Keyboard Snooping from Mobile Phone Arrays with Mixed Convolutional and Recurrent Neural Networks

TYLER GIALLANZA, Darwin Deason Institute for Cybersecurity, USA
TRAVIS SIEMS, Darwin Deason Institute for Cybersecurity, USA
ELENA SMITH, Darwin Deason Institute for Cybersecurity, USA
ERIK GABRIELSEN, Darwin Deason Institute for Cybersecurity, USA
IAN JOHNSON, Darwin Deason Institute for Cybersecurity, USA
MITCHELL A. THORNTON, Darwin Deason Institute for Cybersecurity, USA
ERIC C. LARSON, Darwin Deason Institute for Cybersecurity, USA

The ubiquity of modern smartphones, because they are equipped with a wide range of sensors, poses a potential security risk—malicious actors could utilize these sensors to detect private information such as the keystrokes a user enters on a nearby keyboard. Existing studies have examined the ability of phones to predict typing on a nearby keyboard but are limited by the realism of collected typing data, the expressiveness of employed prediction models, and are typically conducted in a relatively noise-free environment. We investigate the capability of mobile phone sensor arrays (using audio and motion sensor data) for classifying keystrokes that occur on a keyboard in proximity to phones around a table, as would be common in a meeting. We develop a system of mixed convolutional and recurrent neural networks and deploy the system in a human subjects experiment with 20 users typing naturally while talking. Using leave-one-user-out cross validation, we find that mobile phone arrays have the ability to detect 41.8% of keystrokes and 27% of typed words correctly in such a noisy environment—even without user specific training. To investigate the potential threat of this attack, we further developed the machine learning models into a realtime system capable of discerning keystrokes from an array of mobile phones and evaluated the system's ability with a single user typing in varying conditions. We conclude that, in order to launch a successful attack, the attacker would need advanced knowledge of the table from which a user types, and the style of keyboard on which a user types. These constraints greatly limit the feasibility of such an attack to highly capable attackers and we therefore conclude threat level of this attack to be low, but non-zero.

CCS Concepts: • **Human-centered computing** → **Ubiquitous and mobile computing**; • **Computing methodologies** → **Neural networks**; • **Computer systems organization** → *Sensors and actuators*;

Additional Key Words and Phrases: Keyboard Snooping, Machine Learning, Security

ACM Reference Format:
Tyler Giallanza, Travis Siems, Elena Smith, Erik Gabrielsen, Ian Johnson, Mitchell A. Thornton, and Eric C. Larson. 2019. Keyboard Snooping from Mobile Phone Arrays with Mixed Convolutional and Recurrent Neural Networks. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 3, 2, Article 45 (June 2019), 22 pages. https://doi.org/10.1145/3328916

Authors' addresses: Tyler Giallanza, Darwin Deason Institute for Cybersecurity, USA, tgiallanza@smu.edu; Travis Siems, Darwin Deason Institute for Cybersecurity, USA; Elena Smith, Darwin Deason Institute for Cybersecurity, USA; Erik Gabrielsen, Darwin Deason Institute for Cybersecurity, USA; Ian Johnson, Darwin Deason Institute for Cybersecurity, USA; Mitchell A. Thornton, Darwin Deason Institute for Cybersecurity, USA; Eric C. Larson, Darwin Deason Institute for Cybersecurity, USA.

2474-9567/2019/6-ART45 $15.00
https://doi.org/10.1145/3328916

1 INTRODUCTION

The ubiquity of mobile phones and the Internet of Things raises considerable security concerns because the embedded sensors and computational capabilities of these devices could be re-purposed for surveillance. Since smartphones are already ubiquitous and contain a wide variety of sensors, they are especially valuable targets for sensor-based data snooping. This is exemplified by a number of works in the ubiquitous computing community: Yu *et al.* demonstrated how mobile phones could be used to infer handwritten letters [38]. Ali *et al.* showed that WiFi routers could be used to identify typing on nearby keyboards [3]. Lu *et al.* used smartwatch motion sensors to snoop typed passwords of a user's mobile device [18].

In this research, we investigate the feasibility of using smartphones for sensor-based data snooping of keyboard typing. Specifically, we aim to determine if the sensors in an array of mobile phones are able to predict what a user is typing on a keyboard and under what conditions an attacker could successfully deploy such an attack. To simulate a realistic scenario, we designed an IRB-approved human subject experiment to generate sensor data from mobile phones while participants typed on a nearby keyboard. We use a number of different phone models, keyboards, and tabletop positions; we encourage background noise (over 40% of keystrokes occur while participants are speaking); and we do not restrict our participants to a limited typing vocabulary or typing speed, thus testing our system in a realistic situation. We conclude that smartphone sensors, particularly when arrayed, have the ability to detect typing with surprising accuracy, achieving 41.8% accuracy on our dataset without any user-specific calibration. To achieve this accuracy, we employ cascaded convolutional and recurrent neural networks trained upon the audio data of up to eight mobile phones. To investigate the generalizability of the model we further develop a prototype system and deploy it in online experiments. We assume the threat model is such that an attacker can install an application on an array of users' phones or in some way access motion and audio data from the phones on a tabletop. An example attack scenario is an attacker who wishes to gain information about the contents of notes typed during a conference-style meeting. Further investigations are conducted to discern the realism of other factors such as: Does the attacker need physical access to the room and table in which an attack will take place? Is known phone location critical for carrying out the attack? We show that the deployed system performs well when in conditions similar to training data collection, but significantly drops in performance if the table and style of keyboard are not part of the training data. We therefore conclude that, while the attack is possible, it is only feasible for sophisticated attackers.

The contributions of our work are as follows:

(1) We present an architecture for segmenting keystrokes and extracting physical keyboard dynamics from mobile phone sensors arrays with convolutional neural networks.
(2) We design a recurrent neural network architecture for translating sequences from the convolutional network into typed phrases.
(3) We evaluate our designs by creating a human subjects experiment that systematically changes keyboard type, location, and varies the ambient noises in the room.
(4) We develop a real-time prototype and evaluate the deployed prototype in additional experiments varying the room, table, keyboard style, and phone locations to values unseen in the training data.

We believe these results are state-of-the-art and best known in the publicly available scientific literature. Compared to the results that are reported from other researchers, our data was collected in a realistically noisy environment and a more practical setting. Also, the past results from other researchers utilized high-quality sensors (microphones, *etc.*) whereas we use commodity off-the-shelf sensors on standard smartphones.

2 RELATED WORK

Many researchers have investigated the ability for sensors to detect information they were not designed to sense. Specifically, related research has detected keystrokes, handwriting, and passwords from a variety of devices including microphones, smartwatches, and mobile phones. We design our experiment to overcome the limitations in the existing research; namely, we focus on collecting realistic data by preserving noise in data collection and collecting natural typing data rather than restricting typists to a known vocabulary.

