# Mobile Phones Know Your Keystrokes through the Sounds from Finger's Tapping on the Screen

Zhen Xiao*, Tao Chen*, Yang Liu*, Zhenjiang Li*†
*Department of Computer Science, City University of Hong Kong
†City University of Hong Kong Shenzhen Research Institute

*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 infer 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.**

## I. Introduction

Mobile phones nowadays are commonly equipped with (at least) dual microphones [1]. 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 [2]. However, 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 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. 1(a), 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 [3], [4], [5]. This keystroke recognition ability could be further leveraged in the future to develop a new input modality on a ubiquitous surface [6] 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 §II-C) and the barrier to launch this hacking is not high [7]. 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.

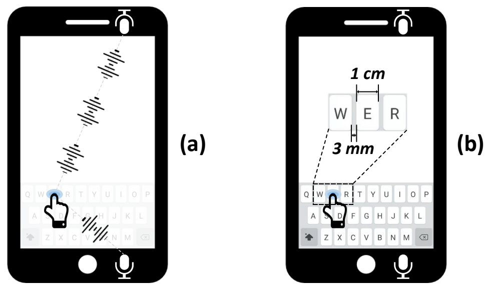

Fig. 1: **(a) The sounds generated from the vibration (of the touching spot on the device) caused by finger's tapping on the screen can be captured by microphones. (b) Typical length of a key and distance between two adjacent keys.**

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, or the device vibration triggered by the keyboard software and operating system (attack model is in Section II). However, it is indeed also 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 demonstrate this keystroke recognition ability, we need to address the following two 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. 1(b)) are short normally, *e.g.*, around 1 cm and 3 mm [7], 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 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) 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) [8] 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.

On the other hand, we leverage the auto-encoder framework from the deep learning domain [9] 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 that could lead to the desired number of clusters, and the preserved features tend to be more user independent finally.

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.

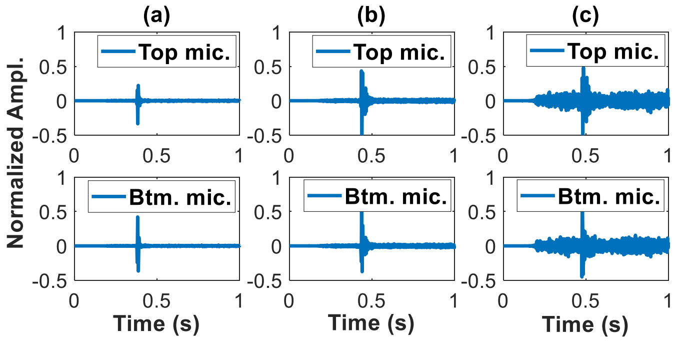

Fig. 2: **The sounds generated due to the finger's tapping on the screen received by the top-microphone (the 1st row) and the bottom-microphone (the 2nd row) on a mobile phone (Samsung Galaxy S7) in three environments with different background noise levels: (a) a quiet library (35 dBSPL noise), (b) a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

- We propose effective techniques to address the weak acoustic signals and unsupervised keystroke recognition two 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 II. The *TapLeak* design is detailed in Section III, and the system evaluation is conducted in Section IV. We review related works in Section V before the conclusion in Section VI.

## II. Preliminary and Attack Model

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

### *A. 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 [10] 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 III.

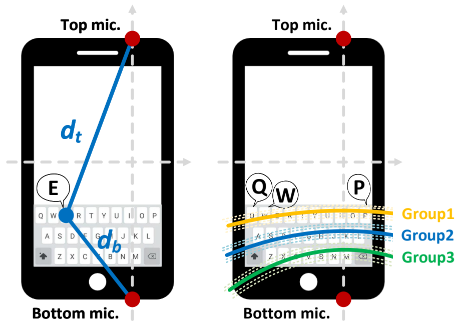

Fig. 3: **(a) Illustration of the TDoA measure for tapping key "E". (b) Representative hyperbolas for each group.**

