Keystroke Recognition With the Tapping Sound Recorded by Mobile Phone Microphones

Zhen Xiao, *Student Member, IEEE*, Tao Chen, Yang Liu, *Member, IEEE*, Jiao Li, *Student Member, IEEE*, and Zhenjiang Li, *Member, IEEE*

Abstract—Mobile phones nowadays are equipped with at least dual microphones. We find when a user is typing on a phone, the sounds generated from the vibration caused by finger's tapping on the screen surface can be captured by both microphones, and these recorded sounds alone are informative enough to localize the user's keystrokes. This ability can be leveraged to enable useful application designs, while it also raises a crucial privacy risk that the private information typed by users on mobile phones has a great potential to be leaked through such a recognition ability. In this paper, we address two key design issues and demonstrate, more importantly alarm people, that this risk is possible, which could be related to many of us when we use our mobile phones. We implement our proposed techniques in a prototype system and conduct extensive experiments. The evaluation results indicate promising successful rates for more than 4000 keystrokes from different users on various types of mobile phones.

Index Terms—Side-channel attacks, acoustic signal processing, smart devices

1 Introduction

Mobile phones nowadays are commonly equipped with (at least) dual microphones. The one at the bottom of a phone receives the user's voice in the phone call, and the other one on the top of the phone measures the ambient noise level for the noise cancellation. Multiple microphones lead to a significant advance for improving the phone-call quality [1]. Meanwhile, in practice, users need to type on their mobile phones from time to time, e.g., writing messages, inputting passwords, etc. In this paper, we find that when a user is typing, finger's tapping on the phone's screen surface could cause a vibration of the touching point on the device. The sound generated from this vibration can be captured by both top- and bottom-microphones on the phone as illustrated in Fig. 1a, and these recorded sounds are informative enough to infer the user's keystrokes.

On a positive side, recent studies have exploited the ability of microphones to develop various useful applications [2], [3], [4]. This keystroke recognition ability could be further leveraged in the future to develop a new input modality on a ubiquitous surface [5] using two lightweight microphones only. On the contrary, this also raises an immediate and serious privacy concern — plenty of the user's private information (frequently typed by users on the phone, e.g., personal data, passwords, messages, etc.) has a great potential to be compromised through mobile phones when the microphone data is hacked (attack model is in Section 2.3) and the barrier to launch this hacking is not high [6]. Hence, in this paper, we focus on studying this keystroke recognition ability from an attacking perspective to alarm people such a potential privacy leakage risk that could severely sacrifice the user's typing safety.

We would like to note that this paper is not intended to say the user's tapping necessarily leads to the typing privacy leakage. Because such tapping sounds are relatively weak, they can be overwhelmed by the strong ambient noises, e.g., people's conversations nearby (attack model is in Section 2). However, it is indeed common that the sounds generated from the vibration caused by the user finger's tapping can be recorded by microphones clearly in practice, which are thus worth drawing our attention on this potential privacy leakage risk. To implement this keystroke recognition, we need to address the following three challenges.

1) *Weak acoustic signals*. Although microphones can receive the sounds from the vibration caused by finger's tapping when a user is typing, they are very weak signals, e.g., users are even not aware of their existence usually. On the other hand, due to the limited size of a mobile phone, the length of each key and the distance between two adjacent keys (Fig. 1b) are short normally, e.g., around 1 cm and 3 mm [6], respectively. With a maximum microphone sampling rate on many phones, e.g., 192 KHz, the resolution (i.e., the distinguishable distance cross two consecutive acoustic samples that is 1.7 mm with the 192 KHz sampling rate) in principle can recognize user's keystrokes on different keys. However, the screen tapping is a subtle motion and its produced sounds are weak with low signal

• Zhen Xiao, Tao Chen, Yang Liu, and Jiao Li are with the Department of Computer Science, City University of Hong Kong, Hong Kong, China. E-mail: {zxian4-c, tachen6-c, yliu562-c, jli242-c}@my.cityu.edu.hk.
• Zhenjiang Li is with the Department of Computer Science, City University of Hong Kong, Hong Kong, China, and also with the City University of Hong Kong Shenzhen Research Institute, Shenzhen, Guangdong 518057, China. E-mail: zhenjiang.li@cityu.edu.hk.

Manuscript received 5 July 2021; revised 3 Dec. 2021; accepted 10 Dec. 2021. Date of publication 21 Dec. 2021; date of current version 5 May 2023.
This work was supported in part by The Science Technology and Innovation Committee of Shenzhen Municipality under Grant JCYJ20190808183203749 and in part by the GRF Research Grants Council of Hong Kong under Grant CityU 11217817.
(Corresponding author: Zhenjiang Li.)
Digital Object Identifier no. 10.1109/TMC.2021.3137229

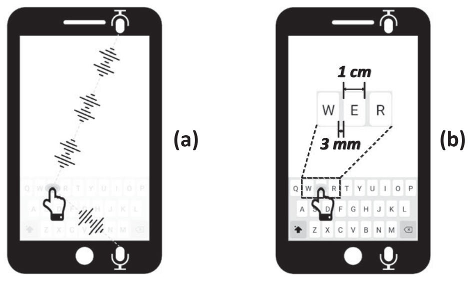

Fig. 1. (a) The sounds generated from the vibration (of the touching spot on the device) caused by finger's tapping on the screen can be captured by microphones. (b) Typical length of a key and distance between two adjacent keys.

to noise ratios (SNRs). This challenges the precise segmentation and feature extraction for the recorded sounds in the first place, because a slight signal processing inaccuracy may bring the errors easily overwhelming the desired resolution. As a result, if this issue is not addressed, the consequent keystroke recognition design is not viable.

2) *Impact of electronic typing sound and vibration*. The mobile phone may generate an electronic sound through the speaker and/or a device vibration for each keystroke during the typing. If a user does not disable them on her device, they will also be recorded by the microphones and interfere with the user finger's typing sound. We find that such interfering noises will have a significant impact on the performance of the keystroke recognition. Therefore, the mechanism to detect and exclude (if detected) the electronic typing sound and the vibration sound is necessary prior to the keystroke recognition.

3) *Unsupervised keystroke recognition*. Even the acoustic signals were precisely processed finally, the user's keystrokes still cannot obtained immediately, because the adversary may not have the labelled ground truth from the victim user to train a classification system to recognize this user's keystrokes. Hence, a more practical setting is to achieve an unsupervised keystroke recognition design, e.g., the adversary utilizes her own data to train a classification system, yet it could be further applied to a new victim user as well. However, different people may have detailed typing behavior differences, leading to different keystroke features. As a result, an effective design to largely extract user-independent keystroke features to enable the recognition is needed.

To address above challenges, we first decompose the original recognition task (for all the keyboard keys) into a series of recognition tasks with smaller "sizes" (e.g., less keystrokes to be recognized for each). This could avoid the requirement on the high-standard input data to recognize a large amount of keys simultaneously. In particular, we utilize the time difference of arrival (TDoA) [7] of the generated sound for each keystroke measured by two microphones to divide all the keys into three groups. Then we can train three classifiers for each group. However, the recorded acoustic signals are weak with low SNRs as stated above. It is not straightforward to precisely identify the starting point of each sound wave that corresponds to one keystroke, whereas if this identification is inaccurate, the pre-grouping will be wrong and the final result cannot be correct. We thus propose effective de-noise and segmentation designs to tackle this issue.

Meanwhile, we thoroughly study the characteristics of the electronic typing sound and the vibration sound, and obverse that due to the relatively long temporal delay of the electronic typing sound, our signal pre-processing can remove it automatically, while the vibration sound is still mixed with the keystroke sound. To detect the existence of the vibration sound, we find that it exhibits a dominating response in the frequency domain, based on which we can introduce a mechanism to detect it reliably. To further eliminate it from the mixed signal, we first utilize a band-pass filter to obtain its main feature and reconstruct the vibration signal. Then, we propose a searching method to identify a proper separation point in the mixed signal, so that a clear keystroke signal can be obtained by subtracting the reconstructed signal from the mixed signal starting from this separation point.

Finally, we leverage the auto-encoder framework from the deep learning domain [8] to extract the representative features to achieve good recognition performance even with the low-SNR data as input. In addition, we further leverage the auto-encoder to fulfill an unsupervised keystroke recognition system design to avoid requiring the labelled ground truth from the victim user. Our basic idea is to cluster different keystrokes and integrate the clustering-related loss functions into the neural network design for keystroke recognition. By doing so, the entire design does not rely on any ground truth data from the victim user to accomplish the system training. Instead, the system will automatically mine the representative features leading to the desired number of clusters, and the preserved features tend to be more user independent.

To demonstrate the efficacy of above designs, we develop a prototype system, named *TapLeak*. We conduct extensive experiments with six volunteers and we act as the adversary to attack more than 4000 users' keystrokes on different types of mobile phones, wherein the volunteers' data are not used in the system training. The results show that *TapLeak*'s top-1 successful rate is 84% and the top-3 accuracy increases to 92%. In summary, we make the following contributions.

- We demonstrate the possibility to infer user's keystrokes only using the sounds from finger's tapping on screen and revealing (more importantly alarm people) the potential typing privacy leakage risk that may not be viable before.
- We propose effective techniques to address the weak acoustic signals, impacts of electronic typing sound and vibration, and unsupervised keystroke recognition three main challenges in designing *TapLeak*.
- We develop a prototype system and conduct extensive experiments by attacking different users' more than 4000 keystrokes on different mobile phones.

The rest of this paper is organized as follows. We introduce the preliminary and attack model in Section 2. The design is detailed in Section 3 and evaluated in Section 4. We introduce possible defense mechanisms and limitations in Section 5. We review related works in Section 6 before we conclude in Section 7.

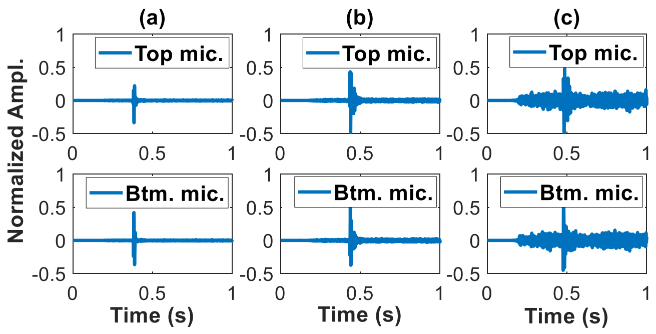

Fig. 2. The sounds generated due to the finger's tapping on the screen received by the top-microphone (the 1st row) and the bottom-microphone (the 2nd row) on a mobile phone (Samsung Galaxy S7) in three environments with different background noise levels: (a) a quiet library (35 dBSPL noise), (b) a normal office (55 dBSPL noise), and (c) a noisy canteen (70 dBSPL noise), respectively.

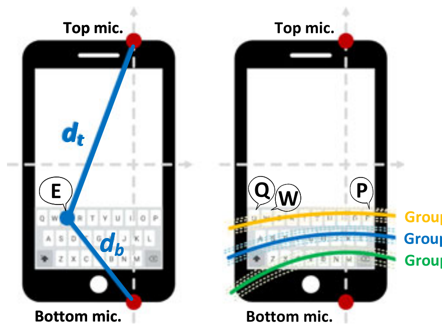

Fig. 3. (a) Illustration of the TDoA measure for tapping key "E". (b) Representative hyperbolas for each group.

2 Preliminary and Attack Model

In this section, we introduce the preliminary and the attack model before we detail the *TapLeak* design in Section 3.

2.1 Detectable Sounds Due to Finger's Tapping

Although the sound generated due to user's finger tapping on the screen is weak, e.g., users do not hear them usually, the microphones on the mobile phone are near the sound source and sensitive enough to capture them. In Fig. 2, we collect such sounds in three environments with different background noise levels, including a quiet library, a normal office and a noisy canteen. From the result, we can see that the sounds are detectable by both microphones, even in the noisy canteen scenario. We repeat the experiments using different phones and obtain a similar result. This brings an opportunity to infer the user's keystrokes through such recorded sounds.

For these recorded sounds, we further analyze their characteristics and have the following observations:

- *Time domain*: for each keystroke, the recorded sounds last for around 40 ms, leading to 7680 sampling points at the maximum sampling rate 192 KHz [9] on many phones.
- *Frequency domain*: the energy of the recorded sounds mainly fall in the range less than 2000 Hz, which could be mixed with the background noise.