2.1 Eavesdropping Attacks

As mobile devices become more prevalent and more sensor-rich, more research has utilized commercial mobile devices for eavesdropping. Many of these studies utilize mobile device sensors to extract information entered on the mobile device itself. PIN Skimmer, for example, utilizes camera and microphone data to correctly guess more than 50% of PINs after 5 attempts [32]. TouchLogger [4] and TapLogger [37] utilize motion sensor data to extract PINs entered on mobile phones. These methods have been shown to work on mobile phones of varying sizes and hardware configurations with accuracy 81 times better than random [5]. The reliance on motion data rather than camera or microphone data is that motion data does not require obtaining explicit user permissions, thus increasing the feasibility of the attack scenario. It has been shown that users are often not aware of these threats and are thus more vulnerable to this kind of attack [21]. Similarly, Simon *et al.* detected pre-defined sentences and identified the authors of anonymous messages with a mobile application that did not require special permissions [33]. Furthermore, Yu *et al.* used mobile phone audio and accelerometer sensors to detect nearby handwritten words, achieving at best 60% word-level accuracy on words chosen from a fixed vocabulary and data generated in a noise-controlled environment [38].

Smartwatch devices have also demonstrated vulnerabilities to side channel attacks. Recently, Lu *et al.* demonstrated that sensors embedded in smartwatches can be used to detect passwords entered within 20 attempts [18]. The system uses motion data to detect both tapped and swiped passwords on smartwatches, generalizing across different smartwatch models.

Our work is similar to these studies in spirit because we attempt to use sensors from commercially available mobile phones rather than specialized equipment for our attack. However, we focus on detecting keystrokes typed on a keyboard physically separate from the mobile device. This introduces difficulties caused by distance not present when collecting data from the device that is being attacked.

2.2 Keystroke Detection

Many researchers have investigated the use of audio sensors to perform keystroke recognition. Zhuang *et al.* used a hidden Markov model on cepstrum features extracted from audio data collected by a microphone to achieve 94% character-level accuracy [39]. However, these results occur using a mechanical keyboard in a noise free environment. Additionally, the system is trained and tested on the same person using the same keyboard in a similar noise environment. Kelly *et al.* achieved similar results using audio analysis from studio-quality microphones recording in a sound-proof environment [16]. These studies demonstrate the ability for audio-based systems to detect keystrokes but employ studio-quality microphones and noise-controlled environments.

In the ubiquitous computing community, Ali *et al.* showed that wifi routers can be used to facilitate keystroke detection, achieving about 93.5% accuracy in a human subjects trial [3]. Despite the high accuracy in this study, there are a number of drawbacks, including a low-noise experimental setup, user-specific training, and lack of variation in the keyboard and router used.

In terms of using mobile devices for keystroke detection, Liu *et al.* detected keystrokes via sensors on a smartwatch worn by the typist [17]. This achieved 57% word-level accuracy on the top guess, but the dictionary used was greatly domain-restricted to the same topic the subject typed about. Marquardt *et al.* used mobile phone accelerometers to achieve 40% accuracy detecting keystrokes, but limited the vocabulary of the typists to a small dictionary of known words and limited the typing rate of the users [20].

Much of the existing work in detecting keystrokes leverages specialized hardware such as high-quality microphones. While we agree that these methods can achieve high accuracy and good performance, in this paper we are concerned with investigating mobile devices in particular because an attacker that can install malicious applications on a mobile device can easily disguise their surveillance and would not require physical access to a room prior to surveillance. The existing research in keystroke detection from mobile devices is limited in terms of generalizability and collection of realistic data. Smartwatch-based attacks are limited because smartphones are much more prevalent. Furthermore, existing studies utilizing mobile phones restricted the typists' dictionary, trained and tested on the same typists, and/or restricted the amount of background noise. To the best of our knowledge, no researchers have achieved results by systematically varying keyboard type and location, and allowing ambient noise with a large number of participants to generate natural typing data.

3 SYSTEM DESIGN

In order to detect and classify each keystroke, we require features from the signal data that help delineate each keystroke. These differences are best intuited using an independent source-filter model, which gained popularity in the speech community for describing the process of speech production [8]. In this model, the excitation source and filter that alters the excitation are independent of one another. For keystroke detection, the excitation source is the key movement and strike against the keyboard, which create sound waves and motion. The filter that manipulates this excitation consists of any medium that the signal travels through before arriving at each mobile phone on the table. The transfer function of the path between the excitation and each mobile phone determines how the signal is changed once it is sensed. We can regard the excitation and filtering processes as independent by assuming the force of the keystroke does not alter the position or contact points between the keyboard and the table. Using the source-filter model, we can expect the signals from each keystroke to be slightly different depending on a number of factors. The excitation signal might be affected by (1) the mechanical structure of the key that changes audibility or vibration when pressed, (2) the strike angle at which the user hits the key, and (3) how long the key is depressed. The filter between each phone and source is affected by (1) the angle between the key strike and phone (2) the distance between the phone and each strike, and (3) the material in which the signal travels. Each of these factors assist in identifying the exact keystroke because the path between each key and phone is slightly different and the user strikes each key at a different angle and force. As such, we expect that many confusions will occur between keys that are physically close because they are likely to have similar, but not identical, transfer functions. Moreover, the excitation signal can be affected by the user's typing style and consistency. For our machine learning model to function properly, these differences must produce keystroke signatures with enough specificity to delineate them, even in the presence of other sound and motion signals from the environment.

We use a mixed convolutional and recurrent neural network architecture to detect keystrokes from phone sensor data. Because we expect the keystrokes to produce identifiable sound and vibration signals, we focus on the microphone sensor, the accelerometer sensor, and combining both in a multi-modal network. Our overall processing pipeline is as follows: first, we combine the sensor data generated by the phones; second, we window the signal and extract features for processing; third, we determine which windows contain keystrokes and which do not by using a convolutional neural network; fourth, we analyze windows that contain keystrokes with a convolutional neural network to predict which key is pressed; and fifth, we refine the predictions with a recurrent neural network and language model.

The overall system is composed of three parts: the keystroke detector, the convolutional keystroke classifier, and the recurrent keystroke classifier. The keystroke detector and the convolutional keystroke classifier share convolutional layers, making them two different modes of the same network. The keystroke detector is trained on all windows from the source signal, including windows that do contain keystrokes and windows that do not contain keystrokes, and the convolutional keystroke classifier is only trained on windows that contain keystrokes.

At testing time, the keystroke detector is tested on all windows from the source signal, returning the windows that contain keystrokes. The convolutional classifier is then tested on the windows that the keystroke detector returned, whether or not those windows do in fact contain a keystroke.

3.1 Data Preprocessing

Before any further processing can take place, the signals from multiple phones must be aligned temporally. Each timestamp value yields sub-millisecond precision [1], but we cannot assume that all devices share the same base time synchronization. Although all devices use NTP (Network Time Protocol) to synchronize their *base system time* to *server-distributed time* [23], each device's base time varies slightly due to clock drift [34]. Since each device is recording the same signal, we attempt to synchronize the base time of all devices by comparing the signals to each other.

Specifically, we perform a cross-correlation on the absolute value of the normalized audio signal for each phone over 15-second time windows. The 15-second window ensures that the phones will be synchronized provided that their base system times are within 15 seconds of each other before synchronization. Due to the varying quality of the phones' microphones, the audio signals do not always perfectly correlate. To remedy this issue, we perform the cross-correlation over 20 different randomly sampled time windows, sum the correlation coefficients, and use the index of the maximum coefficient as the amount of offset for that phone. To synchronize all of the phones, we randomly choose one phone as the baseline signal and synchronize every other phone with the baseline phone.