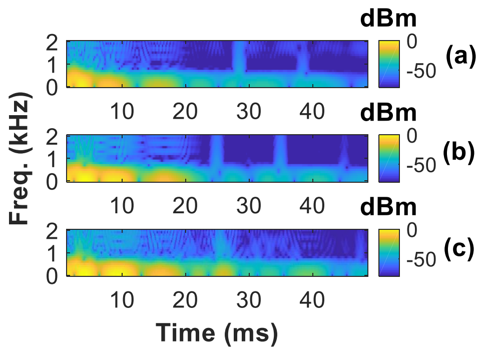

Fig. 4: **Spectrum of the keystrokes on keys (a) "Q", (b) "W" and (c) "P" in one group, respectively.**

## B. Distinct Sound Features

For these detectable sound waves, we further 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 [11]. Since the keyboard is closer to the bottom-microphone usually, when a user types on the screen, there exists a time different of arrival (TDoA) for the generated sound received by two microphones, *i.e.*, $\Delta t = \frac{d_t \quad d_b}{v}$, where $d_t$ and $d_b$ (in Fig. 3(a)) 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 I), 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. 3(b) 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 the adversary 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 III).

## C. 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 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 victim's mobile phone and the keyboard, so as to know the phone's size and select the representative hyperbolas for each group in advance, *e.g.*, adversary can peep in the victim's vicinity. Moreover, in this paper, we focus on the keystroke inference on the standard English keyboard from mobile devices in a portrait orientation.

*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 inference noises from the environment.

*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. These functions can be disabled in the device setting. In our current design, we primarily assume they are disabled, while we also investigate their impacts through experiments in Section IV.

*4) Hacking microphone data.* We assume that the adversary can access two microphones of the victim's phone to collect the microphone data and send them out. To this end, the adversary can develop a malicious APP as a Trojan [12]. The adversary can disguise the Trojan as some useful legitimate APP or game and publish it to the APP market to fool the

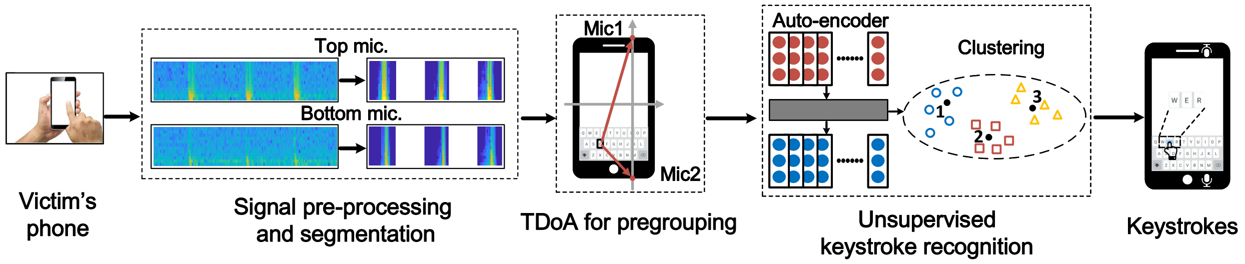

Fig. 5: **Illustration of the working flow to infer the user's keystrokes in *TapLeak* with three main steps.**

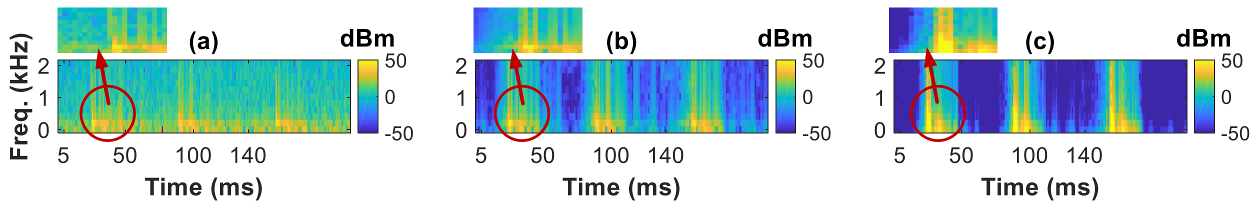

Fig. 6: **Pre-processing to identify the starting point for each keystroke. Spectrum of (a) the raw sound wave, (b) the sound after Wiener filter and (c) the sound after Wiener filter plus a further difference amplifying.**

victim to install, which provides the functions, *e.g.*, voice recognition, to gain the permission to access microphones [13] during the installation of the Trojan APP. Once it is installed, this APP listens in the background and sends collected microphone data to the adversary.