We consider these factors for designing *TapLeak* in Section 3.

2.2 Distinct Sound Features

For these detectable sound waves, we then investigate their distinct features to be used in the *TapLeak* design. In particular, we characterize these features from temporal and frequency two dimensions.**Temporal Features.** We can record the sounds for both microphones at the same time [10]. Since the keyboard is closer to the bottom-microphone usually, when a user types on the screen, there exists a time difference of arrival (TDoA) for the generated sound received by two microphones, i.e., $\Delta t = \frac{d_t - d_b}{v}$, where d_t and d_b (in Fig. 3a) represent the distances to top- and bottom-microphones respectively, and v is the speed of sound. However, the TDoA values for different tapping locations may not be unique and all the locations with the same TDoA values could form a hyperbola with respect to the two microphones' locations. According to the 192 KHz sampling rate, we can draw more than 50 hyperbolas covering the keyboard area on a phone (this number can vary slightly due to the phone size).

To avoid recognizing all the keys simultaneously (Section 1), we can classify all the keys into groups and then recognize them in each group only. In *TapLeak*, we view each row on the keyboard as one group and there are thus three groups in total. For each group, there could be a set of hyperbolas (with different TDoA values) overlapping with all the keys in this group, and we select the hyperbola with a median TDoA value in the set to represent this group. Fig. 3b illustrates the three representative hyperbolas for each group. We note that the selection of the hyperbolas for each group could be different on different phones, which however is an one-time effort. As we discuss soon in the attack model, we assume that the adversary knows the specific phone type of the victim user and can thus complete this selection in advance.

In summary, the temporal difference (TDoA) first classifies the received sounds into one of three key groups (according to the closeness to the three representative hyperbolas), based on which we will further recognize them inside each group. **Frequency Features.** Mobile phone is a rigid object but it may have heterogeneous densities and structures at different places inside the device. As a result, when the user's finger taps different spots on the screen, the frequency spectrum of this vibration exhibits different features, which in turn generates the acoustic sounds with distinct features. Therefore, we can further distinguish different keys in each group according to such frequency-domain features. For instance, Fig. 4 shows the spectrum of keystrokes on keys "Q", "W" and "P" in the first group (row), respectively. We can see that for keys "W" and "P", which are far away to each other, their spectrums are quite different. Moreover, for the neighboring keys "Q" and "W", their difference still exists yet becomes less obvious from our manual observation. Thus, we propose to leverage neural networks to extract their subtle differences to enable the keystroke recognition (Section 3).

2.3 Attack Model

The goal of this attack is to infer the user's keystrokes on a mobile device using the finger's tapping sounds captured

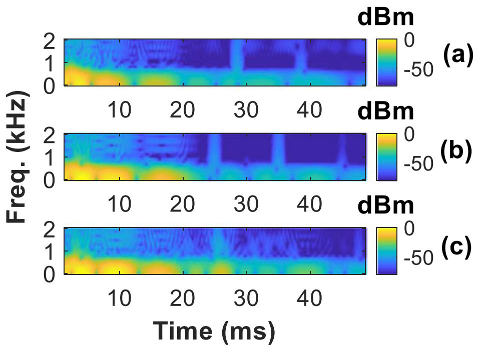

Fig. 4. Spectrum of the keystrokes on keys (a) "Q", (b) "W" and (c) "P" in one group, respectively.

by device's microphones. We consider the following attack model.

1) *Phone and keyboard types*. We assume that the adversary has the prior knowledge of the type of the victims mobile phone, e.g., adversary can peep in the victim's vicinity, so that the adversary knows the necessary information to select the representative hyperbolas for each group in advance, such as the phone size, the microphone locations and the keyboard layout. Therefore, we mainly focus on the keystroke inference design in this paper, with the standard English keyboard in a portrait orientation by default.
2) *Ambient noises*. In *TapLeak*, we consider the impact from background noise. However, we assume that there are no other ambient noises that dominate the recorded sounds, e.g., loud conversations of people nearby. The adversary can launch the attack selectively to minimize the influence from such ambient noises from the environment. Moreover, we validate the impact of the background noise to the performance of *TapLeak* in Section 4.4.
3) *Software generated signals*. It is possible that the operating system generates a sound through the speaker and/or a device vibration for each keystroke during the typing. We do not assume this function is disabled. Instead, we develop a mechanism to first detect their existence. If detected, we further propose a design to exclude them before the keystroke recognition.
4) *Hacking microphone data*. We assume that the adversary can access two microphones of the victims phone to collect the microphone data and send them out. To this end, the adversary can develop a malicious APP as a Trojan [11]. The adversary can disguise the Trojan as some useful legitimate APP and publish it to the APP market to fool the victim to install, which provides the functions, e.g., voice recognition, to gain the permission to access microphones and Internet [12] during the installation of the Trojan APP. Once it is installed and used, i.e., the victim gives corresponding permissions, the malicious APP can listen to the user's finger tapping sounds on the screen and send the collected acoustic data to the adversary in the background.

3 Design of *TapLeak*

The working flow of *TapLeak* is illustrated in Fig. 5, which contains four main steps:

- *Signal pre-processing and segmentation*. For the recorded tapping sound waves, the adversary needs to identify the starting point for each keystroke precisely, challenged by the relatively high noise levels, e.g., low SNRs. To this end, the adversary should design a proper pre-processing to improve the quality of the signal first.
- *Vibration detection and elimination*. In addition to the background noise, the raw acoustic signal may contain electronic typing sounds and vibrations generated by the mobile phone operating system. The adversary has to first detect their existence and then eliminate them to minimum their impact to the further recognition.
- *Pre-grouping*. For the segmented sound clips from both microphones that correspond to one keystroke, the adversary needs to further synchronize them to compute their TDoA value for pre-grouping. The TDoA measurements should be accurate, so that the correct group can be selected for the current keystroke.
- *Keystroke recognition*. According to the pre-grouping result, the adversary selects the neural network corresponding to the current group for the keystroke recognition.

We now detail the design of each step in the rest of this section.

3.1 Signal Pre-Processing and Segmentation

For the recorded tapping sound waves, the first step is to identify the starting point for each keystroke and then segment them as a series of sound clips (one clip corresponds to one keystroke). The adversary obtains a stream of sound

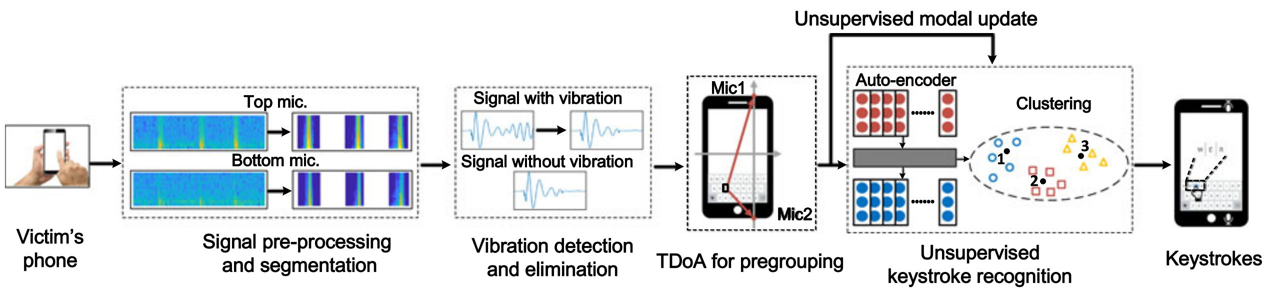

Fig. 5. Illustration of the working flow to infer the user's keystrokes in *TapLeak* with three main steps.

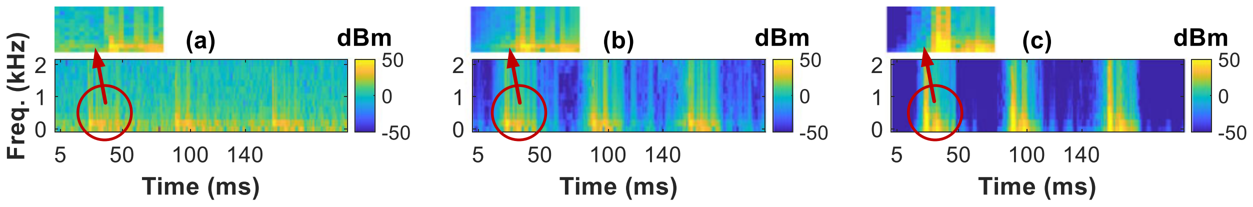

Fig. 6. Pre-processing to identify the starting point for each keystroke. Spectrum of (a) the raw sound wave, (b) the sound after Wiener filter and (c) the sound after Wiener filter plus a further difference amplifying. Most of the empty time interval between two keystrokes is omitted for a clear illustration of the spectrum.

clip pairs (from two microphones), which will be used by the following system modules.

3.1.1 Design Challenge

The challenge in the first step is the high-standard precision requirement on the starting point search. Due to the limited microphone sampling rate, e.g., 192 KHz, one sampling point difference leads to a 1.7 mm error in the distance calculation to each microphone. However, the recorded sounds are weak with low SNRs and the background noises could hide the starting pointing to prevent a precise boundary search. Fig. 6a shows the spectrum of the raw sound saves recorded with three keystrokes. Although it is easy to tell the existence of these three keystrokes, the exact starting point of each keystroke is blurred by the noise.

As stated in Section 1, the length of a key and distance between two adjacent keys are short, e.g., around 1 cm and 3 mm, respectively. The search error of the starting point can thus easily cause an inaccurate TDoA measure later, which, as a direct consequence, could lead to a wrong pre-grouping result and also the final keystroke recognition result.

3.1.2 Solution

We propose the following pre-processing before the segmentation to overcome this issue.

Because the keystroke signal and the noise have an overlap in the frequency domain, we cannot apply the band-pass filter to remove the noise directly. Thus, we leverage Wiener filter [13] to handle such frequency-overlapped noise. Wiener filter requires to collect one short piece of noise samples, e.g., 0.25 seconds, before the filtering. This aims to "analyze" the noise's frequency characteristics to determine the parameters in the filter, which then can be adopted to process the recorded sounds. Fig. 6b shows the result after we apply the Wiener filter. We can see the background noise is excluded substantially and the starting point becomes more identifiable.

However, the starting point's boundary is still not "sharp" enough in Fig. 6b due to the residual noise. After analyzing this spectrum, we observe that the noise becomes much weaker (compared with keystroke sounds) already. Therefore, we propose to further amplify the difference between the keystroke sound and noise, so that the keystroke sound could dominate the spectrum finally with clearer boundaries for each starting point. To this end, for the sound wave after the Wiener filter, we compute the square for each time step t as follows, which can make the large amplitude even larger and a small amplitude (e.g., <1 for noise) even smaller

$$s'(t) = \sum_{n=t}^{t+\mathcal{W}} s^2(n), \tag{1}$$

where $s(n)$ is the amplitude of the acoustic signal and $\mathcal{W}$ is a sliding window and we set its size $|\mathcal{W}|$ to 20 in our current design (e.g., approximately 0.1 ms with the 192 KHz sampling rate). Fig. 6c shows the spectrum for $s'(t)$, which now has a sharper starting point for each keystroke.

After the signal pre-processing above, we can segment the sound waves into clips with a clearly identified starting point for each. Through our study, we select 50 ms as the segmentation window size in our current design, which can ensure the segmented sound clips containing the entire keystroke information and produce the spectrum with sufficient size serving as the input of the neural network to achieve good keystroke recognition performance (Section 3.4). Two clips (from two microphones) that own a similar starting point in the time domain (e.g., correspond to the same keystroke) then form a pair for the pre-grouping module.

3.2 Electronic Typing Sound and Vibration

Before designing the pre-grouping module, we note that when a user is typing, the mobile phone could play an electronic typing sound through the speaker and/or trigger the motor of the phone to vibrate. Both of them are non-negligible sounds, which can be recorded by the microphone and mixed with the user finger's tapping sound. These extra sounds could impair the keystroke recognition. Hence, prior to the pre-grouping module design, we first study how to detect and further eliminate them in this section.