3.2 Windowing

Next, we break the time aligned signals into windows. During pilot experiments, we observed that each keystrokes had audio and motion signatures up to 200*ms*, but typically were less than 100*ms*. However, some typists begin a new keystroke only 40*ms* after the previous stroke. Therefore, it is impossible to use a window to capture the entire signal and simultaneously only one keystroke. As a tradeoff, we use windows that are 100*ms* long with a step size of 25*ms*. We label each window with which key was pressed during that window, or a special value indicating no key was pressed for the duration of the window. If multiple keys were pressed in the same window, the window is labeled with the value of the first key. In particular, we label windows based on a "key down" event. This occurs when the key is first pressed by the user (as opposed to when the key is released by the user); we use key down events because they generate the strongest signal (based upon our observations).

This windowing strategy allows us to detect any keystrokes that occur at least 25*ms* apart. Keys that occur within 25*ms* of each other cannot be detected.

3.3 Feature Extraction

We use three variations of audio data as features to our model: unprocessed audio data, fast Fourier transform based features (FFT-based), and Mel-frequency scaled Cepstral Coefficients (MFCCs). We choose these features because they are valuable for separating the source and filter responses in the signal data. MFCCs, in particular, change convolution to addition in the Cepstral domain, which may help separate keystrokes with similar excitation signals but different filtering paths and vice-versa. Before extracting FFT-Based features and MFCCs, we apply a pre-emphasis filter to the signal. The pre-emphasis filter has been shown to boost classification results for MFCCs in the automatic speech recognition community [24]. We use a infinite impulse response pre-emphasis filter coefficient of 0.97. Moreover, we also apply a hamming window to our time series windows before further frequency analysis to mitigate spectral leakage into other bands.

FFTs divide signals into frequency components in a computationally efficient way [35]. In our analysis, we use an FFT size of 512 and sampling rate of 44.1 kHz. Thus, our frequency resolution is about 86 Hz per bin. By applying the FFT in a windowed manner, we can calculate the short time Fourier transform (STFT) of the signal over time. This allows us to work with the frequency domain representation of the original signal directly, forcing the convolutional neural network to interpret the signal based on its frequency content. In our implementation we use a sliding window of size $25ms$ with an overlap of $10ms$. Thus, we calculate the FFT of the signal every $15ms$, about 67 times per second, and the number of features per window is 256 (half of the FFT magnitude spectrum). To reduce complexity of the STFT, we also average the features across frequencies mapping their values to the Mel frequency scale. Thus the STFT features are aggregated from 256 features per time window to 26 features per time window.

MFCCs are typical features in speech processing systems because they partially mimic how the human ear differentiates sounds [26]. MFCCs additionally condense the information found in an FFT, mapping a given window of an FFT onto the Mel scale using a number of filters. Specifically, calculating MFCCs is as follows: Take the FFT of a window of a signal; map the powers of the spectrum onto the Mel scale; take the log of the powers at the Mel frequencies; take the discrete cosine transform (DCT) of the resulting logs; and use the amplitudes of the resulting spectrum as the MFCC. We use a window size of 512 audio samples and a bank of 40 filters as parameters for our MFCCs. Finally, we apply a sinusoidal liftering of length 22 to increase the magnitude of higher ceps coefficients, as is common in speech recognition applications [15].

Because the MFCCs use the DCT, they have an energy compaction property that tends to place larger amplitudes in the initial coefficients, and also tends to decrease correlation among the MFCC coefficients [27]. Historically, this de-correlating process was beneficial to machine learning algorithms. However, with the advent of deep learning-based feature extraction, it is unclear how necessary the use of MFCCs (or the FFT) are for pre-processing [25]. However, the Fourier transform operation is difficult to learn and the increase in amount of required training data and model complexity may not be warranted. As such, we employ the use of all three input features in our deep learning model: raw audio, FFT-based, and MFCCs. Moreover, it is unclear if the biological motivations of the MFCCs are helpful outside of automated speech recognition. That is, humans do not have the ability to discern keystrokes audibly so there is no reason that biologically motivated operations (like Mel-scaling) would assist in classification.

In addition to audio data, we also use various motion data as features. Specifically, we use accelerometer data (x, y, and z sampled at 100Hz) as well as gyroscope data (x, y, and z sampled at 100Hz). Rather than use accelerometer and gyroscope data as separate features, we combine them into one feature with six channels (accelerometer $x/y/z$, and gyroscope $x/y/z$) to decrease model size and training time. That is, temporal filters are applied to the motion data simultaneously much like filtering is applied in natural language processing of sentence embeddings [14].

3.4 Keystroke Detection

Before classifying which keystrokes occurred, we first determine if keystrokes occurred (*i.e.*, detection versus classification). Using the extracted audio features, the raw audio, and the motion data, we apply a convolutional neural network to the sensor data. We use a multi-modal architecture such that any combination of the four features (FFTs, MFCCs, raw audio, and raw motion) can be used as inputs to the model. We choose a multi-modal architecture because it is unclear which extracted features are most important for detection and classification, therefore we use all features in our training process. Each input passes through a number of convolutional layers, and the output of the convolutions are concatenated and passed through a number of shared dense layers. The final dense layer predicts if the given window contains a keystroke or does not contain a keystroke.

As discussed, the convolutional layers are all one dimensional convolutions. Multi-dimensional inputs like MFCCs and motion are treated as one-dimensional along the time axis, with multiple channels. This is a common method for analyzing frequency transformed signal data [2]. The MFCC input is 40 channeled (because we use 40 filters when mapping to the mel scale), the motion input is 6 channeled (x, y, and z channels for both accelerometer and gyroscope), and the raw audio and FFT inputs are both single channeled.

Each input passes through a different architecture of convolutions before being passed to the dense layers. Each convolutional architecture contains a mix of one dimensional convolutional layers, max pooling layers, and ADD layers. The max pooling layers perform dimensionality reduction by choosing the best performing features in a given window [30]; we down-sample in the pooling layers by a factor of 8 for raw audio, by a factor of 4 for motion, and by a factor of 2 for all other input features. Other down-sampling factors were investigated, but provided similar results. The ADD layers are part of the residual block architecture as introduced in [13]. Residual blocks allow training of deeper networks that use more convolutional layers while reducing the effects of the vanishing gradient problem, training shallow and deep layers more evenly because the "addition" operation allows for multiple back-propagation update paths [13]. Figure 1 shows the architecture of the network.

After each input passes through convolutional layers, the result is flattened into a one dimensional array. These arrays are concatenated for each input to the network, and the concatenated value is passed to a number of dense layers. We use three dense layers: one with 128 nodes, one with 64 nodes, and an output layer with 2 nodes. All layers are initialized using Glorot's method, which helps prevent saturated neurons in early epochs [11]. The first two layers use the ReLU activation function to help avoid the vanishing gradient problem [11], and the last layer uses the softmax function, which outputs a score for each class such that the scores for all classes sum to equal 1, with two classes (key/not a key).

3.5 Keystroke Classification: Convolutional Neural Network

We use the same architecture described above for keystroke detection. The convolutional network is a reasonable choice for discerning keystrokes because of the source-filter model that explains differences in sensed keystrokes, presented earlier. It is possible (but not guaranteed) that the convolutions can learn to separate the excitation and filtering of the sensed signal. In general, most CNNs can adapt their filter kernels to identify fine structures in the sensed data that it identifies as discerning.