## III. Design of *TapLeak*

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

*Signal 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.

*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.

*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.

### A. 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). So the adversary obtains a stream of sound clip pairs (from two microphones), which will be used for the pre-grouping module in Section III-B.

*1) Design challenge:* The challenge in this 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. 6(a) 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 due to the noise.

As stated in Section I, 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.

*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 [14] 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. 6(b) 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. 6(b) 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

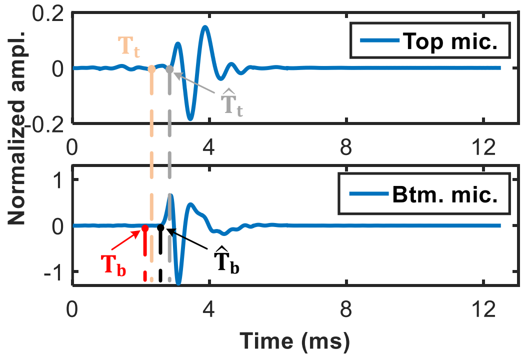

Fig. 7: **Analysis of the TDoA error if we compute TDoA using the search starting points directly. The amplitude of the sound from the bottom-microphone is stronger because user's tapping position is closer to this microphone.**

amplitude (*e.g.*, <1 for noise) even smaller:

$$s^0(t) \quad = \quad \sum\nolimits_{n=t}^{t+W} s^2(n), \tag{1}$$

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. 6(c) shows the spectrum for $s^0(t)$, 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. 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.

### *B. Pre-grouping based on TDoA*

As aforementioned in Section II-B, we divide the keyboard keys into three groups and recognize keystrokes inside one group only, to avoid recognizing all the keys each time[1]. 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. 3(b), 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 utilize the neural network of this group to conduct the keystroke recognition in Section III-C.

Although we can record the sounds for both microphones at the same time [11] 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

$$TDoA \quad = \quad (\hat{T}_t - \hat{T}_b) + (E_t - E_b). \tag{2}$$

In fact, the error from the second term $E_t - E_b$ can be further minimized. To this end, we can fix the calculated starting point for one microphone, and move the sound clip from the other microphone to perform the cross-correlation [8]. 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. So the cross-correlation basically introduces the starting point search error only once. Through our experiment in Section IV, this can improve the pre-grouping accuracy by 30%.

### *C. 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.

*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) [15] of the acoustic signal as input [16].
- **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:

$$L_{rec} = \frac{1}{N} \sum\nolimits_{t=1}^{N} f(x_i) - h(f(x(t))), \tag{3}$$

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.

[1]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 IV.

*2) Unsupervised recognition with auto-encoder:* With the auto-encoder framework stated above, we now introduce our recognition design in *TapLeak*.

**Encoder and decoder designs.** The auto-encoder framework is widely utilized in the computer vision domain [17], wherein the input is the 2D image. Therefore, the encoder and decoder are usually implemented by a multi-layer perceptron or 2D convolutional neural network (CNN) [18]. For the acoustic signal, we can apply the short-term Fourier transform (STFT) and then compute its MFCC to obtain its MFCC spectrum (Fig. 8), which can also be viewed as a 2D "image":

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 perceptron or 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 $i^{th}$ input strip, and the corresponding CNN $f^i(x^i)$ can be expressed as:

$$f^i(x^i) = \sum_{j=1}^{n} \sum_{k=1}^{m} x^i_{j,k} \cdot w_k, \qquad (4)$$

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 IV, we find this encoder and decoder design can effectively improve the recognition performance (Section IV).