3.2.1 Observations

We denote the electronic typing sound and the vibration sound as *e-sound* and *v-sound*, respectively. Fig. 7 illustrates the keystroke signal mixed with both of them, which show different characteristics. Because the e-sound and the v-sound are generated due to the user's tapping, both of them will be recorded after the user finger's tapping sound[1], while we find that they appear after distinct delays, which are likely caused by the responding time of different hardware components as follows.

1. The e-sound and v-sound are independent from each other — the e-sound is an electronic sound and the v-sound is a mechanical vibration sound. Hence, when one of them is disabled on the phone, another one can still appear.

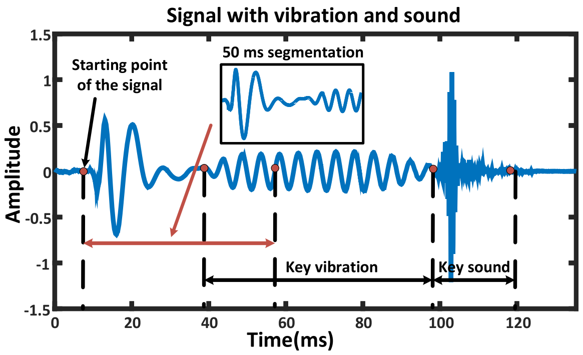

Fig. 7. Keystroke signal with key sound and vibration. The vibration has a delay about 30ms after the starting point of the signal, while the delay for the key sound is about 100 ms.

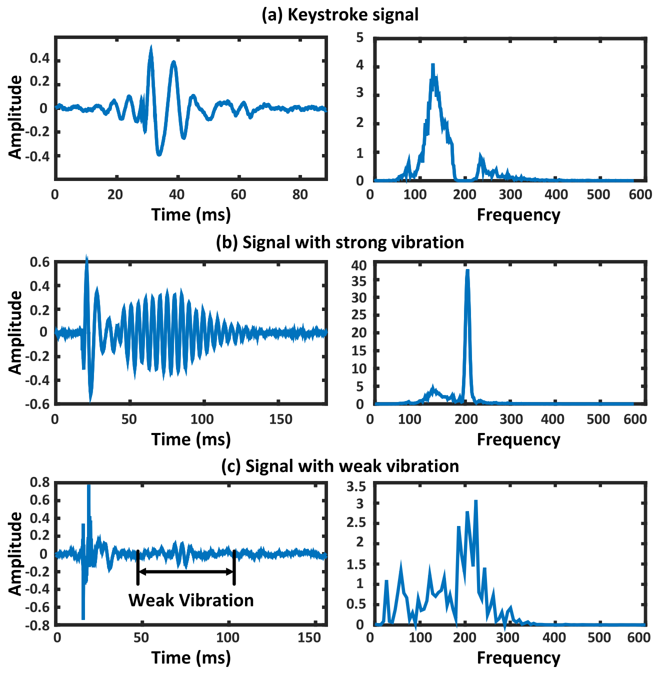

Fig. 8. Keystroke signal and its corresponding frequency response in different cases. (a) Pure keystroke signal. (b) Signal with a strong vibration. (c) Signal with a weak vibration.

1) The e-sound. The latency of the e-sound is approximately 100 ms through our measurements on different types of smartphones, e.g., Nexus 5X, SAMSUNG S7 and Huawei P30 Pro. As stated in the signal pre-processing (Section 3.1), the segmentation window is 50 ms. The e-sound will not be included in the user finger's tapping sound after the segmentation. Moreover, it is also unlikely for the e-sound to overlap with the user's next tapping sound, since if so, the user needs to type at least 10 keys per second. Therefore, we introduce a time constraint empirically, e.g., 150 ms, between the starting points of two consequent keystroke segmentations in the pre-processing stage to avoid that we select an e-sound or v-sound as the beginning of one segmentation window. With this operation, the e-sound can be eliminated directly, which will not influence the keystroke recognition.

One point worth noting is that the e-sound is triggered by the "AudioManager" in the smartphone's operating system. With the hardware advance and the software optimization in the future, the latency of the e-sound could be shortened. Because the the adversary is able to get the prior knowledge of the type of the victim's smartphone (e.g., to peep in the victims vicinity) and measure this latency in advance, the attack still could work after the segmentation window size is adjusted.

2) The v-sound. The latency of the v-sound is approximately 30 ms through our measurement. Different from the e-sound, v-sound will be included in the segmentation window and mixed with the user finger's tapping sound, as shown in Fig. 7. One naive solution is to reduce the segmentation window size, e.g., less than 30 ms. However, we find that the user's tapping sound usually lasts more than 30 ms. If the segmentation window becomes too small, useful features will be lost and the recognition performance naturally deteriorates, as unveiled in Section 4. Therefore, we aim to identify and then eliminate the v-sound, so that we can preserve the user's tapping sound with the best effort to ensure a good recognition performance. In the following, we elaborate the design of the v-sound detection and elimination.

3.2.2 Detection of Vibration Sound

Because the attacker may not know whether the vibration function is enabled on the victim user's mobile phone, *TapLeak* needs to detect the vibration sound first. If so, it will be further eliminated. The motor vibrates following its own designed frequency, denoted as f_v. Therefore, we should observe it in the frequency domain from the recorded v-sound. We note that this vibration frequency f_v is not fixed on different types of the smartphones, e.g., 180 Hz, 200 Hz and 300 Hz on Nexus 5X, SAMSUNG S7 and Huawei P30 Pro respectively, the adversary can adjust the system parameter related to f_v as stated below after the adversary knows the type of the victim's smartphone (the attack model) prior to launching this attack. To facilitate our discussion, we adopt $f_v = 200$ Hz as an example to introduce the following design.

Fig. 8a first shows the frequency response of a pure keystroke signal. Compared with the frequency response of the keystroke sound mixed with the v-sound shown in Fig. 8b, we find that there is a distinct peak around f_v (e.g., 200 Hz) corresponding to the v-sound and the magnitudes of the overall frequency components of the mixed sound signal are larger than the pure keystroke signal. These two observation inspire us to check whether enough signal strength can be measured from that particular frequency range as well as the total power of the recorded signal to detect the vibration, e.g., greater than certain thresholds.

However, the practical hurdle is that the motor may have different vibration levels. Fig. 8c shows the keystroke signal mixed with a weak v-sound. We can see that the frequency response, around f_v (e.g., 200 Hz), still exist. However, it is non-trivial to select a suitable threshold to determine its appearance reliably, due to the motor's different vibration levels.

To achieve a more reliable v-sound detection, we view it as a classification problem with two classes — with and without the v-sound. Based on the observations stated before, we select the frequency component at f_v (e.g., 200 Hz) and the power of the entire signal as two features to train a classifier. In particular, we aim to train a simple yet

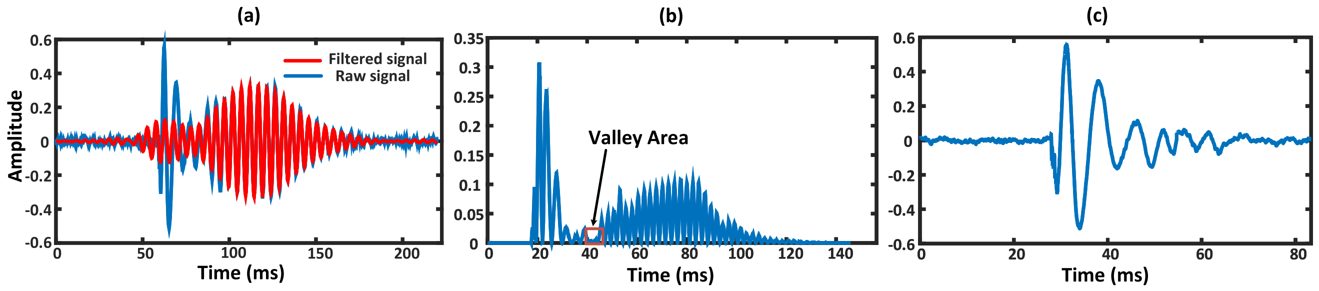

Fig. 9. (a) The blue line is the raw keystroke signal and the red line is the filtered signal with a band pass filter. (b) The square value of the raw keystroke signal. The red rectangular locates the valley area. (c) The signal after vibration elimination.

effective linear function $D(p_i, f_i)$ to detect the presence of the v-sound as follow

$$D(p_i, f_i) = w_1 \times p_i + w_2 \times f_i + b, \tag{2}$$

wherein for the ith segmented signal, p_i is the total power of the signal in this segmentation window, f_i is the strength of the frequency component at f_v (e.g., 200 Hz) after FFT, w_1, w_2 and b are linear coefficients. Then the detection result V_i defined as

$$V_i = \begin{cases} 1, if & D(p_i, f_i) > 0 \\ -1, if & D(p_i, f_i) < 0, \end{cases} \tag{3}$$

where the value of 1 indicates the v-sound is detected, and -1 indicates there is no v-sound.

To verify the design, we collect 1300 keystroke signals without the v-sound (50 signals for each key) and the same amount of keystroke signals containing the v-sounds of different levels to train the classifier. We select more low-level v-sound data to make the classifier more reliable to distinguish the signals with weaker v-sounds from the signal without the v-sound. The experiment result shows that the classifier can reach the detection accuracy more than 99%, while the accuracy of the threshold-based method is no more than 83% with the best threshold that we have searched.

3.2.3 Elimination of Vibration Sound

If the v-sound is detected, it will be eliminated from the user finger's tapping sound. To this end, the following method is proposed in *TapLeak*. Because the v-sound mainly resides around the vibration frequency of the motor in the frequency domain, e.g., f_v, we can first utilize a band pass filter, which covers this frequency, e.g., $[f_v - 10, f_v + 10]$ Hz, to extract the v-sound. Fig. 9a shows the filtered signal as well as the original mixed signal. We can see that the filtered contains two parts. The first part (left) is the frequency components falling into the band-pass filter (within [190, 210] Hz) from the user's tapping sound and the second part (right) is mainly the v-sound, which matches the same part in the original mixed sound signal well. We denote the original mixed sound signal as $r(t)$ containing vibration and the filtered signal as $f(t)$. The user finger's tapping sound signal $s(t)$ can be obtained as follow

$$s(t) = \begin{cases} r(t), t \le T_s, \\ r(t) - f(t), t > T_s, \end{cases} \tag{4}$$

where T_s is a separation point to be discussed next. To find a proper separation point T_s in Eqn. (4), we then look at the original mixed sound signal again in the time domain. For the ease of illustration, we calculate the square of each signal sample values in the mixed sound as shown in Fig. 9b. We observe a valley area in the signal and select the point with the weakest signal strength as T_s due to the following reasons. The v-sound starts to appear when the user finger's tapping sound enters the ending phase. On the other hand, the strength of the v-sound is also relatively small initially. Hence, the point with the lowest signal strength in the valley area of the mixed sound implies that both of the user's tapping sound and the v-sound reach a weakest state, and we hence select it as the separation point T_s in our current design.

To put all above discussions together, we summarize the working flow to handle the vibration sound in Fig. 10. For each segmented sound clip (from Section 3.1), we first apply the vibration detection (Section 3.2.2). If no v-sound is detected, the processing moves to the pre-grouping module (Section 3.2.3); Otherwise, it is a mixed sound and the v-sound will be eliminated by the following three steps:

- *Step 1: Finding valley area.* We first calculate the envelop of the signal sample squares of the mixed sound signal to search the valley area. Considering that the v-sound increases first and then decreases, we start the search from the vibration part by checking the value of each peak in a reverse order, i.e., from the end of the signal to the beginning. Once the values of peaks stop decreasing within a certain time interval (e.g., 500 samples), a boundary of the valley area is considered to be found. We set the width of the valley area as 1000 samples, which corresponds to 5 ms of the signal.
- *Step 2: Selecting the separation point.* Within the valley area, we select the signal sample with the minimum signal strength as T_s in our current design.
- *Step 3: Subtracting the v-sound.* We apply a band pass filter to the original mixed sound signal, and substitute the separation point value obtained from Step 2

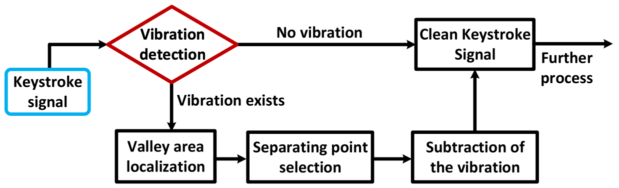

Fig. 10. The flowchart of vibration elimination process.

into Eqn. (4) to obtain the keystroke sound $s(t)$. Fig. 9c shows the sound $s(t)$ obtained from the mixed signal in Fig. 9a.