The input features pass through the same convolutional layers that have already been trained for keystroke detection. In this way, the convolutional layers are treated as "feature extraction" layers that reduce the dimensionality of the data. After the convolutional layers, however, we employ a different set of dense layers to perform keystroke classification. We use three dense layers: one with 256 nodes, one with 128 nodes, and one with n nodes for n output classes.

This training method using shared convolutional layers with different dense layers for each classification task is often known as multi-task learning and often helps to prevent over fitting [6, 7, 10, 29]. All layers are initialized using Glorot's method, the first two layers use the ReLU activation function, and the output layer uses the softmax function as described above.

Because the CNN does not receive any information about the keystrokes before or following the keystroke to which it is currently classifying, the CNN model must classify the keystroke based solely upon the observed signal data. In this way, the CNN can be thought of as a *physical keystroke dynamics model*. We would expect such a model to have similar classification probabilities for keystrokes that occur frequently and keys that are located near one another on the keyboard because they generate similar sensor signatures (as predicted by the source-filter model of the process). While the keystroke dynamics are important for classification, other factors should also be taken into account such as the time between consecutive keystrokes, the likelihood of the previous keystroke classified, and others. These knowledge sources are better captured by a sequential classification procedure like a recurrent neural network, discussed next.

3.6 Keystroke Classification: Recurrent Neural Network

We apply an additional layer of processing on top of the CNN to improve results and take into account the sequential nature of the problem, building out a recurrent neural network (RNN) architecture.Because RNNs have the ability to learn order-dependent features, they are naturally suited for text processing [22].

Our problem is different from typical natural language processing because our input features are not characters but rather character probabilities. Typically, RNNs use one-hot-encoded vectors as inputs [12]; this can be thought of instead as a 100% probability of a certain character. Rather than use one-hot-encoded vectors, we pass the softmax output of the CNN directly into the RNN. This allows us to take advantage of every guess generated by the CNN—often the second guess or the third guess generated by the CNN are correct.

Our goal in using the RNN is that it can correct errors made by the CNN. By analyzing the confusions made by the CNN, we discovered that the CNN makes a number of systematic errors. First, the CNN often confuses keystrokes for frequent classes, for example by mistaking infrequent letters such as *z* for frequent letters such as *e*. Second, the CNN often confuses characters that are physically close on the keyboard, for example by mistaking *r* for *t*. Finally, the CNN often confuses keys that are physically similar in size on the keyboard, such as *backspace* and *enter*. These observations support the notion that the CNN learns a physical dynamics model of the keystrokes.

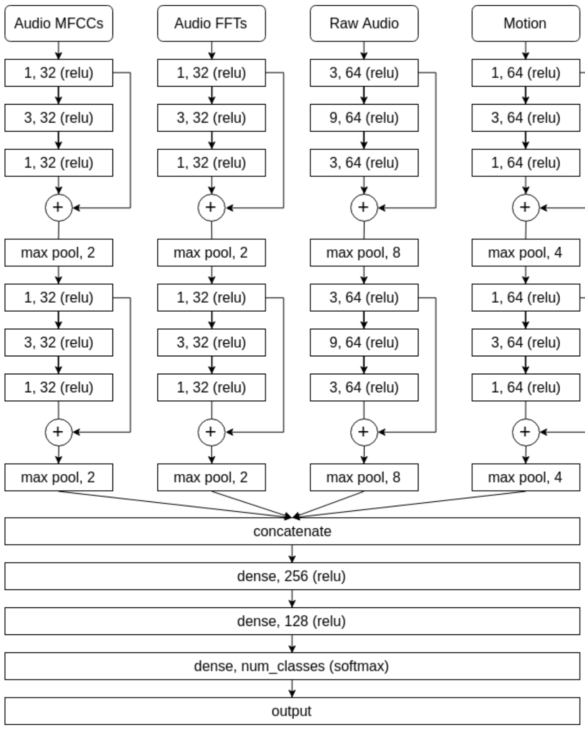


Fig. 1. Convolutional network architectures used for keystroke classification.

In addition to these confusions, an RNN that is aware of typical spelling and grammar conventions should be able to correct "spelling mistakes" that occur when the CNN makes incorrect guesses. For example, an output from the CNN that reads "for exsmple" should be corrected to "for example". With the goal of correcting systematic errors and spelling errors caused by the CNN, we use a translation architecture for the RNN. In this architecture, the RNN takes a sequence of inputs (*i.e.*, keystrokes), and then outputs another, possibly different length, sequence (*i.e.*, letters).

We choose to model the CNN decoding as a translation problem because error correction and translation share many similar properties. Systematic errors, such as confusing keys that are physically close to each other, are fundamentally translation tasks in that there is a direct mapping of input to output. Spelling errors are similarly correctable by a translation architecture because translation requires learning the correct spelling of words and frequent word orderings.

We base our RNN architecture on Google's Neural Machine Translation (NMT) [36]. The model is essentially a sequence-to-sequence model with an attention module. It is composed of three parts: the encoder, the decoder, and the attention module. At training time, the encoder is provided the source language text (CNN output) as input and the decoder is provided the destination language text (true keystrokes). At prediction time, the CNN output is input to the encoder and a special start sequence is input to the decoder. The output from the first step of the decoder is iteratively given as input to the next step of the decoder until a special stop signal is generated.

After the output from the RNN is generated, the RNN output and timing information are given as input into a number of dense layers. For each keystroke, the timing information is simply how long (number of windows) it has been since a key was pressed, and how long it will be (number of windows) until the next key is pressed. This timing information provides additional insight into keystroke classification, because different keys have different time signatures. Figure 2 shows the architecture of the RNN.

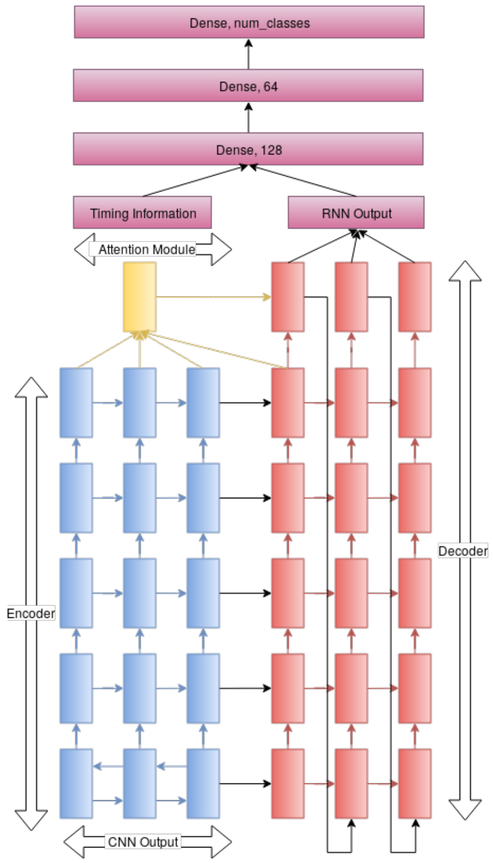


Fig. 2. Layout of Recurrent Neural Network

The Google NMT architecture offers a number of advantages suited to our problem. First, the architecture can operate directly on the character level rather than on the word level. This is a requirement for our use-case because we cannot use a model that operates on word embeddings as the CNN does not necessarily output correctly spelled words. Second, the attention module enables the architecture to learn long-term language dependencies. This allows us to correct the spelling of one word using the corrected spelling of another word, which is important for frequent n-grams such as "next to" or "on top of". Finally, this architecture has been shown to function well even when the computations are rounded mathematically [36], making it suitable for use on an embedded or mobile device. Thus, it is plausible that an attacker could deploy the model in real-time applications.