**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 whose data is used in the system 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

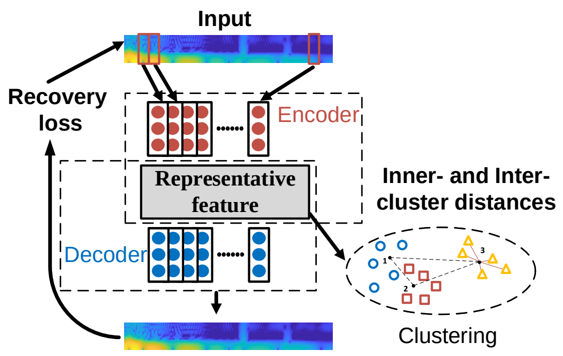

Fig. 8: **Illustration of the auto-encoder based unsupervised keystroke recognition design.**

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 [19], 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.[2] 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:

$$L_{inner} = \frac{1}{N} \sum_{i=1}^{N} ||f^{(t)}(a_i) - c_i^{(t)}||, \qquad (5)$$

$$c_i^{(t)} = \arg\min_{c_j^{(t\ 1)}} ||f^{(t)}(a_i) - c_j^{(t\ 1)}||, \qquad (6)$$

where $f^{(t)}(\cdot)$ is the encoder at the $t^{th}$ iteration in the training, $a_i$ is the feature value of the $i^{th}$ 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.

*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

[2]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.

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:

$$L_{inter} = \sum_{i=1}^{N} ||f^{(t)}(a_i) - s_i^{(t)}||, \quad (7)$$

$$s_i^{(t)} = \arg\min_{c_j^{(t\ 1)} \neq c_i^{(t\ 1)}} ||f^{(t)}(a_i) - c_j^{(t\ 1)}||, \quad (8)$$

where $s_i^{(t)}$ is the second closest cluster center to the $i^{th}$ 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:

$$L_{loss} = L_{rec} + \alpha \times L_{inner} - \beta \times L_{inter}, \quad (9)$$

where $\alpha$ and $\beta$ are two parameters. With $L_{loss}$, we can train the auto-encoder and clustering model together in an iterative manner [20]. The cluster centers are randomly initialized at the beginning. 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 turn 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.

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.

## IV. System Evaluation

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

### A. Experiment Setup

**Implementation.** We implement *TapLeak* on Samsung Galaxy S7, Nexus 5X and Huawei P30 Pro as the victim devices. We invite one volunteer as the adversary. For each device, we collect 200 keystrokes for each key from the adversary to form the training data set. 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 I 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.

| 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 |

TABLE I: **Inference time of different components.**

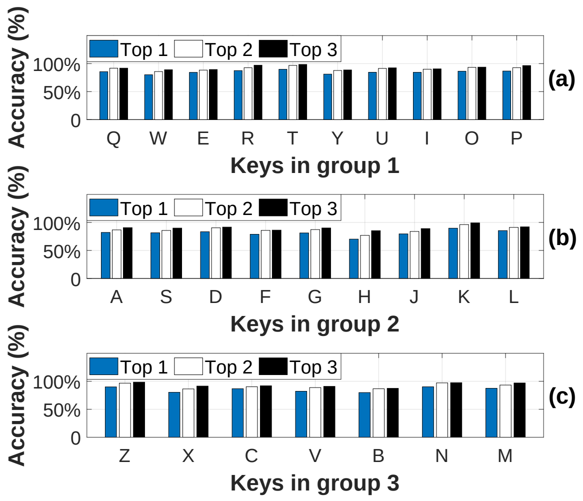

Fig. 9: **Overall performance of *TapLeak* for each key.**

**Methodology.** To evaluate the system performance, we invite 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).

**Metrics.** We use the following metrics to show the performance of *TapLeak*.

*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.* In a 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 $i^{th}$ row and $j^{th}$ column, the reported value means that the percentage of the number of $i^{th}$ keys (*i.e.*, the typed keys) are identified as the $j^{th}$ key by *TapLeak*, out of the total typed number for the $i^{th}$ key.