3.3 Pre-Grouping Based on TDoA

As aforementioned in Section 2.2, we divide the keyboard keys into three groups and recognize keystrokes inside one group only, to avoid recognizing all the keys each time[2]. In particular, for each sound clip pair obtained after the signal segmentation, we calculate their TDoA value with respect to the two microphones. As discussed for Fig. 3b, the adversary can determine the representative hyperbola (with the median TDoA value) for each group in advance. Then the current sound clip pair will be classified into the group whose representative hyperbola's TDoA is closest to this sound clip pair's. The adversary will later use the neural network of this group to conduct the keystroke recognition in Section 3.4.

Although we can record the sounds for both microphones at the same time [10] and have also identified the starting points for both sound clips in one pair already, we do not suggest to compute their TDoA directly due to the following reason. Denote $\hat{T_b}$ and T_b as the actual and our searched starting points for the sound clip received by the bottom-microphone. The search error for this starting point is $E_b = T_b - \hat{T_b}$. We can similarly define the error for the top-microphone as $E_t = T_t - \hat{T_t}$. Therefore, the calculated TDoA equals to

$$TDoA = (\hat{T_t} - \hat{T_b}) + (E_t - E_b). \tag{5}$$

In the above equation, the error comes form the second term $E_t - E_b$, indicating that the accuracy of TDoA measurements is affected by the errors of generating the starting points of the dual microphones. Hence, instead of computing difference between the starting points directly, we propose to improve the accuracy of TDoA measurements for *TapLeak*. In particular, we can fix the calculated starting point for one microphone and move the sound clip from the other microphone to perform the cross-correlation [7]. As these two sound clips essentially refer to the same audio content, the peak of the cross-correlation indicates that they are best overlapped with each other, and the starting point for the second microphone can be determined. Therefore, the cross-correlation basically introduces the starting point search error only once. Through our experiment in Section 4, we find that this can improve the pre-grouping accuracy by 30%.

3.4 Unsupervised Keystroke Recognition

The adversary finally needs to train one neural network for each group to recognize the keystrokes inside the group. Because the keyboard is closer to the bottom microphone usually (so its recorded sound is stronger), we thus use the segmented sound clips from this microphones for the keystroke recognition. However, the adversary does not have the labelled ground truth data from the victim user to train each neural network. Thus, the adversary is expected to achieve an unsupervised design, e.g., using her own data, to avoid the demand on the victim's data in the system training. To this end, we propose to utilize an advanced neural network framework, named auto-encoder, to extract the most representative features from the input sound clips and then leverage it to accomplish the unsupervised recognition system design.

3.4.1 Auto-Encoder Framework

An auto-encoder neural network contains the following four major components usually:

- *Input*: when the neural network handles acoustic signals, we normally provide the Mel-Frequency Cepstrum Coefficients (MFCC) [14] of the acoustic signal as input [15], which is a well-established representation of the sound's features (rather than processing raw sound waves) and is widely adopted in prior designs [16], [17], [18], [19], [20].
- *Encoder*: the encoder fulfills a non-linear conversion to extract the most representative features from the input.
- *Representative feature*: the extracted feature is viewed as a new representation of the original input (by preserving the most essential characteristics for a learning task).
- *Decoder*: the decoder adjusts the representative feature by utilizing it to recover the original input.

According to the similarity between the original input and the recovered input by decoder, we can define a recovery loss as

$$L_{rec} = \frac{1}{N}\sum_{t=1}^{N} f(x_i) - h(f(x(t))), \tag{6}$$

where $x(t)$ is the input at time stamp t, N is the number of input data, $f(\cdot)$ is the encoder and $h(\cdot)$ is the decoder. To couple above auto-encoder network with a specific learning task, the "representative feature" could serve as the input of the learning task, and its own loss function will be combined with L_{rec}. The training aims to minimize the overall loss.

3.4.2 Unsupervised Recognition With Auto-Encoder

With the auto-encoder framework stated above, we now introduce our recognition design in *TapLeak*. With this unsupervised recognition design, the training data do not need to be collected from the victim. Instead, the attacker can use his/her own typing data to train the system and apply it to attack the victim's typing directly.

Encoder and Decoder Designs. The auto-encoder framework is widely utilized in the computer vision domain [21], wherein the input is the 2D image. Therefore, the encoder and decoder are usually implemented by a multi-layer

2. To amplify the difference between the tapping sound and the residual noise for precisely identifying the starting point, Eqn. (1) could distort the original sound wave, which becomes less effective for recognizing keystrokes. Hence, we leverage it for the starting point search only. Afterwards, we still use the sound waves after the Wiener filter merely for the consequent keystroke recognition. Because they are still mixed with certain noises, it is necessary to avoid recognizing all the keys each time and use pre-grouping to improve the performance as evaluated in Section 4.

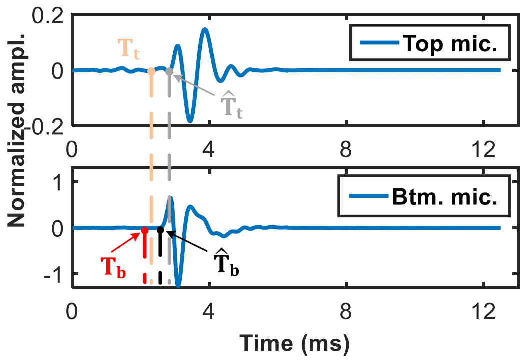

Fig. 11. Analysis of the TDoA error if we compute TDoA using the search starting points directly. The amplitude of the sound from the bottom-microphone is stronger because user's tapping position is closer to this microphone.

perceptron (MLP) or 2D convolutional neural network (CNN) [22]. For the acoustic signal, we can apply the short-term Fourier transform (STFT) and then compute its MFCC to obtain its MFCC spectrum (Fig. 12), which can be viewed as a 2D "image" and processed by using the 2D CNN (tailored to extract features from the images):

- Its x-axis is time. We set its length as 40 ms that can cover the segmented sound clip in the time domain.
- Its y-axis is the frequency. We set its range from 0 to 2 KHz to cover the frequency range of the tapping sound.
- The value of (x, y) indicates the frequency response.

Because the length of the spectrum's x-axis is short and the useful features are mixed with noises (low SNRs), we find that using one 2D CNN to analyze such a spectrum image as a whole could bias the feature extraction to the higher SNR parts and miss many useful yet less "obvious" parts.

Therefore, we propose to divide the spectrum image input into several strips along the time domain, e.g., the duration of each strip is set to 7 ms empirically in our current implementation. For each strip, we associate a three-layer CNN to analyze its feature. The purpose of this design is to fully modify the entire spectrum image to generate the final representative feature. In particular, we denote x^i as the the ith input strip, and the corresponding CNN $f^i(x^i)$ can be expressed as

$$f^i(x^i) = \sum_{j=1}^{n} \sum_{k=1}^{m} x^i_{j,k} \cdot w_k, \tag{7}$$

where n is the length of the input x^i, w is the coefficient vector of CNN, and m is the length of w. Each CNN in the decoder can be designed similarly. Through the evaluation in Section 4, we find this encoder and decoder design can effectively improve the recognition performance (Section 4).

Recognition Design. With the auto-encoder framework, we next fulfill the keystroke recognition design. Supposing for each keystroke sound clip, the adversary has its ground truth (i.e., which key is tapped). The adversary can train another neural network for classification (recognition), wherein the input of this network is the representative feature extracted by the auto-encoder and the output is the different keys to be recognized in the current group. However, the problem is that this recognition ability will be only effective for the user

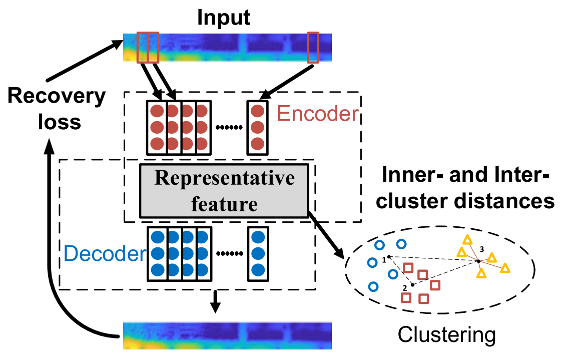

Fig. 12. Illustration of the auto-encoder based unsupervised keystroke recognition design.

whose data is used in the training, while the adversary may not have such ground truth from the victim user.

To overcome this issue, we find the adversary can collect her own tapping sound data but without requiring the ground truth of the data (otherwise the classifier can mainly recognize adversary's tapping). Then the adversary applies the clustering algorithm to cluster the representative features extracted from the auto-encoder, and the total number of clusters equals to the number of keys to be recognized in this group, which is known by the pre-grouping in advance. With the tailored loss functions defined below, we can combine them with the recovery loss L_{rec}, and train the auto-encoder and the clustering model, e.g., K-means [23], at the same time. By doing so, the overall system does not rely on any ground truth data to accomplish the training. Instead, the auto-encoder will automatically mine the representative feature that could lead to the desired number of clusters, and the preserved features tend to be more user independent.[3] To fulfill this design, we introduce the following two loss functions for clustering.

Inner-cluster distance L_{inner}. To cluster one data point, we can compute its distance to all the cluster centers and assign it to the cluster with the smallest distance. For a good clustering, the data points in one cluster should gather tightly near their cluster center. As a result, the average distance between the data points and their closest cluster center is one metric to quantify the quality of the clustering result. We thus define the inner-distance loss term as the average distance between the data points and their cluster centers as follows

$$L_{inner} = \frac{1}{N} \sum_{i=1}^{N} ||f^{(t)}(a_i) - c_i^{(t)}||, \tag{8}$$

$$c_i^{(t)} = \arg\min_{c_j^{(t-1)}} ||f^{(t)}(a_i) - c_j^{(t-1)}||, \tag{9}$$

where $f^{(t)}(\cdot)$ is the encoder at the tth iteration in the training, a_i is the feature value of the ith sound clip input, N is the total number of inputs used for training, and $c_i^{(t)}$ is the closest cluster center for a_i. Minimizing L_{inner} basically tends to gather all the features in one cluster close to each other.

3. Because the area that the user's finger touches the screen during typing is small, the characteristics from the vibration of the tapping spot likely dominates the feature of the generated sound. As a result, it is possible to recognizes keystrokes in an unsupervised manner.

Inter-cluster distance L_{inter}. On the other hand, the distance between cluster centers is also important to achieve a good clustering. For two neighboring clusters, if the distance between their centers is small, it is more likely to incur a clustering error. However, such inter-cluster distance cannot be used to define a loss term directly, because the auto-encoder and the clustering model need to be trained in an iterative manner. For a given iteration to optimize the auto-encoder, all the clustering centers are fixed from the last iteration, which thus form a set of constant values of distances between clustering centers. Such constant distances cannot be used to update weights of the current clustering model.

Fortunately, we observe that for each data point belonging to cluster c, if their distances to the centers of all other clusters (not c) increase, this implies that the cluster c tends to be farther away from other clusters, because the center of one cluster is computed as the average coordinates of data points in this cluster. So we define the inter-distance loss term as

$$L_{inter} = \sum_{i=1}^{N} ||f^{(t)}(a_i) - s_i^{(t)}||, \tag{10}$$

$$s_i^{(t)} = \arg\min_{c_j^{(t-1)} \neq c_i^{(t-1)}} ||f^{(t)}(a_i) - c_j^{(t-1)}||, \tag{11}$$

where $s_i^{(t)}$ is the second closest cluster center to the ith input data in iteration t. Maximizing L_{inter} thus tends to make different cluster centers separate away from each other.