Each block of NMT architecture is a long-short term memory (LSTM) cell [9]. We use 5 LSTM layers in the encoder, 6 LSTM layers in the decoder, and the attention module described in [19]. The bottom layer in the encoder is a bi-directional LSTM; the bi-directionality allows the network to exploit any right-to-left hierarchies present in the source language [31]. Additionally, the LSTM layers are residually connected. With residual connections, the input to the bottom layer is summed (element-wise) with the output from the bottom layer, and the sum is used as input in the top layer. The residual connections allow for deep layering of LSTMs without the drawback of the vanishing gradient problem [36], training the bottom layers and the top layers evenly.

Although the amount of text generated by the user varies, the RNN can only analyze a constant number of letters at one given time (called the sequence length). For an input size of n letters and a sequence length of s letters, we break the input text into n/s chunks and input these chunks sequentially to the RNN. Longer sequence lengths allow the RNN to learn longer dependencies, but longer sequence lengths result in fewer training examples because there are fewer chunks to process. We determined the sequence length by gridsearching values from 6 to 18.

4 DATA COLLECTION

To collect training and testing data, we designed an IRB-approved study with the goal of recording realistic typing behavior. The data collection setup included one computer (2017 MacBook Pro), three different keyboards (built-in laptop keyboard, Dell mechanical keyboard, and Bluetooth-connected Apple keyboard), and eight iPhones (one iPhone 7, one iPhone 7 Plus, and six iPhone 5S models). The phones were laid out on a wooden conference table in four different seating positions, with a phone on each side of the keyboard. All phones were placed in the same location for all participants, and all phones were charged to 100% battery for the duration of the experiment.

4.1 Experimental Setup

We collected data in two different formats: demographic data and interview style. In demographic data sessions, one participant fills out a survey that records demographic information including gender, age, keyboard experience, *etc*. In interview style sessions, two participants engage in a conversation about a variety of topics ranging from daily activities to political viewpoints. One participant takes on the role of "interviewer," asking questions related to pre-determined topics. The order of topics presented is randomized. As the other participant answers the questions verbally, the interviewer types notes on the answers. Note that the interviewer does not type verbatim what the interviewee speaks—rather, notes on the conversation are taken.

Participants are made aware that they are participating in a study designed to detect typing data from mobile phone sensors and consent to the collection of their anonymized demographic data. Participants are instructed to avoid touching or bumping the mobile phones, but no special instructions about typing style are given. Participants are free to type in shorthand or full sentences, they are allowed to type at any speed, and they are allowed to correct typos or leave them. Furthermore, participants are invited to speak and make non-verbal noises freely, simulating realistic behavior and typical noise levels. Because only one participant is active at a time, demographic data sessions often have little noise, whereas interview sessions are explicitly designed to create noisy typing environments because one participant speaks while the other types.

Each individual randomly cycles among three different keyboards and two different seating locations at the table — throughout the interview the participant physically switches keyboards and physically moves to a different table location. Each person sits in each position with each keyboard for at least one topic. Thus the total number of configurations for each session is: (2 participants) x (2 interviewer seating positions) x (2 interviewee seating positions) x (3 keyboards) = 24 configurations per session. The mobile application, written using Swift for iOS, records audio from the built-in bottom microphone and motion data including accelerometer and gyroscope. Sampling rates for the sensors are 44.1 kHz for audio and 100 Hz for all motion data.

4.2 Demographics and Statistics

The data collection experiment consisted of 20 participants in 10 different sessions. Most participants were recruited from a local mid-sized private university. Table 2a and Table 2b give the demographic information and typing ability for the 20 recruited participants.

Table 2 shows the additional statistics we collected on our dataset. The statistics concerning typing speed are of particular importance because they control how to segment keystrokes; the average typing speed is 340 characters per minute and the median time between keystrokes is 182.9ms with a standard deviation of 34.3ms (Figure 3a). Further, only 3.4% of keystrokes occurred within 25ms of each other. This means that our windowing strategy, which is accurate within 25ms, covers 96.6% of cases observed in the data.

Because our experiments encouraged natural typing styles, we observed a number of different scenarios that increase the difficulty of keystroke prediction. First, there were a large number of *keystroke overlaps*, wherein a second key is pressed down before the previous key is released. We found this overlap to happen in 29% of keystrokes. Additionally, we discovered that more than half of the keystrokes for some typists are separated

Table 1. Demographics and Typing Ability of Study Participants

Gender (n, %)	Male (11, 55%) Female (9, 45%)
Age (μ,range)	23.6 (19-54)
Highest Education (n, %)	High School (5, 25%) Associates (2, 10%) Bachelors (9, 45%) Masters (4, 20%)
Social Status (n, %)	Married (1, 5%) Single (19, 95%)
Ethnicity (n, %)	Hispanic (2, 10%) Non-Hispanic (18, 90%)
Race (n, %)	White (17, 85%) Asian (2, 10%) Black (1, 5%)
Profession (n, %)	Student (11, 55%) Comp. Scientist (6, 30%), Business Person (4, 20%)

(a) Demographics

Mac Users (n, %)	8 (40%)
Computer Use hrs/wk (μ (min,max))	28.4 (10 - 80)
Years using QWERTY keyboard (μ (min,max))	15.7 (0 - 25)
Self-evaluated typing rank (1-10, $\mu \pm 1.96 \cdot \sigma$)	7.1 (± 3.5)
Typing class (n, %)	15 (75%)
Handedness (n, %)	Left (2, 10%), Right (18, 90%)
English as first language (n, %)	18 (90%)
Taken programming class (n, %)	16 (80%)
Hobby that affects hand dexterity? *i.e., plays piano* (n,%)	11 (55%)

(b) Typing Ability

by less than 100*ms*. Related works assumed at least 100*ms* in between every keystroke [16, 20, 39], but, based on our observations of natural typing behavior, we cannot make such a generous assumption (30% of observed keystrokes in our dataset occurred within 100*ms* of each other).

We also discovered a limitation of our ground truth labeling process—the operating system limits the polling-rate of external devices (the USB-connected and Bluetooth-connected keyboards). The polling is limited to once every 15*ms*, meaning that sub-15*ms* precision is impossible for those keyboards.

We designed the interview style session to include audio noise, specifically in the form of talking. To understand how frequently this talking overlapped with the sounds of keystrokes, we use long-term spectral divergence as a voice activity detection algorithm, as proposed by Ramirez *et al.* [28]. We determined that at least one person was talking during 42% of all logged keystrokes. We believe the large percentage of talking-noise mirrors the way an actual eavesdropping attack might occur in a meeting scenario—with people talking while another person typed.

In natural typing behavior, some keys are pressed more frequently than others. In other research, participants were forced to type the same key repeatedly, generating the same number of samples for every key [16, 39]. This is not a realistic mirroring of actual typing. For example, "e", and "t" occur frequently, but should not be given the same consideration as the least frequently typed keys, "ALT", "z", and "q". The most frequent key, "space," was typed 500 times more often than the least frequent key, "q". This reveals that real typing behavior is heavily class imbalanced. Figure 3b shows a histogram of how frequently each key was pressed.