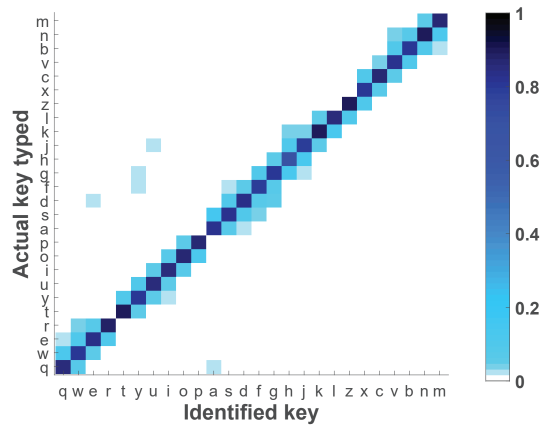

Fig. 10: **Confusion matrix of each key in *TapLeak*.**

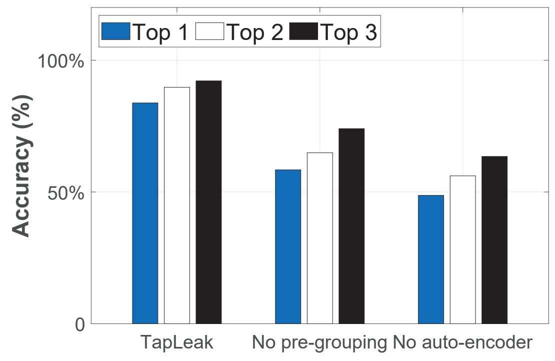

Fig. 11: **Impacts of different system components.**

## B. Overall Performance

As stated in Section II-B, in *TapLeak*, we view each row on the keyboard as one group and there are thus three groups in total. Fig. 9 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 likely dominates the feature of the generated sound. As a result, *TapLeak* can achieve a good performance even if it recognizes keystrokes in an unsupervised manner.

To understand how the recognition errors distribute, Fig. 10 further plots the confusion matrix. We observe that most of the 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.

## C. Impacts of System Components

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

**Performance gains.** *TapLeak* contains two major components: pre-grouping and auto-encoder based recognition. In Fig. 11, 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. 11 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 III-B, we analyze that using two identified starting points from two microphones could cause larger errors compared with the cross-correlation. Fig. 12(a) compares their performance of the pre-grouping accuracy. We can see that the cross-correlation improve the accuracy by 24.4% to 36.2%.

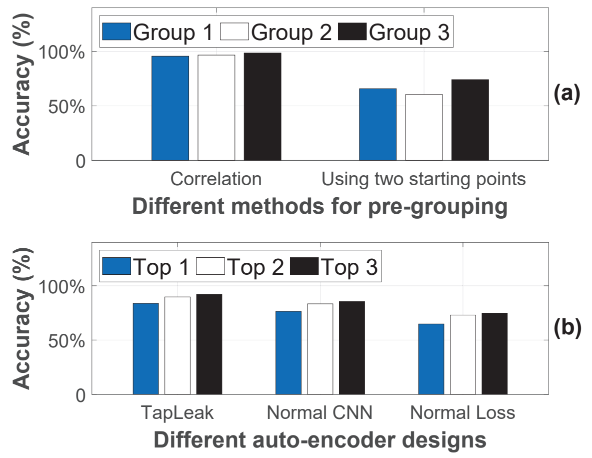

Fig. 12: **Keystroke recognition accuracy (a) with different methods to calculate TDoA for pre-grouping, and (b) with different auto-encoder designs.**

**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 spectrum image ("Normal CNN"). Fig. 12(b) 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. 12(b) shows

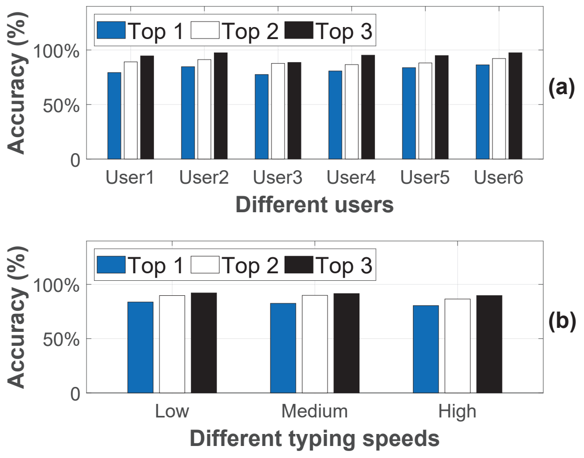

Fig. 13: **Keystroke recognition accuracy (a) for different users and (b) under different typing speeds.**

