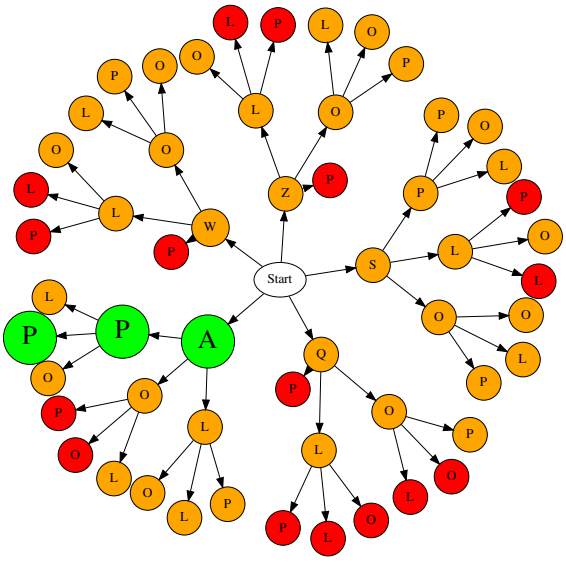


Fig. 4: Example tree with $\mathcal{T}_\Lambda = app$. In *green* we show the recognized word, in *orange* we show possible yet less probable words, and in *red* we show words which do not exists in the dictionary, hence removed. Figure best viewed in colors.

B. Results

We now present the performance evaluation of the word recognition phase. In particular, we evaluated the average Levenshtein Distance from the recognized word and the real one, the accuracy of top-3 presences of the word, and the average position in the list of possible word out of the tree $\mathcal{T}$, sorted on probabilities.

The Levenshtein distance is a popular metric to compare two strings. Basically, the Levenshtein distance is the minimum number of simple changes applied to a string A to change it to a string B. The changes can be removing a character or inserting it, or substituting a character with another one. For instance, the Levenshtein distance between string *home* and string *cone* is 2, as we need to change the characters *h* and *m* into characters *c* and *n*, respectively. For our analysis, as we always compare strings of the same length, the changes can only be related to the substitution of characters.

Figure 5a shows the average rank of the correct word among all the possibilities. We plot the results with probabilities equal to 0.4 which matches the KBD recognition with $\mathcal{M}^{A,M}$ shown in Figure 2b, 0.6 which matches the KBD recognition with $\mathcal{M}^{A}$, 0.8 and 1. We also show higher probabilities than those we presented in Section III to show the possible performances of the system in case improved touch event recognition is used. Obviously higher probabilities on average achieve better ranks, as there is a higher chance of correctly identifying the right area touched. Aside from the $p = 1$ case which shows a different trend, all the other 3 probabilities have lines with similar slopes. The rank increases up to a string with a length of 4, then starts to decrease. This happens because with few characters there may be multiple possible words, hence the right word may end up in a lower higher rank. Interesting to note that the density of the words length, shown with a blue line on the right y axis, there are far more words with longer lengths, with the top at 8 and 9 characters. However, as in our analysis we remove words outside the dictionary, having longer words actually improves our recognition ability.

Figure 5b shows the accuracy of the word recognition, which follows a similar yet inverse slope compared to Figure 5a. It is possible to see that longer words (i.e. greater than 8 characters) can be recognized with satisfactory performances even with low probabilities, while higher probabilities enable to recognize even shorter words.

However when comparing strings, the Levenshtein distance is regarded as one of the more truthful metrics to understand how close two strings are to each other. Figure 5c and Figure 5d show that even with low recognition probabilities, it is possible to achieve a rather low Levenshtein distance, even for short words. Interestingly, with longer words, thus with a possibility for the Levenshtein distance to increase, it remains constant, meaning a more accurate recognition.

V. Conclusion and Future Work

Sensors integrated in modern mobile devices open up for several services, by monitoring the movements and position of the devices. However, as we have shown in this work they can be also used to obtain sensitive information about what the user is typing, by monitoring the orientation of the device. Correlating subsequent identified touched areas with an English dictionary, a rather challenging task, shows that words can be recognized even with modest accuracies in recognizing the touched area.