Overall loss function L_{loss}. With the two loss functions introduced above, the final loss function in *TapLeak* is

$$L_{loss} = L_{rec} + \alpha \times L_{inner} - \beta \times L_{inter}, \tag{12}$$

where α and β are two parameters. With L_{loss}, we can train the auto-encoder and clustering model together in an iterative manner [24]. The cluster centers are initialized at the beginning randomly and we will investigate more advanced initialization strategies (e.g., hierarchical clustering) in the future. The total number of clusters equals to the total number of keystrokes to be recognized. Then in any iteration that optimizes the auto-encoder, the cluster centers computed from the previous iteration are fixed and the overall loss L_{loss} is calculated by the gradient descent to train the auto-encoder. After the optimization of the auto-encoder, the cluster centers will be updated using the auto-encoder obtained so far. This training process repeats until the overall loss becomes stable and cannot be further reduced. When the adversary collects his/her own typing sounds to form the training data set, the label of each sound is collected at the same time. Hence, to further determine the "key" that each cluster corresponds to, these labels will be used — the label that appear most in each group will be the label for this cluster.[4]

4. Even with such labels, the entire network training is still unsupervised, because these labels are not used in the training at all. Only when the training is completed and the adversary needs to further determine the "meaning" of each cluster, these labels are then used in the majority vote.

After training, when a new sound is provided, it will be processed by the encoder to obtain its representative feature, based on which we further classify it into one cluster corresponding one keystroke. We note that because *TapLeak* does not require any labelled ground truth in the training, this system is more user independent. Therefore, the adversary can use it to infer the keystrokes from a victim user whose data is not used in the system training any more, as evaluated in the next section.

3.5 System Fine-Tuning

After the victim user has been attacked, the attacker can obtain the actual tapping sound data from the victim user. There is a chance for the attacker to leverage the victim user's data to fine-tune the system to achieve a higher accuracy for the future attacks, while the attacker needs to decide when the fine-tuning can be terminated when the additional gain becomes marginal.

Because the fine-tuning only utilizes the victim's data to adjust the system slightly, we find that the value of the loss function keeps relatively stable, which is not a good indicator. The key component of the system is the unsupervised clustering. For a clustering algorithm, the stability of all the cluster center positions can indicate the convergence of the model during training process. Hence, we find that the coordinates of all the cluster centers could serve as a clearer sign to indicate the model convergence. Following this idea, we monitor the variance of the coordinates of all the cluster centers during the fine-tuning process as follows

$$M_i = \frac{1}{T}\sum_{j=0}^{T-1} c_i^{(t-j)}, \tag{13}$$

$$V_t = \frac{1}{NT}\sum_{i=1}^{N}\sum_{j=0}^{T-1} ||c_i^{(t-j)} - M_i||, \tag{14}$$

where T is the number of iterations used to calculate the variance, c_i^t is the coordinates of the ith cluster center in the tth iteration, M_i is the average of the coordinate for the ith cluster center in the past T iterations, and V_t is the variance of the coordinates for all the cluster centers in the tth iterations.

After certain amount of data is obtained from the victim user (e.g., 100 keystroke sounds), the attacker can launch the fine-tuning. In our current design, we select T as five, and the attacker keeps tracking the value of V_t. If V_t becomes stable (e.g., the difference is less than 1% for ten consecutive iterations), the attacker can terminate the fine-tuning process and adopt this customized model to improve performance. Experiment results shows that the fine-tuning can further improve by 2% to 6% for the top-1 to top-3 recognition accuracy.

4 System Evaluation

In this section, we evaluate the performance of *TapLeak*.

4.1 Experiment Setup

Implementation. We implement *TapLeak* on Samsung Galaxy S7, Nexus 5X and Huawei P30 Pro as the victim devices.

TABLE 1
Inference Time of Different Components

Operations	Inference time
Signal processing and pre-grouping	200 ms
Recognizing one keystroke in group 1	15 ms
Recognizing one keystroke in group 2	14 ms
Recognizing one keystroke in group 3	14 ms

For each device, we collect 200 keystrokes for each key from the adversary to form the training data set and the system training does not need any data from victim users. We develop *TapLeak* using a desktop with Intel i7-8700K CPU and Nvidia GTX 2080Ti GPU. As stated in the system design, we train *TapLeak* for each type of the mobile phone to launch the attack. The training of the neural network for one group takes around 5 hours. Table 1 further shows the time consumption to infer one keystroke in different components (measured on the desktop), e.g., the average time cost is nearly 215 ms for one keystroke inference.

Methodology. To evaluate the system performance, we invite one volunteer as the adversary and other six volunteers (different from the adversary) to serve as the victim users including three males and three females. These victim users type on mobile phones and their keystrokes cover all the keys on the boards. We also consider different settings in the data collection, such as the ambient noise level, the typing speed, the angle how the user holds the phone, etc. We collect 4,680 keystrokes from victim users to evaluate the performance, and their data is not used to train *TapLeak* (all of them are used for the evaluation). In particular, each victim user follows the alphabet order to type the 26 letters and performs it 30 rounds with their daily typing behaviors. Because *TapLeak* segments the typing sound portion in the signal pre-processing step prior to the keystroke recognition (and there is no typing sound generated after the users finger finishes the current typing and before it touches the next key), we do not particularly design different typing orders in the data collection from the victim users.

Metrics. The main purpose of this paper is to reveal the typing privacy risk on mobile phones by using the tapping sounds only and avoiding the victim user's data for training. Existing works were studied mainly in different settings. Therefore, we focus on evaluating *TapLeak* in this section by using the following metrics.

Top-k Accuracy. A list of key candidates can be provided by *TapLeak* ordered by the distances to each cluster center (i.e., likelihood). Given the first k candidates, i.e., the candidates with the first k highest likelihood, we check whether the typed key (i.e., ground truth) is among them. Particularly, for n keystrokes, we define its top-k accuracy as $A_k = \frac{m}{n}$, where m is the number of inferences in which the top-k candidates contain the ground truth.

Confusion matrix. Each row represents each key on the keyboard, and each column represents each identified key by *TapLeak*. For example, for an entry located at the ith row and jth column, the reported value means that the percentage of the number of ith keys (i.e., the typed keys) are identified as the jth key by *TapLeak*, out of the total typed number for the ith key.

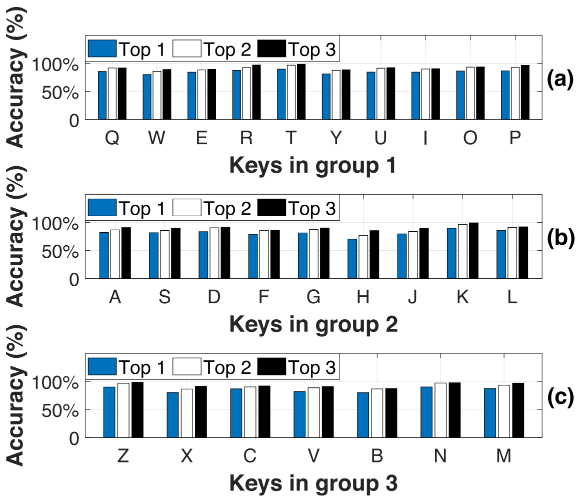

Fig. 13. Overall performance of *TapLeak* for each key.

4.2 Overall Performance

As stated in Section 2.2, in *TapLeak*, we view each row on the keyboard as one group and there are thus three groups in total. Fig. 13 shows the top-1 to top-3 accuracy of *TapLeak* for each key in the three groups. The average top-1 accuracy cross all three groups is 83.8%, and the top-2 and top-3 accuracy further increases to 89.7% and 92.2%, respectively. These results indicates the efficacy of the *TapLeak* design. Because the area that the user's finger touches the screen during typing is small, the characteristics from the vibration of the tapping spot dominates the feature of the generated sound. Thus, *TapLeak* can achieve a good performance even if it recognizes keystrokes in an unsupervised manner.

To understand how the recognition errors distribute, Fig. 14 further plots the confusion matrix. We find that the most wrongly identified keys are recognized as the neighboring keys in the same group. For example, a few keystrokes of key "Q" are identified as key "W", which is the neighbor of "Q" in the same group. The short distance between two adjacent keys could produce similar keystroke sounds with more similar MFCC features, which we believe is the main reason that causes the recognition error.

4.3 Impacts of System Components

Next, we investigate the impacts of the system component design choices on the performance.

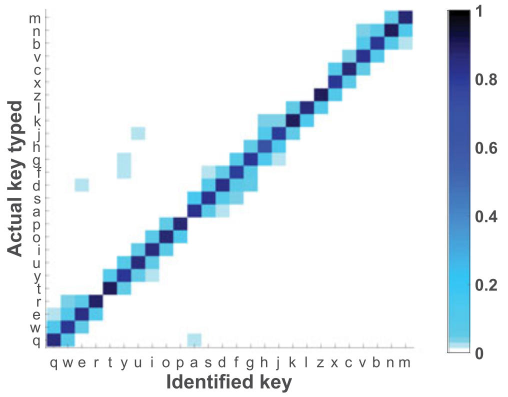

Fig. 14. Confusion matrix of each key in *TapLeak*.

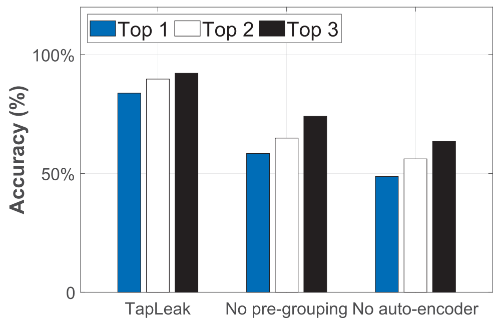

Fig. 15. Impacts of different system components.

Performance gains. *TapLeak* contains two major components: pre-grouping and auto-encoder based recognition. In Fig. 15, we first investigate the performance gain that each of these two components brings. In particular, we develop other two versions of *TapLeak*: "No pre-grouping" means that we always recognize all the keys at the same time, and "No auto-encoder" means that we apply the clustering to the MFCC of the sound clips directly. Fig. 15 shows that the top-1 to top-3 accuracy of *TapLeak* without pre-grouping drops to 58.4%, 64.9% and 74.1%, respectively. The reason is that the neighboring keys in different groups cannot be distinguished any more in the first place. Moreover, the top-1 to top-3 accuracy of *TapLeak* without auto-encoder (with pre-grouping) decreases to 48.7%, 56.1% and 63.5%, respectively. This significant reduction indicates that the auto-encoder can extract representative features effectively that are needed in the unsupervised clustering.

Different pre-grouping methods. In Section 3.3, we analyze that using two identified starting points from two microphones could cause larger errors compared with the cross-correlation. Fig. 16a compares their performance of the pre-grouping accuracy. We can see the cross-correlation improve the accuracy by 24.4% to 36.2%.

Different auto-encoder designs. We next study the impact of different auto-encoder designs. First, to fully modify the entire spectrum image to generate representative features, we propose to divide the spectrum image into several strips. With respect to this design, we compare it with the direct analysis using one 2D CNN applied to the entire

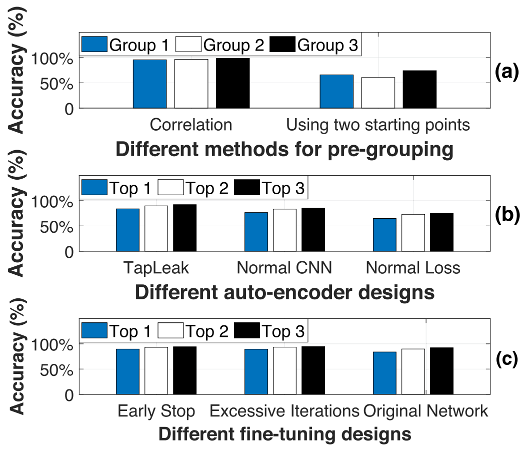

Fig. 16. Keystroke recognition accuracy (a) with different methods to calculate TDoA for pre-grouping, (b) with different auto-encoder designs, and (c) with fine-tuning.