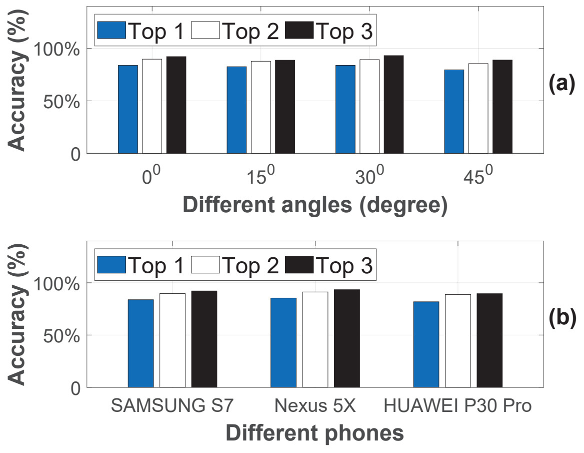

Fig. 14: **Keystroke recognition accuracy (a) with different angles between the device screen and the horizontal plane, and (b) with different type of mobile phones.**

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.

## *D. Other Micro-Benchmarks*

**Different users.** Fig. 13(a) 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, *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. 13(b) 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. 14(a) 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 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 192 KHz for all the three mobile phones, and the number of groups in the pre-grouping 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. 14(b), 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 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. 15(a) 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, which in turn degrade the keystroke recognition performance.

## *E. Possible Defense Mechanism*

Finally, we discuss the possible mechanism to defend this attack. We notice that the operating system of a mobile device can generate a sound through the speaker and/or trigger the motor to vibrate for each keystroke. Both could produce extra sounds. Such sounds will be mixed with the sound generated by the user finger's tapping, which may potentially disturb and even impair the keystroke recognition.

We first examine the software generated sounds only. Fig. 15(b) shows the performance of *TapLeak* under different levels of the software generated sounds. From the result, we find that such sounds do not impact the system performance.

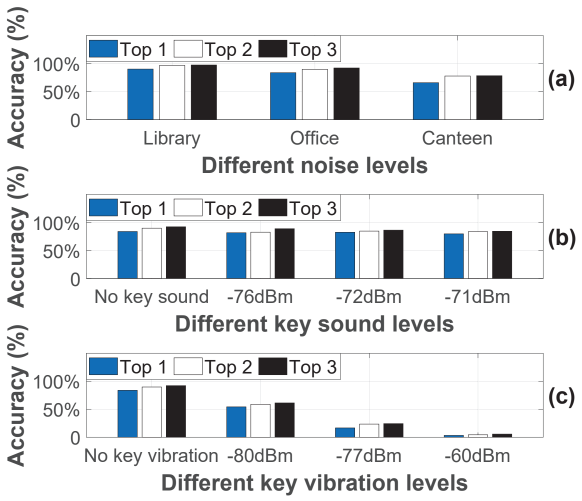

Fig. 15: **Keystroke recognition accuracy (a) with different background noise levels, (b) with different software generated key sound levels and (c) with different software generated device vibration levels.**

This is because although for each keystroke, we now record two types of the sounds. The software generated sound always suffers a short delay, *e.g.*, about 100 ms, probably due to the responding time of the operation system. This delay is larger than the duration (*e.g.*, 40 ms) of the sound generated by the vibration caused by the finger's tapping. It is unlikely for the software generated sound to overlap with the consequent keystroke sounds (if so, the user needs to type at least 10 keys per second). As the software generated sound usually has a much larger amplitude compared with the vibration generated sounds, we can easily separate them to exclude its impact. Thus, the software generated sound has a limited impact on the proposed attack, which is not a suitable defence measure.

However, we find that the vibration of the device's motor triggered by the operating system could impact the performance of *TapLeak* significantly. Fig. 15(c) 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. By analyzing the recorded sounds, we observe that the delay of the additional sound caused by the motor's vibration is only 30 ms, which will be overlapped with the sound from the finger's tapping. Moreover, the frequency response of the motor's vibration also mixes with finger's tapping sound in the frequency domain. Hence, the performance of *TapLeak* degrades, while this inspires us that enabling the motor vibration could be a possible mechanism to defend this attack.