Future work within this topic are devoted to the improvement of the touched area recognition, by using more complex yet more accurate models, and to also monitor the transition phase from the base tilt to the one referring to the touched area. Moreover, also correlating subsequent words together, hence recognizing sentences, may remove those sentence which are less probable than others, thus raising even more the overall accuracy. We will also develop different models which may be used whether the user is typing with the right hand, the left hand, or with both hands.

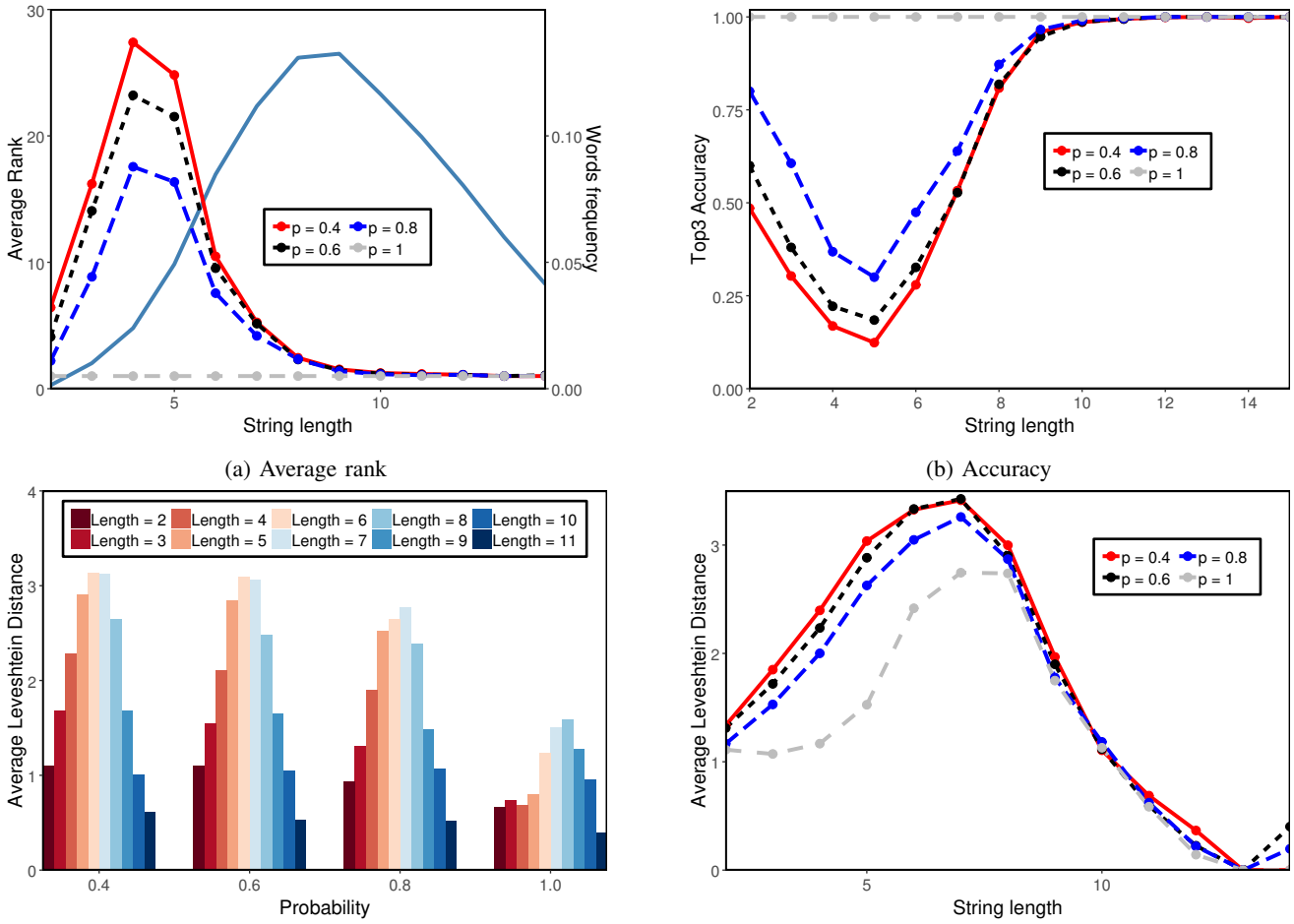
References

[1] Ling Bao and Stephen S. Intille. Activity recognition from user-annotated acceleration data. In Alois Ferscha and Friedemann Mattern, editors, *Pervasive Computing*, pages 1–17, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[2] Luca Bedogni, Marco Di Felice, and Luciano Bononi. Context-aware android applications through transportation mode detection techniques. *Wireless Communications and Mobile Computing*, 16(16).