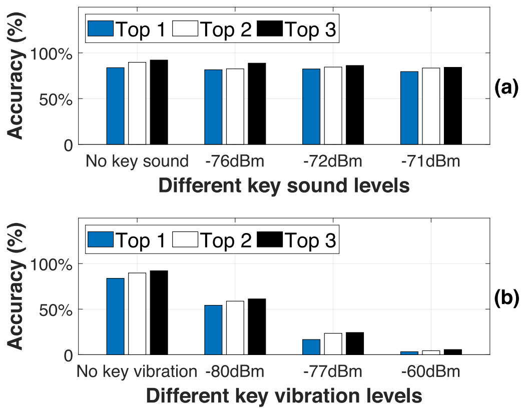

Fig. 17. Accuracy (a) with different electronic key sound levels, and (b) with different electronic vibration levels.

spectrum image ("Normal CNN"). Fig. 16b shows that the top-1, top-2 and top-3 accuracy of the system with auto-encoder using 2D CNN are 76.4%, 83.4% and 85.5%, respectively. Our design outperforms it by nearly 10%. On the other hand, we also propose two dedicated loss functions for the keystroke recognition. If we disable these loss functions using the recovery loss only ("Normal Loss"), Fig. 16b shows that the top-1, top-2 and top-3 accuracy of the system will be reduced to 64.8%, 73.0% and 74.8%, respectively. The accuracy reduction reaches about 20%, which indicates that our proposed loss functions can guide the auto-encoder to learn a better representative feature for the clustering task.

System fine-tuning. The adversary can further improve the performance after the victim's data is obtained with fine-tuning. To study its efficiency, we collect extra data from different users to fine-tune the network, and use the same inference data as the original network to test the accuracy. Fig. 16c compares the performances of the original network (without fine-tuning), the fine-tuned network terminated after excessive iterations (e.g., 10,000) and the fine-tuned network with our early stop design. The result shows that the top-1 to top-3 accuracy of the two fine-tuned networks achieve comparable performance and both of them can improve the original network by 2% to 6%. However, the early stop design only spends less than 1000 iterations to terminate the fine-tuning, which is much more efficient.

Electronic typing sound and vibration. As stated in Section 3.2, the electronic typing sound and vibration sound could be mixed up with the acoustic keystroke signal. We first examine the influences of these two factors to *TapLeak* under different energy levels. Fig. 17a shows the performance of *TapLeak* under different levels of the electronic typing sounds. The result confirms that the typing sounds do not impact the system performance because it has a relatively long delay and occurs after the keystroke signal ends. As for the vibration, we find that it can degrade the accuracy significantly. Specifically, Fig. 17b shows that the top-1 accuracy decreases to 54.2% at -80dBm power level of the motor vibration,and rapidly drops to 16.6% and 3.2% when the power increases to -77dBm and -60dBm, respectively.

To overcome the impact from the vibration sound, we propose a design to detect and then eliminate the vibration sound in Section 3.2. Fig. 18a shows that detection accuracy of our method compared with the threshold-based method. For the

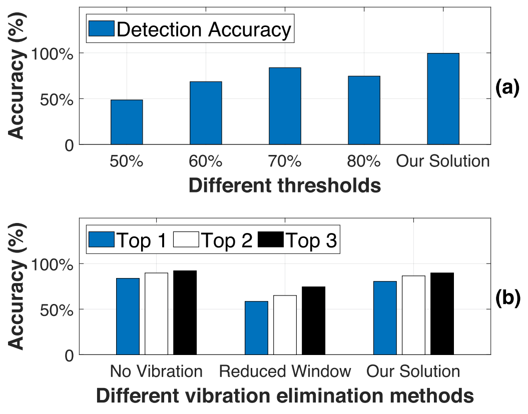

Fig. 18. (a) Vibration detection accuracy with different thresholds using naive method (b) Keystroke recognition accuracy with different elimination methods.

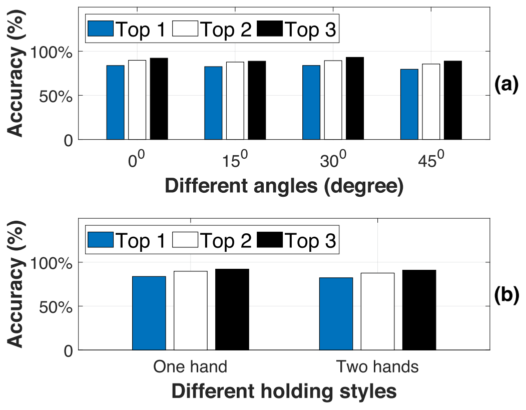

Fig. 20. Keystroke recognition accuracy (a) with various angles between the device screen and the horizontal plane, and (b) with different holding styles.

threshold-based method, the numbers in the x-axis mean the percentage of the power around the vibration frequency to the whole power of the signal. From the result, we find that our proposed method can detect the vibration sounds reliably, e.g., more than 99%, under different vibration levels, which outperforms the threshold-based detection method. For the threshold-based method, we have searched different thresholds to test the performance, and find that the accuracy can only reach up to 83%. Fig. 18b further shows that the top-1 to top-3 accuracy of *TapLeak* with our proposed vibration elimination method is 80.5%, 86.5% and 89.7%, respectively. It is comparable to the performance without any vibration sounds. Meanwhile, we compare it with the naive solution to directly reduce the segmentation window as stated in Section 3.2.1. After the window size is reduced, the size of the corresponding MFCC feature map reduces as well, causing the lost of some useful features. We hence adopt the neural network with a smaller structure that fits the size of the new MFCC feature map. Fig. 18b shows that the top-1 to top-3 accuracy of *TapLeak* with the setting of "Reduced Window" drops to 58.4%, 64.9% and 74.1%, respectively.

4.4 Other Micro-Benchmarks

Different users. Fig. 19a plots the top-1 to top-3 accuracy of *TapLeak* for the six different victim users in the experiments. We find that is robust and performs well among all the users,

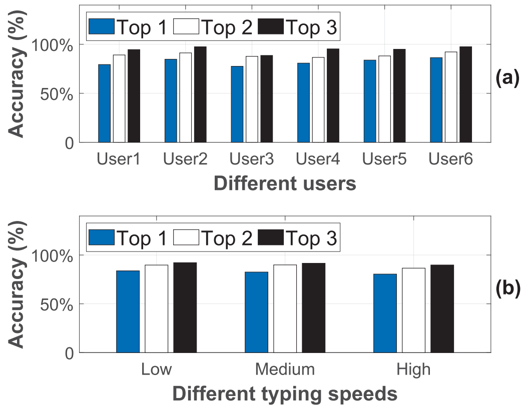

Fig. 19. Keystroke recognition accuracy (a) for different users and (b) under different typing speeds.

i.e., the average of top-1 accuracy is 82.1%, while top-2 and top-3 accuracy increase to 89.2% and 94.7%, respectively.

Different typing speeds. We next examine the impact of different typing speeds. To this end, we divide the typing speeds into three levels: *low speed*: around 60 keystrokes per minute (i.e., the typing speed of collecting data in previous experiments), *medium speed*: around 90 keystrokes per minute, and *high speed*: around 120 keystroke per minute. Because the duration of each tapping sound lasts around 40 ms, the interval between two consecutive typing (even with a relatively high speed) is much lager than this duration. Therefore, we can see from Fig. 19b that *TapLeak* can achieve comparable performance under different typing speeds.

Different device angles. Users may hold the mobile devices with different angles (between the device's body and the horizontal plane) during their typing. To investigate this impact, we examine the system performance under four different angles, including 0, 15°, 30°, and 45°, respectively. For 0, we put the phone on the desk, while for other degrees, the volunteer holds the phone in one hand and uses the other hand to type on the screen. Fig. 20a reports the performance. We find that this factor does not impact the *TapLeak*'s performance significantly, e.g., the top-1 accuracy varies from 79.6% to 83.8% in the experiment.

Different holding styles. So far, the victim users hold the phone using one hand. In this experiment, we further examine the system performance when they hold the phone with two hands. Fig. 20b shows that the top-1 to top-3 accuracy is from 82.3% to 90.9% when the phone is held with two hands, which are comparable to the performance when one hand is used. This is understandable as a smartphone may have heterogeneous densities and structures at different places inside the device. So, the features are different when the user's finger taps different positions on the phone, which are loosely related to the number of hands holding the phone.

Different holding tightness levels. In this experiment, we further investigate the impact of the tightness on the performance by considering user and attacker both side. Fig. 21 shows the results for each case. For instance, in Fig. 21a, the training data is collected when the attacker holds the phone *loosely*, and the three sets of bars represent the top-1 to top-3 accuracy when the user holds the phone loosely (80.4% ~85.9%), moderately (79.5%~82.7%) and tightly (78.9%

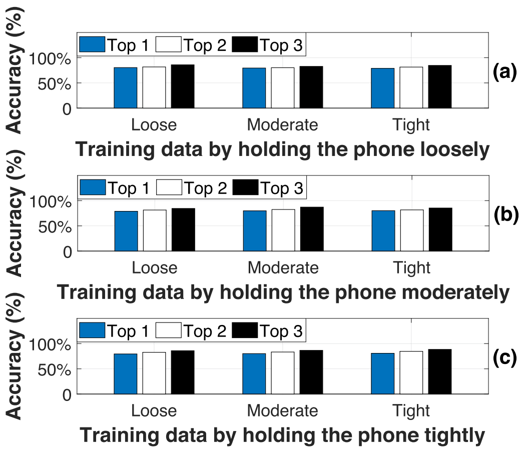

Fig. 21. Performance when the attacker holds the phone (a) loosely, (b) moderately, and (c) tightly. In each sub-figure, the three sets of bars represent the accuracy under different tightness levels of the user's holding.

~84.6%), respectively. In Fig. 21b, when the training data is collected by holding the phone *moderately*, the top-1 to top-3 accuracy for the three tightness levels of the user's holding is 78.9%~84.5%, 79.9%~87.2% and 80.1%~85.4%, respectively. Finally, in Fig. 21c, when the training data is collected by holding the phone *tightly*, the top-1 to top-3 accuracy for the three tightness levels of the user's holding is 79.5%~85.9%, 80.1%~86.5% and 80.8%~88.5%, respectively. We find that the data tested under a similar condition as the collection leads to a slightly better performance, while the difference is not significant, which suggests that our system is robust to different holding tightness levels.

Different background noise levels. To understand how the background noise influences the performance of *TapLeak*, we conduct the experiment in three real environments with different background noise levels, e.g., a quiet library (35 dBSPL noise), a less noisy office (55 dBSPL noise) and a noisy canteen (70 dBSPL noise). Fig. 22a shows the top-1 to top-3 accuracy in the above three environments. We find that the performance of *TapLeak* decreases with the increase of the background noise level. For example, the top-1 accuracy in the quiet library is 90.2%, and reduces to 83.8% and 65.9% in the office and in the canteen, respectively. We observe that some unpredictable loud sounds in these environments, e.g., the people's talking and moving chairs nearby, can overwhelm the recorded tapping sounds,

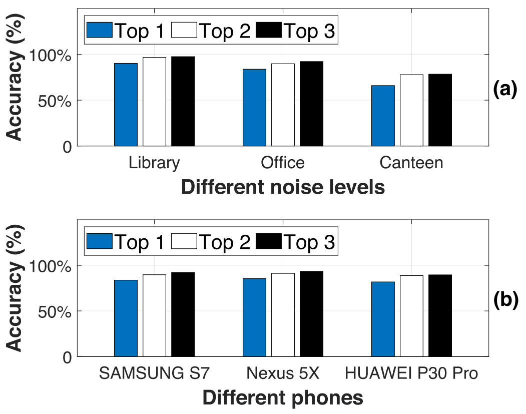

Fig. 22. Performance (a) in different environments with different noise levels, and (b) with different types of mobile phones.

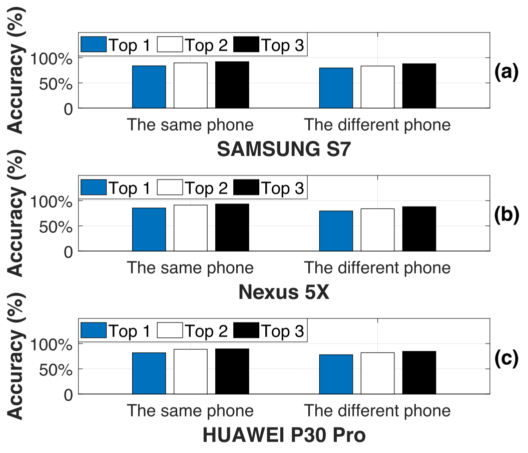

Fig. 23. Performance obtained on the same phone and on a different phone (of the same model) for (a) SAMSUNG S7, (b) Nexus 5X and (c) HUAWEI P30 Pro.

which in turn degrade the keystroke recognition performance. Although we have de-noise designs to improve SNR, the background noise is still the main limitation of *TapLeak* as discussed in Section 5.