5 RESULTS

We train and test a variety of models, both to determine the optimal hyper-parameters and to compare model performance across different keyboards, positions, and inputs. We never train and test on data from the same typist. This ensures results generalize between different typists and there is no need for calibration of the system using person-specific training data. We use a total of 34 classes: each of the 26 letters, the space bar, the left shift, the right shift, the enter key, punctuation, numbers, edit keys, and alt keys. We choose to combine certain keystrokes into groups because they are infrequent in our dataset. That is, we combine all punctuation into one class, all numbers into one class, and both alt keys into one class.

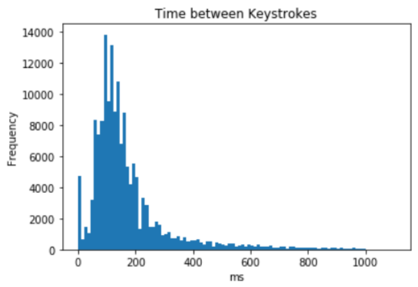
As a baseline, we compare the output of our models to a majority classifier. We report the keystroke-level accuracy for the top guess, the accuracy for the top 5 guesses, and the precision. That is, we report the accuracy calculated across all classes, $acc = (TP + TN)/(TP + TN + FP + FN)$, and the precision, given by: $Precision = \frac{\sum_{c \in Classes} TP_c}{\sum_{c \in Classes} TP_c + FP_c}$ Where TP_c and FP_c are the true positives and false positives for class c. We choose to report this micro-averaged precision because it is more sensitive to errors in frequent classes.

Table 2. Statistics of Data Collection Experiment

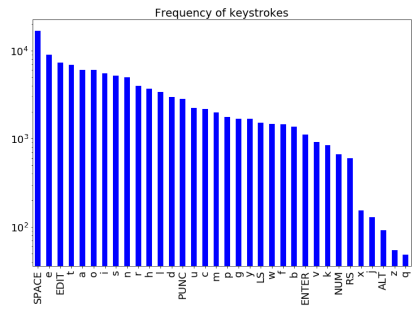
Dataset Size (raw data, time)	69 GB (19 hours)
Keystrokes Collected (n, %)	Survey (13087, 8.1%), Interview (149207, 91.9%), Total (162294, 100%)
Keystrokes by Keyboard (n, %)	Laptop (65804, 40.5%), Mechanical (48291, 29.7%), Bluetooth (48199, 29.7%)
Typing Speed (characters/min)	median = 340.0 cpm (270 - 410)
Overlapping Keystrokes per Typist (μ%, $1.96 \cdot \sigma$)	28.5% (±25.1%)
Time Talking per Session (μ%, $1.96 \cdot \sigma$)	46% (±7.8%)
Time Typing per Session (μ%, $1.96 \cdot \sigma$)	81% (±10.4%)
Keystrokes per session with while talking (μ%, $1.96 \cdot \sigma$)	41.8% (±10.4%)

5.1 Keystroke Detection

First, we tested the keystroke detection model individually. We trained and tested on the entire dataset with 5-fold cross-validation across participants and averaged the results for each fold. Specifically, this means we train on a cohort of 16 participants and test on the remaining 4 participants. Table 3 details the results of different inputs to the keystroke detection model. Since keystroke detection only has two classes (key pressed vs key not pressed), we report the true positive rate and the true negative rate rather than the accuracy and the micro-averaged true positive rate. The results reported here demonstrate the ability of the model to detect when keystrokes occur. In a realistic scenario, the keystroke detection model would never be used independently of the keystroke classification model. The remainder of the results reported are generated from first training the keystroke detection model, then training the keystroke classification model. For brevity, the best performing keystroke detection model (Motion Input) is used for the remainder of the results.



(a) Histogram of Time Between Keystrokes.



(b) Frequency per Key (logarithmic scale).

Fig. 3. Histograms of Time Between Keystrokes and Frequency of Keystrokes.

Table 3. Results for General Training, Averaged Across Testing Folds. TPR=True Positive Rate, TNR=True Negative Rate

Keystroke Classifier		**TPR**	**TNR**	**Accuracy, Top-1**	**Accuracy, Top-5**	**Precision**
Majority Classifier		50%	50%	14.8%	38.9%	14.8%
CNN	Motion Input	**87.5%**	**76.3%**	16.2%	43.7%	38.8%
	Audio Input	82.4%	74.1%	18.1%	44.9%	42.3%
	FFT Input	82.8%	74.8%	21.3%	47.8%	44.1%
	MFCC Input	74.1%	72.4%	*26.3%*	*55.4%*	*48.3%*
	Combined Input	85.3%	74.9%	24.2%	53.7%	45.7%
RNN+CNN with MFCC		–	–	**41.8%**	**70.6%**	**54.9%**

We note that because the best true positive rate from the keystroke detector is 87.5%, the reported results in remaining analyses always include about 12.5% error. Moreover, the remaining models must also classify any false positives from the detector as *not a key*.

5.2 Keystroke Classification: General Training

For keystroke classification, we first trained and tested on the entire dataset. We used the same cross-validation scheme as when testing keystroke detection. When reporting the RNN results, we use the best performing CNN model as input features. Table 3 shows the results of different input combinations for the CNN as well as RNN. Figure 4a demonstrates how the accuracy of the prediction changes with respect to the Top-N guesses. The results reported here mirror how an attacker might perform if they did not know what style of keyboard was being used or at which location someone was typing in a particular setting, and therefore trained a model generically using multiple locations and multiple keyboards.

Table 3 separates out the results based on the features used. The motion and raw audio features perform no better than majority classification. We hypothesize that motion only data is poor performing because of the low sample rates (100 Hz per phone). For raw audio data, we hypothesize that the poor performance is due to a lack of sufficient data to properly characterize the time-domain filters. Theoretically, CNNs in the time domain can discover frequency-based features—however, this requires many times more data and deeper CNN architectures [25]. Once we employ frequency based pre-processing to the input audio (*i.e.*, FFTs and MFCCs), marked improvements occur, with the MFCCs performing slightly better than the FFT-based features. Based on a two tailed T-test for the different CNN architectures, the MFCC input CNN is statistically the best performer with 95% confidence ($p < 0.01$). Interestingly, using only the MFCCs performed better than combining all the different input features — we speculate that adding other features did not add considerable information that was pertinent for classification.

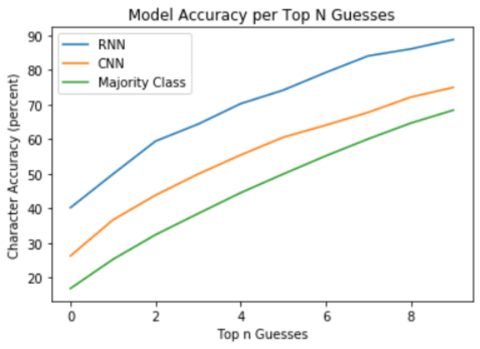
After comparing the different CNN input configurations for the entire dataset, we selected the best performing input configuration (MFCCs only) and performed a gridsearch to find the optimal sequence length for the RNN. To conduct the gridsearch, we adopted a nested cross validation strategy across participants, using 80% of the participants for training, 10% of the participants for selecting the optimal RNN sequence length, and the remaining 10% of the participants for reporting results, repeating this process 10 times. Figure 4b shows the average performance of the RNN models with respect to sequence length. These are also the results reported at the bottom of Table 3. It is clear that the RNN adds considerable predictive capability to the model. For brevity in the remainder of the results, we use the best performing CNN (MFCCs only) and RNN models.