[3] Philippe Golle and Kurt Partridge. On the anonymity of home/work location pairs. In Hideyuki Tokuda, Michael Beigl, Adrian Friday, A. J. Bernheim Brush, and Yoshito Tobe, editors, *Pervasive Computing*, pages 390–397, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[4] Y. Gu, Y. Yao, W. Liu, and J. Song. We know where you are: Home location identification in location-based social networks. In *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9, Aug 2016.



(a) Average rank

(b) Accuracy

(c) Levenshtein Distance

(d) Levenshtein Distance

Fig. 5: Word recognition evaluation. Figure 5a reports the average rank among the possibilities. Figure 5b shows the correctness accuracy of the word. Figures 5c-5d report the average Levenshtein distance versus the area recognition probability and the string length, respectively. Figures best viewed in colors.

[5] Inhwan Hwang, Jungchan Cho, and Songhwai Oh. Privacy-aware communication for smartphones using vibration. *2012 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 447–452, 2012.

[6] S. M. R. Islam, D. Kwak, M. H. Kabir, M. Hossain, and K. Kwak. The internet of things for health care: A comprehensive survey. *IEEE Access*, 3:678–708, 2015.

[7] G. Li, L. Huang, and H. Xu. iwalk: Let your smartphone remember you. In *2017 4th International Conference on Information Science and Control Engineering (ICISCE)*, pages 414–418, July 2017.

[8] Jalal Mahmud, Jeffrey Nichols, and Clemens Drews. Home location identification of twitter users. *ACM Trans. Intell. Syst. Technol.*, 5(3):47:1–47:21, July 2014.

[9] Anindya Maiti, Oscar Armbruster, Murtuza Jadliwala, and Jibo He. Smartwatch-based keystroke inference attacks and context-aware protection mechanisms. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '16, pages 795–806, New York, NY, USA, 2016. ACM.

[10] Anindya Maiti, Murtuza Jadliwala, Jibo He, and Igor Bilogrevic. (smart)watch your taps: Side-channel keystroke inference attacks using smartwatches. In *Proceedings of the 2015 ACM International Symposium on Wearable Computers*, ISWC '15, pages 27–30, New York, NY, USA, 2015. ACM.

[11] 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 & Applications*, HotMobile '12, pages 9:1–9:6, New York, NY, USA, 2012. ACM.

[12] Veljko Pejovic and Mirco Musolesi. Anticipatory Mobile Computing: A Survey of the State of the Art and Research Challenges. *ACM Computing Surveys (CSUR)*, 47(3):47, 2015.

[13] S. Sen, K. Grover, V. Subbaraju, and A. Misra. Inferring smartphone keypress via smartwatch inertial sensing. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 685–690, March 2017.

[14] Ahren Studer, Timothy Passaro, and Lujo Bauer. Don't bump, shake on it: The exploitation of a popular accelerometer-based smart phone exchange and its secure replacement. In *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC '11, pages 333–342, New York, NY, USA, 2011. ACM.

[15] Timothy Trippel, Ofir Weisse, Wenyuan Xu, Peter Honeyman, and Kevin Fu. Walnut: Waging doubt on the integrity of mems accelerometers with acoustic injection attacks. *2017 IEEE European Symposium on Security and Privacy (EuroSP)*, pages 3–18, 2017.

[16] Q. Xia, F. Hong, Y. Feng, and Z. Guo. Motionhacker: Motion sensor based eavesdropping on handwriting via smartwatch. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 468–473, April 2018.

[17] Tuo Yu, Haiming Jin, and Klara Nahrstedt. Writinghacker: Audio based eavesdropping of handwriting via mobile devices. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '16, pages 463–473, New York, NY, USA, 2016.