Different types of mobile phones. Next, we examine the *TapLeak*'s performance on three mobile phones with different screen sizes (e.g., a 5.1-inches Samsung Galaxy S7 with Android 7.0, a 5.2-inches Nexus 5X with Android 6.0, and a 6.47-inches Huawei P30 Pro with Android 10.0). The sampling rates of the microphones are still set as 192 KHz for all the three mobile phones, and the number of groups in the pre-grouping step remains to be three. However, the representative hyperbolas for the three groups are different on different phones due to their different screen sizes and key sizes. In Fig. 22b, we plot the top-1 to top-3 accuracy of the keystroke recognition on each mobile phone, where *TapLeak* can achieve a stable performance cross all three types of the mobile phones.

Different devices of the same model. In our previous experiments, the training data were collected on the same phone (for the same model) as in the testing. In this experiment, we further investigate the impact of the manufacturing differences for the devices of the same model. Fig. 23 shows the results, where the top-1 to top-3 accuracy ("The different phone") for the three new phones is 79.6%~88.1%, 79.5%~88.1% and 77.9%~84.6%, respectively. Compared to the performance when the training data is collected from the same device, the recognition accuracy slightly reduces about 5%. We find that although the manufacturing differences cannot be avoided, its impact to the system is limited. We explain the reason as the pre-grouping step by adopting the TDoA measurement is robust to the manufacturing difference. After the pre-grouping, the keystroke recognition only needs to be performed among seven to ten keys. The manufacturing differences may indeed cause certain feature differences for the same keystroke signal from different devices, while the neural network based recognizer can tolerate many of these differences and distinguish such a relatively small amount of keys from each group well.

Different layouts. The previous experiments are conducted in the portrait mode. In this experiment, we further investigate the *TapLeak*'s performance in the landscape mode.

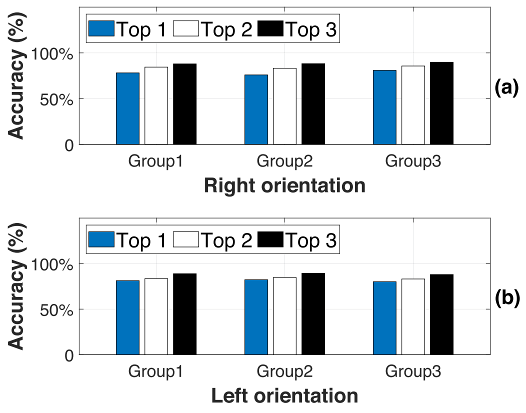

Fig. 24. Keystroke recognition accuracy in the landscape mode (a) with right-orientation and (b) left-orientation poses.

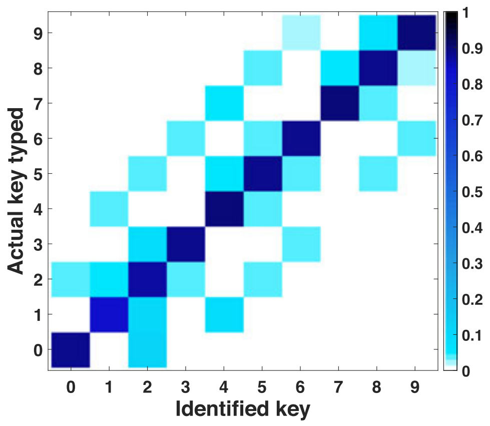

Fig. 25. Confusion matrix for PIN number keyboard.

Similar as the portrait mode, we also divide all the keys into three groups for the landscape mode. Due to the layout change between these two modes, each of the three new groups in the landscape mode contains the following keys:

- Group 1: “q”,“w”,“e”,“r”,“a”,“s”,“d”,“z”,“x”,“c”
- Group 2: “t”,“y”,“f”,“g”,“h”,“v”,“b”
- Group 3: “u”,“i”,“o”,“p”,“j”,“k”,“l”,“n”,“m”

Fig. 24 summarizes the accuracy achieved for each group in the landscape mode. The landscape mode has two different orientations. In the right-orientation pose (the front camera is on the right-hand side), the top-1 to top-3 accuracy is 78.1%∼87.9%, 75.9%∼88.2% and 80.8%∼89.8%, respectively. The left-orientation pose (the front camera is on the left-hand side) achieves a similar performance. Because *TapLeak* recognizes the keystrokes following the same processing pipeline, we do not observe obvious performance differences compared with the portrait mode, e.g., the average top-1 to top-3 accuracy in the portrait mode is 83.8%, 89.7% and 92.2%, respectively.

PIN number keyboard. We also examine *TapLeak* on the popular PIN number keyboard. As the PIN number only contains ten keys, we omit the pre-grouping step and directly compute the MFCC feature map to feed the network. Fig. 25 shows the confusion matrix of the inference for PIN number keyboard. The result shows that there are incorrect inferences between keys located on different rows due to the lack of pre-grouping step, while the average performance of the top-1 accuracy across all ten keys is 88.9%, similar to the twenty-six keyboard setting. Although the number of keys on the PIN number keyboard is less than the twenty-six keyboard, neural networks recognize a similar number of keys each time, e.g., with the grouping for the twenty-six keyboard, the number keys in each group is from seven to ten. As a result, the difficulties of these two recognition problems are similar and their performance is comparable to each other. In contrast, we have evaluated the performance of *TapLeak* when the twenty-six letters are recognized directly in Section 4.3, wherein the top-1 to top-3 accuracy drops to 58.4%, 64.9% and 74.1%, respectively.

5 Points of Discussions

Finally, we discuss the possible mechanisms to defend this attack and the limitations of our current design.

5.1 Possible Defense Mechanisms

Permission management. The successful launch of the attack requires a malicious APP executing in the background with the permissions to access the microphones and Internet. However, it is impossible to deny the incoming applications of these permissions for all APP because legal APP may also require them, e.g., voice recognition, online game, etc. To defend such an attack, people can develop a fine-grained permission management module in the smartphone operating system to temporarily prohibit the vulnerable permissions of APP running in the background, e.g., to access the microphones, Internet, etc., when the keyboard is activated to use. As the prohibition is temporary merely and will be recovered after the keyboard is inactivated, the influence to the normal usage of a legal APP is minimum.

Strong background noise. The result in Section 4.4 shows that the performance of *TapLeak* reduced to 65.9% in a noisy environment indicating that the background noise has a significant influence to the system. Hence, when a user types in a noisy environment, she has a lower chance to suffer from this attack. Inspired by this observation, the user may even proactively play some music on the phone during inputting the sensitive information. The loud acoustic signal can cover the keystroke sound and impact the *TapLeak* attack launched by the adversary.

Random keyboard layout. Considering that the adversary assumes the target mobile phone with a regular keyboard and hence is aware of the positions of each key, the user can also adopt the more advanced keyboard that allows to shuffle the positions of each key to achieve a random when the sensitive information is going to be typed on the phone. Even the adversary can recognize the location of the user finger's tapping, the corresponding key changes due to the random layout of the keys, producing meaningless information for the adversary.

Frequency varying of the e-sound. The latency of the e-sound is relatively large on current mobile phones, e.g., approximately 100 ms, and the e-sound has thus been excluded by the segmentation time-window of *TapLeak* in the first place. In the future, with the hardware advance and software optimization, it is possible that such a latency can be much shortened, so that the tapping sound and the e-sound can be overlapped in the time domain. In this case, a dynamic frequency varying of the e-sound can serve as a potential mechanism to defend this attack, while the user needs to get used to that the e-sound of a key becomes different each time.

5.2 Limitations

Pre-knowledge about the victim user. Our current design requires some prior knowledge of the victim user, e.g., the types of the phone and keyboard, microphone positions, and the portrait or landscape mode, so that the adversary can know the keyboard size and also determine certain system parameters, e.g., the latency of the e-sound, the frequency of the v-sound, the orientation of the phone, etc. One important future work is to study whether a more general design is viable by further releasing these requirements.

Noises. In this paper, we consider the impact from the background noise, but we assume that there are no other strong ambient noises that dominate the recorded sounds. We have designed pre-processing step to improve the SNR and pre-grouping mechanism to reduce the classification difficulty. However, strong noises from environments could still degrade the performance of *TapLeak* significantly as shown in Section 4.4. Therefore, another meaningful future work of this paper is to investigate more powerful signal processing and recognition techniques to enhance the current design, so that the system performance can be further improved in the more noisy environments.

6 Related Work

Inferring user's keystrokes. In the literature, there are some existing efforts made to infer the user's keystrokes on a mobile phone using various sensors on the same device. For instance, TouchLogger [25] and TapLogger [6] utilizes on-board motion sensors to infer keystrokes on numeric keyboards. TapPrints [26] expands the inference area to any location on the screen. To further improve the performance, PIN Skimmer [27] uses microphone to detect the keystroke events and camera to estimate the slant of the phone caused by the tapping action. Of course, we are not the first one to look at the keystroke recognition problem through the user's finger tapping sounds. Narain *et al.* [12] utilizes a set of on-board sensors, including microphones, to realize a keystroke recognition design. Shumailov *et al.* [28] utilizes the TDoA measurements of the acoustic signals between the two microphones to recognize the keystrokes by drawing hyperbolas on the screen of the smartphone. Using the TDoA measurements merely may not be robust enough because the result can be impacted by the background noise significantly. In addition, the impacts from the electronic typing sound and vibration need to be carefully addressed as well. Instead, we design an efficient pre-processing module to improve the SNR of the raw acoustic signal and use the TDoA measurements for the coarse-grained pregrouping only, which could alleviate the impact of the background noise. Under a similar setting, TapSnoop [29] improves the accuracy with adaptive preprocessing and more complex classification model. Compared to our system, TapSnoop requires enough data from the victim. We take one more step to show the possibility to leak user's typing privacy through the tapping sounds only with a comprehensive unsupervised design, so as to reveal (more importantly alarm people) the further privacy leakage risk that may not be viable before.

Recently, researchers also investigate the possibilities to infer user's keystrokes on an external keyboard (e.g., keyboard of a computer) by using one mobile phone (close to the victim) to record the keystroke sounds directly [20], [30], [31], [32], [33] or transmit inaudible sounds first and then record the reflected sounds by user's finger in the typing [34]. On the other hand, some researchers also study the keystroke inference on mobile phones [35], [36] or laptops [37] through some external devices, e.g., smart watch on the user's wrist, nearby camera [38], etc. However, these existing works do not address the unique challenges solved in designing *TapLeak*.

Tracking and sensing designs using acoustic signals. Recent studies propose to play acoustic signals to achieve an accurate tracking of the user's motion. LLAP [39] can achieve a high-quality finger tracking on a 2-D plane. VSkin [10] combines the structure-borne and air-borne sounds to sense the user's finger typing or movement at the back of a mobile phone. Mao *et al.* [4] further utilize RNNs to achieve a room-scale hand motion tracking. RobuCIR [40] efficiently improve the accuracy and robustness for the system to track the hand movements and recognize hand gestures with pure acoustic signal. SonicPrint [41] extends the fingerprint identification by sensing the distinct fingerprint-induced sonic effect produced by the swiping of the user's finger on the smart devices with the microphone. SpiroSonic [42] utilizes the acoustic sensing to measure the human's chest wall motion with the speaker and the microphones from a mobile phone. Zhou *et al.* [3] predicts the bus arrival time with the combination of acoustic signal and other kinds of signals. In addition to the tracking of the user's motion, some other works also study the tracking or ranging for another mobile device [43], [44], [45], [46]. Different from these works, *TapLeak* focuses on inferring a user's keystrokes when the user types on a mobile device.

Auto-encoder in deep learning. The auto-encoder framework is widely used to explore a better representation of the input data in the deep learning domain [8]. Existing works utilize auto-encoder for dimensionality reduction [47], feature extraction [48], recommendation system design [49], and image compression [21]. Moreover, some works [24], [50] embed the deep auto-encoder into a clustering procedure to learn the best representation for the clustering task. To address the unsupervised keystroke recognition issue, we propose a tailored network and dedicated loss functions to integrate the clustering ability with auto-encoder.

7 Conclusion

This paper presents *TapLeak* to demonstrate (more importantly alarm people) a crucial typing privacy leakage risk through microphones on mobile phones only. We propose effective solutions to address weak signals, interference from electronic typing sound and vibration, and unsupervised keystroke recognition three major design issues. We implement our designs in a prototype. Extensive experiments indicate the efficacy of *TapLeak* by attacking more than 4000 keystrokes from different users on various phones. A preliminary version of this study has been published in [51].