We further examined how the number of phones used as input to our models affected classification accuracy. Our experimental setup included a total of 8 devices: 2 devices are next to the keyboard at any given time, and the other 6 are distributed around a table (as would be common in a meeting). We compared using 1 phone (adjacent to the keyboard), 2 phones (both adjacent to the keyboard), 4 phones (the 4 phones closest to the keyboard), and all 8 phones as inputs. Table 4 shows the performance of the models based on the number of input devices used.

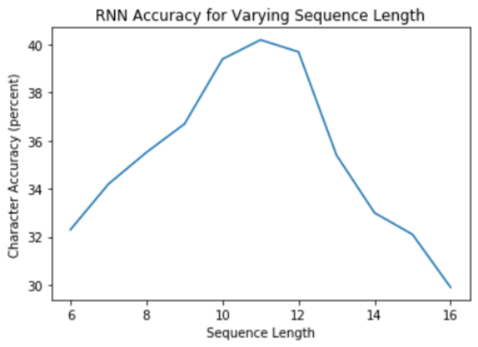
Based on the results, using 4 phones performs the best, and using 8 phones performs slightly worse than using 4 phones, though the difference is not statistically significant. We speculate that the slight decrease in performance with 8 devices is due to added complexities in time synchronization and variation in microphone quality. Because doubling the number of devices from 4 to 8 results in significantly longer training time and added model complexity without improving model accuracy, we use 4 phones for the remainder of the results.

5.3 Keystroke Classification: Per Keyboard Training

Next, we split the data based on keyboard type. This scenario mirrors how an attacker might perform if they knew the style of keyboard that a person used, but was not aware of where they would be located at the table. For each of the three keyboard types used, we performed 5-fold cross-validation within the data collected for that keyboard. Thus, this is only about 33% of the data used for general training. Table 5 shows the performance of the models categorized by keyboard type. The results follow a similar trend to the general training results. The



(a) Prediction Accuracy for Varying N.



(b) RNN Accuracy for Different Sequence Lengths.

Fig. 4. Top-N character level accuracy for CNN and RNN under different configurations.

Table 4. Results by Number of Devices

Number of Phones	Model	Accuracy, Top 1	Accuracy, Top 5	Precision
1 Phone	Majority Class	14.8%	38.9%	14.8%
	CNN	19.8%	43.9%	34.1%
	RNN	30.5%	56.3%	47.3%
2 Phones	Majority Class	14.8%	38.9%	14.8%
	CNN	24.2%	53.6%	44.1%
	RNN	38.2%	67.2%	51.4%
4 Phones	Majority Class	14.8%	38.9%	14.8%
	CNN	26.3%	55.4%	48.3%
	RNN	**41.8%**	**70.6%**	**54.9%**
8 Phones	Majority Class	14.8%	38.9%	14.8%
	CNN	25.2%	52.9%	46.8%
	RNN	40.1%	67.5%	52.1%

Table 5. Results for Keyboard Training

Keyboard Type	Model	Accuracy, Top 1	Accuracy, Top 5	Precision
Laptop, Capacitive	Majority Class	14.4%	38.5%	14.4%
	CNN	28.2%	57.6%	51.1%
	RNN	**43.2%**	**70.2%**	**57.4%**
USB, Mechanical	Majority Class	15.0%	39.5%	15.0%
	CNN	27.4%	55.8%	49.1%
	RNN	38.3%	69.9%	56.6%
Bluetooth, Capacitive	Majority Class	14.7%	39.0%	14.7%
	CNN	19.8%	46.3%	48.3%
	RNN	33.4%	64.3%	48.4%

laptop and mechanical keyboards perform similarly. However, it is clear that the bluetooth capacitive keyboard performs noticeably worse.

5.4 Keystroke Classification: Per Position Training

Finally, we split the data based on where the typist was sitting. This scenario mirrors how an attacker might perform if they knew where they would be located at the table but was not aware of the style of keyboard that a person used. For each location, we performed 5-fold cross-validation for each position at which a typist sat. Because we used two locations, each cross validation uses about 50% of data compared to general training. Table 6 shows the performance of the models categorized by typist location. The results for each location are about the same as in the general training scenarios. Thus we conclude that knowing position of the keyboard is not particularly advantageous to an attacker.

5.5 Word-Level Accuracy

In a realistic attack scenario, the goal is to gather intelligible text rather than individual keystrokes. Thus, we deployed a basic language model to transform the keystroke-level predictions into word-level predictions.

First, we gathered the keystroke-level predictions from our RNN sequence classifier. To eliminate low confidence keys, we selected the top n highest confidence guesses for each keystroke. We investigate a number of values for n (reported below). Next, we segmented the keystrokes into words. We employed two competing methods for word segmentation: a whitespace method and a windowing method. For the whitespace method, we segmented the keystrokes into words wherever the model guessed the "space" key or "enter" key. This model assumes high confidence in whitespace classification. For the windowing method, we tried all combinations of keystrokes in a fixed-sized window. After finding the highest scoring beginning word, we shifted the window over by the length

Table 6. Results for Position Training

Position	Model	Accuracy, Top 1	Accuracy, Top 5	Precision
Position A	Majority Class	14.4%	38.5%	14.4%
	CNN	26.5%	51.5%	44.7%
	RNN	39.5%	69.3%	54.8%
Position B	Majority Class	14.5%	37.8%	14.5%
	CNN	27.1%	53.2%	45.7%
	RNN	40.2%	69.8%	54.9%

of that word. This method is more forgiving if the classifier does not perfectly find whitespace in the keystrokes. Thus, we hypothesized this model might be biased to select short, common words because of their frequency.

After word segmentation, we removed any words that did not appear in our dictionary (we formed the dictionary from the words typed by participants in our study; this limitation is discussed in next section). We then scored each remaining word using competing methods. Table 7 details the scoring methods we tested, where P_{char} is the confidence score for a given keystroke. Based on the scores for the various possible words, we chose the highest scoring word as the word-level prediction of our model. Figure 5a shows a comparison of the different models with varying thresholds for n.

Table 7. Methods Used for Word Scoring.

Sum Method	$score = \sum P_{char}$
Log Sum Method	$score = \sum \log(P_{char})$
Normalized Method	$score = \frac{1}{L}\sum^{L} \log(P_{char})$

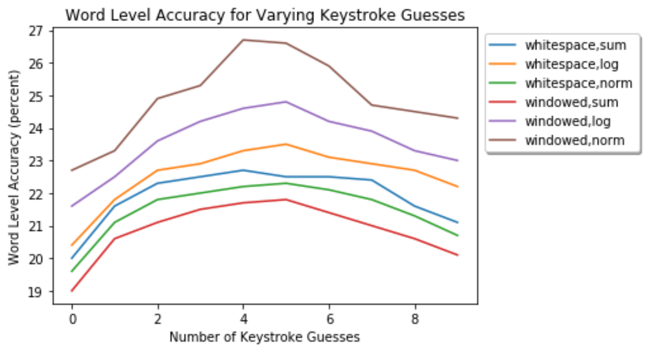
Based on the results, the top performing model uses windowing for word segmentation, normalized word scoring, and a threshold of $n = 5$ letters from the RNN. We further examine the performance of this model by analyzing the Top-N word prediction accuracy. That is, what percentage of the actual typed words are in the top N guessed words from the model. Figure 5b shows this model for varying word guesses, ranging from about 27% accuracy for Top-1 words to 38% accuracy for the top 10 words.