## V. 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 [21] and TapLogger [7] utilize on-board motion sensors to infer keystrokes on numeric keyboards. TapPrints [22] expands the inference area to any location on the screen. To further improve the performance, PIN Skimmer [23] 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.* [13] utilize a set of on-board sensors, including microphones, to realize a keystroke recognition design. Shumailov *et al.* [24] utilize the TDoA measurements of the acoustic signals between the two microphones to recognize the keystrokes. Under a similar setting, TapSnoop [25] 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 [26], [27], [28], [29], [30] or transmit inaudible sounds first and then record the reflected sounds by user's finger in the typing [31]. On the other hand, some researchers also study the keystroke inference on mobile phones [32], [33] or laptops [34] through some external devices, *e.g.*, smart watch on the user's wrist, nearby camera [35], 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 [36] can achieve a high-quality finger tracking on a 2-D plane. VSkin [11] 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.* [5] further utilize RNNs to achieve a room-scale hand motion tracking. In addition to the tracking of the user's motion, some other works also study the tracking or ranging for another mobile device [37], [38], [39], [40]. 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 [9]. Existing works utilize auto-encoder for dimensionality reduction [41], feature extraction [42], recommendation system design [43], and image compression [17]. Moreover, some works [20], [44] 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 to address this issue.

## VI. 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 and unsupervised keystroke recognition two major design issues, and implement our designs in a prototype system. Extensive experiments indicate the efficacy of *TapLeak* by attacking more than 4000 keystrokes from different users on various mobile phones.

## Acknowledgement

The work described in this paper was sponsored by the project JCYJ20190808183203749 supported by the Science Technology and Innovation Committee of Shenzhen Municipality, and the GRF grant from Research Grants Council of Hong Kong (Project No. CityU 11217817). Zhenjiang Li is the corresponding author.

## References

[1] B. Jiang, S. Wu, F. Lin, and J. Xu, "Detecting and switching between noise reduction modes in multi-microphone mobile devices," Aug. 15 2017, uS Patent 9,736,287.

[2] W. Jin, M. J. Taghizadeh, K. Chen, and W. Xiao, "Multi-channel noise reduction for hands-free voice communication on mobile phones," in *Proc. of IEEE ICASSP*, 2017.

[3] P. Xie, J. Feng, Z. Cao, and J. Wang, "Genewave: Fast authentication and key agreement on commodity mobile devices," *IEEE/ACM Transactions on Networking*, 2018.

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

[5] W. Mao, M. Wang, W. Sun, L. Qiu, S. Pradhan, and Y.-C. Chen, "Rnn-based room scale hand motion tracking," in *Proc. of ACM MobiCom*, 2019.

[6] J. Liu, C. Wang, Y. Chen, and N. Saxena, "Vibwrite: Towards finger-input authentication on ubiquitous surfaces via physical vibration," in *Proc. of ACM CCS*, 2017.

[7] Z. Xu, K. Bai, and S. Zhu, "Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors," in *Proc. of ACM WiSec*, 2012.

[8] F. Gustafsson and F. Gunnarsson, "Positioning using time-difference of arrival measurements," in *Proc. of IEEE ICASSP*, 2003.

[9] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013.

[10] G. Zhang, C. Yan, X. Ji, T. Zhang, T. Zhang, and W. Xu, "Dolphinattack: Inaudible voice commands," in *Proc. of ACM CCS*, 2017.

[11] K. Sun, T. Zhao, W. Wang, and L. Xie, "Vskin: Sensing touch gestures on surfaces of mobile devices using acoustic signals," in *Proc. of ACM MobiCom*, 2018.

[12] Y. Zhou, Z. Wang, W. Zhou, X. Jiang, and P. Ning, "Detecting malicious apps in official and alternative android markets," in *Proc. of ACM CODASPY*, 2012.

[13] S. Narain, A. Sanatinia, and G. Noubir, "Single-stroke language-agnostic keylogging using stereo-microphones and domain specific machine learning," in *Proc. of ACM WiSec*, 2014.

[14] P. Scalart *et al.*, "Speech enhancement based on a priori signal to noise estimation," in *Proc. of IEEE ICASSP*. IEEE, 1996.