REFERENCES

[1] W. Jin, M. J. Taghizadeh, K. Chen, and W. Xiao, "Multi-channel noise reduction for hands-free voice communication on mobile phones," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2017, pp. 506–510.

[2] P. Xie, J. Feng, Z. Cao, and J. Wang, "GeneWave: Fast authentication and key agreement on commodity mobile devices," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1688–1700, Aug. 2018.

[3] P. Zhou, Y. Zheng, and M. Li, "How long to wait? predicting bus arrival time with mobile phone based participatory sensing," in *Proc. ACM 10th Int. Conf. Mobile Syst. Appl. Services*, 2012, pp. 379–392.

[4] W. Mao, M. Wang, W. Sun, L. Qiu, S. Pradhan, and Y.-C. Chen, "RNN-based room scale hand motion tracking," in *Proc. ACM 25th Annu. Int. Conf. Mobile Comput. Netw.*, 2019, pp. 1–16.

[5] J. Liu, C. Wang, Y. Chen, and N. Saxena, "VibWrite: Towards finger-input authentication on ubiquitous surfaces via physical vibration," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 73–87.

[6] Z. Xu, K. Bai, and S. Zhu, "TapLogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors," in *Proc. ACM Conf. Secur. Privacy Wireless Mobile Netw.*, 2012, pp. 113–124.

[7] F. Gustafsson and F. Gunnarsson, "Positioning using time-difference of arrival measurements," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2003, vol. 6, pp. 553–556.

[8] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, Aug. 2013.

[9] G. Zhang, C. Yan, X. Ji, T. Zhang, T. Zhang, and W. Xu, "DolphinAttack: Inaudible voice commands," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 103–117.

[10] K. Sun, T. Zhao, W. Wang, and L. Xie, "VSkin: Sensing touch gestures on surfaces of mobile devices using acoustic signals," in *Proc. ACM 24th Annu. Int. Conf. Mobile Comput. Netw.*, 2018, pp. 591–605.

[11] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets," in *Proc. 19th Annu. Netw. Distrib. Syst. Secur. Symp.*, 2012, pp. 50–52.

[12] S. Narain, A. Sanatinia, and G. Noubir, "Single-stroke language-agnostic keylogging using stereo-microphones and domain specific machine learning," in *Proc. ACM Conf. Secur. Privacy Wireless Mobile Netw.*, 2014, pp. 201–212.

[13] P. Scalart and J. V. Filho, "Speech enhancement based on a priori signal to noise estimation," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 1996, pp. 629–632.

[14] T. Kinnunen and H. Li, "An overview of text-independent speaker recognition: From features to supervectors," *Speech Commun.*, vol. 52, no. 1, pp. 12–40, 2010.

[15] Z. Tüske, P. Golik, R. Schlüter, and H. Ney, "Acoustic modeling with deep neural networks using raw time signal for LVCSR," in *Proc. Annu. Conf. Int. Speech Commun. Assoc.*, 2014, pp. 890–894.

[16] M. Likitha, S. R. R. Gupta, K. Hasitha, and A. U. Raju, "Speech based human emotion recognition using MFCC," in *Proc. IEEE Int. Conf. Wireless Commun. Signal Process. Netw.*, 2017, pp. 2257–2260.

[17] K. S. Ahmad, A. S. Thosar, J. H. Nirmal, and V. S. Pande, "A unique approach in text independent speaker recognition using MFCC feature sets and probabilistic neural network," in *Proc. IEEE 8th Int. Conf. Advances Pattern Recognit.*, 2015, pp. 1–6.

[18] A. Maurya, D. Kumar, and R. Agarwal, "Speaker recognition for hindi speech signal using MFCC-GMM approach," *Elsevier Procedia Comput. Sci.*, vol. 125, pp. 880–887, 2018.

[19] A. K. H. Al-Ali , D. Dean, B. Senadji, V. Chandran, and G. R. Naik, "Enhanced forensic speaker verification using a combination of DWT and MFCC feature warping in the presence of noise and reverberation conditions," *IEEE Access*, vol. 5, pp. 15400–15413, 2017.

[20] J. Liu, Y. Wang, G. Kar, Y. Chen, J. Yang, and M. Gruteser, "Snooping keystrokes with mm-level audio ranging on a single phone," in *Proc. ACM 21st Annu. Int. Conf. Mobile Comput. Netw.*, 2015, pp. 142–154.

[21] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto, "Deep convolutional autoencoder-based lossy image compression," in *Proc. IEEE Picture Coding Symp.*, 2018, pp. 253–257.

[22] J. Walker, C. Doersch, A. Gupta, and M. Hebert, "An uncertain future: Forecasting from static images using variational autoencoders," in *Proc. Springer Eur. Conf. Comput. Vis.*, 2016, pp. 835–851.

[23] K. Wagstaff, C. Cardie, S. Rogers, S. Schrödl, "Constrained K-means clustering with background knowledge," in *Proc. 18th Int. Conf. Mach. Learn.*, 2001, pp. 577–584.

[24] C. Song, F. Liu, Y. Huang, L. Wang, and T. Tan, "Auto-encoder based data clustering," in *Proc. Springer Progress Pattern Recognit. Image Anal. Comput. Vis. Appl.*, 2013, pp. 117–124.

[25] L. Cai and H. Chen, "TouchLogger: Inferring keystrokes on touch screen from smartphone motion," *USENIX Summit Hot Top. Secur.*, vol. 11, no. 2011, pp. 9–14, 2011.

[26] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. R. Choudhury, "TapPrints: Your finger taps have fingerprints," in *Proc. ACM 10th Int. Conf. Mobile Syst. Appl. Services*, 2012, pp. 323–336.

[27] L. Simon and R. Anderson, "PIN skimmer: Inferring pins through the camera and microphone," in *Proc. ACM 3rd Workshop Secur. Privacy Smartphones Mobile Devices*, 2013, pp. 67–78.

[28] I. Shumailov, L. Simon, J. Yan, and R. Anderson, "Hearing your touch: A new acoustic side channel on smartphones," 2019, *arXiv: 1903.11137*.

[29] H. Kim, B. Joe, and Y. Liu, "TapSnoop: Leveraging tap sounds to infer tapstrokes on touchscreen devices," *IEEE Access*, vol. 8, pp. 14737–14748, 2020.

[30] D. Asonov and R. Agrawal, "Keyboard acoustic emanations," in *Proc. IEEE Symp. Secur. Privacy*, 2004, pp. 3–11.

[31] L. Zhuang, F. Zhou, and J. D. Tygar, "Keyboard acoustic emanations revisited," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 1, pp. 1–26, 2009.

[32] T. Zhu, Q. Ma, S. Zhang, and Y. Liu, "Context-free attacks using keyboard acoustic emanations," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 453–464.

[33] Y. Berger, A. Wool, and A. Yeredor, "Dictionary attacks using keyboard acoustic emanations," in *Proc. ACM 13th Conf. Comput. Commun. Secur.*, 2006, pp. 245–254.

[34] L. Lu *et al.*, "KeyListerber: Inferring keystrokes on QWERTY keyboard of touch screen through acoustic signals," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 775–783.

[35] C. Wang, X. Guo, Y. Wang, Y. Chen, and B. Liu, "Friend or foe?: Your wearable devices reveal your personal PIN," in *Proc. ACM 11th Asia Conf. Comput. Commun. Secur.*, 2016, pp. 189–200.

[36] Y. Liu and Z. Li, "aLeak: Privacy leakage through context-free wearable side-channel," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 1232–1240.

[37] H. Wang, T. T.-T. Lai, and R. Roy Choudhury , "Mole: Motion leaks through smartwatch sensors," in *Proc. ACM 21st Annu. Int. Conf. Mobile Comput. Netw.*, 2015, pp. 155–166.

[38] G. Ye *et al.*, "Cracking android pattern lock in five attempts," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2017, pp. 130–144.

[39] W. Wang, A. X. Liu, and K. Sun, "Device-free gesture tracking using acoustic signals," in *Proc. ACM 22nd Annu. Int. Conf. Mobile Comput. Netw.*, 2016, pp. 82–94.

[40] Y. Wang, J. Shen, and Y. Zheng, "Push the limit of acoustic gesture recognition," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 566–575.

[41] A. S. Rathore *et al.*, "SonicPrint: A generally adoptable and secure fingerprint biometrics in smart devices," in *Proc. ACM 18th Int. Conf. Mobile Syst. Appl. Services*, 2020, pp. 121–134.

[42] X. Song *et al.*, "SpiroSonic: Monitoring human lung function via acoustic sensing on commodity smartphones," in *Proc. ACM 26th Annu. Int. Conf. Mobile Comput. Netw.*, 2020, pp. 1–14.

[43] C. Peng, G. Shen, Y. Zhang, Y. Li, and K. Tan, "BeepBeep: A high accuracy acoustic ranging system using COTS mobile devices," in *Proc. ACM 5th Int. Conf. Embedded Netw. Sensor Syst.*, 2007, pp. 1–14.

[44] S. P. Tarzia, P. A. Dinda, R. P. Dick, and G. Memik, "Indoor localization without infrastructure using the acoustic background spectrum," in *Proc. ACM 9th Int. Conf. Mobile Syst. Appl. Services*, 2011, pp. 155–168.

[45] J. Yang *et al.*, "Detecting driver phone use leveraging car speakers," in *Proc. ACM 17th Annu. Int. Conf. Mobile Comput. Netw.*, 2011, pp. 97–108.

[46] W. Mao, J. He, and L. Qiu, “CAT: High-precision acoustic motion tracking,” in *Proc. ACM 22nd Annu. Int. Conf. Mobile Comput. Netw.*, 2016, pp. 69–81.
[47] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Sci.*, vol. 313, no. 5786, pp. 504–507, 2006.
[48] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Extracting and composing robust features with denoising autoencoders,” in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 1096–1103.
[49] X. Li and J. She, “Collaborative variational autoencoder for recommender systems,” in *Proc. ACM 23rd SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2017, pp. 305–314.
[50] X. Guo, X. Liu, E. Zhu, and J. Yin, “Deep clustering with convolutional autoencoders,” in *Proc. Springer Int. Conf. Neural Inf. Process.*, 2017, pp. 373–382.
[51] Z. Xiao, T. Chen, Y. Liu, and Z. Li, “Mobile phones know your keystrokes through the sounds from finger’s tapping on the screen,” in *Proc. IEEE 40th Int. Conf. Distrib. Comput. Syst.*, 2020, pp. 965–975.

Zhen Xiao (Student Member, IEEE) received the BE degree from the City University of Hong Kong, Hong Kong, in 2018. He is currently working toward the PhD degree with the Department of Computer Science, City University of Hong Kong, Hong Kong. His research interests include wireless sensing, deep learning, and cyber security.

Tao Chen received the BE degree from Northwestern Polytechnical University, Xian, China, in 2017, and the PhD degree from the City University of Hong Kong, Hong Kong, China, in 2021. His research interests include signal processing, wireless and machine learning, and cyber security.

Yang Liu (Member, IEEE) received the BE degree from Xi’an Jiaotong University, Xi’an, China, in 2016 and the PhD degree from the City University of Hong Kong, Hong Kong, China, in 2020. She is currently a postdoctoral with the Department of Computer Science, City University of Hong Kong. She is working on developing intelligent mobile and wearable sensing technologies from IoT systems, machine learning, and cyber security.

Jiao Li (Student Member, IEEE) received the BE degree from the School of Software Engineering, Xidian University, Xian, China, in 2018. She is currently working toward the PhD degree with the Department of Computer Science, City University of Hong Kong, Hong Kong, China. Her research interests include cyber security, mobile computing, and deep learning.

Zhenjiang Li (Member, IEEE) received the BE degree from Xi’an Jiaotong University, Xi’an, China, in 2007, and the MPhil and PhD degrees from the Hong Kong University of Science and Technology, Hong Kong, China, in 2009 and 2012, respectively. He is currently an associate professor with the Department of Computer Science, City University of Hong Kong. His research interests include Internet of Things, wearable and mobile computing, smart health, deep learning, and distributed computing.