[15] T. Kinnunen and H. Li, "An overview of text-independent speaker recognition: From features to supervectors," *Speech communication*, 2010.

[16] Z. Tüske, P. Golik, R. Schlüter, and H. Ney, "Acoustic modeling with deep neural networks using raw time signal for lvcsr," in *Proc. of INTERSPEECH*, 2014.

[17] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto, "Deep convolutional autoencoder-based lossy image compression," in *Proc. of IEEE PCS*, 2018.

[18] J. Walker, C. Doersch, A. Gupta, and M. Hebert, "An uncertain future: Forecasting from static images using variational autoencoders," in *Proc. of Springer ECCV*, 2016.

[19] K. Wagstaff, C. Cardie, S. Rogers, S. Schrödl *et al.*, "Constrained k-means clustering with background knowledge," in *Proc. of ICML*, 2001.

[20] C. Song, F. Liu, Y. Huang, L. Wang, and T. Tan, "Auto-encoder based data clustering," in *Proc. of Springer CIARP*, 2013.

[21] L. Cai and H. Chen, "Touchlogger: Inferring keystrokes on touch screen from smartphone motion." *USENIX Summit on Hot Topics in Security*, 2011.

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

[23] L. Simon and R. Anderson, "Pin skimmer: Inferring pins through the camera and microphone," in *Proc. of ACM SPSM*, 2013.

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

[25] H. Kim, B. Joe, and Y. Liu, "Tapsnoop: Leveraging tap sounds to infer tapstrokes on touchscreen devices," *IEEE Access*, 2020.

[26] D. Asonov and R. Agrawal, "Keyboard acoustic emanations," in *Proc. of IEEE S&P*, 2004.

[27] 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. of ACM MobiCom*, 2015.

[28] L. Zhuang, F. Zhou, and J. D. Tygar, "Keyboard acoustic emanations revisited," *ACM Transactions on Information and System Security*, 2009.

[29] T. Zhu, Q. Ma, S. Zhang, and Y. Liu, "Context-free attacks using keyboard acoustic emanations," in *Proc. of ACM CCS*, 2014.

[30] Y. Berger, A. Wool, and A. Yeredor, "Dictionary attacks using keyboard acoustic emanations," in *Proc. of ACM CCS*, 2006.

[31] L. Lu, J. Yu, Y. Chen, Y. Zhu, X. Xu, G. Xue, and M. Li, "Keylister-ber: Inferring keystrokes on qwerty keyboard of touch screen through acoustic signals," in *Proc. of IEEE INFOCOM*, 2019.

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

[33] Y. Liu and Z. Li, "aleak: Privacy leakage through context-free wearable side-channel," in *Proc. of IEEE INFOCOM*, 2018.

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

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

[36] W. Wang, A. X. Liu, and K. Sun, "Device-free gesture tracking using acoustic signals," in *Proc. of ACM MobiCom*, 2016.

[37] C. Peng, G. Shen, Y. Zhang, Y. Li, and K. Tan, "Beepbeep: a high accuracy acoustic ranging system using cots mobile devices," in *Proc. of ACM SenSys*, 2007.

[38] S. P. Tarzia, P. A. Dinda, R. P. Dick, and G. Memik, "Indoor localization without infrastructure using the acoustic background spectrum," in *Proc. of ACM MobiSys*, 2011.

[39] J. Yang, S. Sidhom, G. Chandrasekaran, T. Vu, H. Liu, N. Cecan, Y. Chen, M. Gruteser, and R. P. Martin, "Detecting driver phone use leveraging car speakers," in *Proc. of ACM MobiCom*, 2011.

[40] W. Mao, J. He, and L. Qiu, "Cat: high-precision acoustic motion tracking," in *Proc. of ACM MobiCom*, 2016.

[41] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, 2006.

[42] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proc. of ICML*, 2008.

[43] X. Li and J. She, "Collaborative variational autoencoder for recommender systems," in *Proc. of ACM SIGKDD*, 2017.

[44] X. Guo, X. Liu, E. Zhu, and J. Yin, "Deep clustering with convolutional autoencoders," in *Proc. of Springer ICONIP*, 2017.