6 VULNERABILITY ANALYSIS

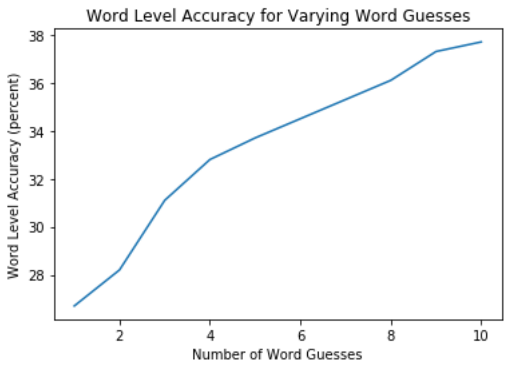
While the models from previous analysis perform well, it is unclear what an attacker must do to achieve this level of performance. For instance, can a model built from data collected on a particular tabletop transfer to another type of tabletop? Can data from a particular keyboard generalize to a similar keyboard? These questions are critical to understanding what level of proficiency an attacker requires to carry out a successful keyboard snooping attack. To mirror a realistic attack scenario, we developed a real-time system that used input data from surrounding phones and provided a prediction of the typed phrases. The pre-trained machine learning models were created from our existing dataset and deployed on a server. This allowed us to collect new typing data in an online machine learning system and test the performance over a variety of live scenarios. For testing the online system, a single typist (not included in the training data) typed a known paragraph containing 85 words and 443 characters. We collected new typing data while varying a number of additional parameters including the tabletop and the location of the phones. For each trial one parameter was changed while the rest were kept constant. We report the results of the online system over five trials for each set of parameters.

6.1 Room Variation

In the first trial we varied the room the participant typed in, while keeping the tabletop typed upon identical to the tabletop where training data was collected. Table 8 shows the performance based on the room. The results indicate that varying the room has an impact on model accuracy, but not a detrimental one. It is unsurprising that the room effects the accuracy of the model because each room has different acoustic characteristics that influence how the audio from the keystrokes reaches the phone microphone. However, the source-filter model analogy speculates that the tabletop is the primary filtering process. Thus, changing the room but maintaining the same tabletop yields small changes in the model accuracy. We conclude that it is not critical for an attacker to gain access to the specific room to launch a successful attack.



(a) Comparison of Word-Level Accuracy for Different Methods



(b) Word-Level accuracy for Varying Top Word Guesses

Fig. 5. Word level accuracy under various language model configurations and for Top-N predictions.

Table 8. Results for Varying Rooms

Room	Model	Character Accuracy	Word Accuracy	Word Accuracy, Top-5
Original Room	RNN	39.5%	14.1%	27.1%
	Language Model	32.3%	25.9%	32.9%
New Room	RNN	34.2%	12.9%	24.7%
	Language Model	27.8%	21.4%	28.3%

6.2 Table Variation

In the second trial we varied the tabletop the participant typed on. We compared the results from the original wooden tabletop to two new tabletops: one made of a composite faux-wood and one made of metal (in the style of a lab table). Table 9 shows the performance based on the tabletop. The results indicate that varying the tabletop has a significant impact on model accuracy. In particular, the results for the metal tabletop are worse than the results for the composite tabletop. Using our source-filter model we hypothesize that these results are due to the different transfer functions of the different materials. Because the metal tabletop is extremely different from the original wooden tabletop (the material is entirely different and the thickness of the tabletop is dramatically different) but the composite tabletop is somewhat similar to the original wooden tabletop (the material is similar and thickness is the same), the results are consistent with the source-filter model. We conclude that prior knowledge of the tabletop surface material is critical to a successful attack at the present and an attacker requires this knowledge. It is possible that a model trained on data from multiple tabletops will be able to overcome this obstacle, or that a different model could be trained for each type of tabletop, but the current results indicate that a model trained on data from one type of tabletop cannot generalize to other types of tabletops.

6.3 Phone Location Variation

Next, we varied the phone location and number of phones used, keeping constant the table, room, and keyboard. We compared the results from the original phone locations to two new phone locations: systematically shifted and randomly shifted. In the systematically shifted scenario, every phone is moved 1 inch horizontally further from the keyboard than the original location and tilted 15 degrees counterclockwise from the original location. The systematically shifted location is designed to test the resilience of the model to minute changes in phone position. In the randomly shifted scenario, every phone is moved between 1-3 inches horizontally further from the keyboard

Table 9. Results for Varying Tables

Table	Model	Character Accuracy	Word Accuracy	Word Accuracy, Top-5
Original Table	RNN	39.5%	14.1%	27.1%
	Language Model	32.2%	25.9%	32.9%
Composite Table	RNN	18.7%	7.1%	10.6%
	Language Model	15.9%	8.2%	15.3%
Metal Table	RNN	16.9%	1.2%	3.4%
	Language Model	13.2%	2.4%	5.9%

Table 10. Results for Varying Phone Locations

Location	Phones	Model	Char. Accuracy	Word Accuracy	Word Accuracy, Top-5
Original	2 Phones	RNN	37.7%	12.6%	26.2%
		Lang. Model	31.6%	24.2%	30.3%
	4 Phones	RNN	42.0%	14.5%	27.6%
		Lang. Model	32.6%	24.5%	31.6%
Systematically Shifted	2 Phones	RNN	34.6%	9.8%	23.9%
		Lang. Model	30.0%	20.2%	27.7%
	4 Phones	RNN	36.7%	10.9%	24.3%
		Lang. Model	29.5%	20.6%	28.4%
Randomly Shifted	2 Phones	RNN	30.7%	7.9%	18.5%
		Lang. Model	24.8%	17.3%	24.8%
	4 Phones	RNN	31.6%	9.8%	22.5%
		Lang. Model	22.8%	18.2%	25.3%

than the original location, and tilted 5-20 degrees clockwise or counterclockwise from the original location. The randomly shifted location is designed to emulate a more realistic attack scenario, such as eavesdropping on a meeting, wherein phones are casually strewn on the table in random locations and orientations.

Table 10 shows the performance of the three different locations using two phones and four phones. The results indicate that phone location does impact model accuracy, but not dramatically so. Unsurprisingly, the systematically shifted scenario yields better performance than the randomly shifted scenario, presumably because the difference from the original phone locations is smaller. It is also worth noting that varying the location of the phones degrades model performance more when more phones are present; varying the location affected the four phone setup more than the two phone setup. We speculate that this is due to the fact that the two additional phones added in the four phone setup are already far from the keyboard. Because these phones are farther away, they are more sensitive to increases in noise and benefit from being aimed more directly at the keyboard. Because the two primary phones are so close to the keyboard, varying the orientation is less impactful. We conclude that phone location is helpful for an attacker to know, but not critical to a successful attack.

6.4 Keyboard Variation

Finally, we varied the keyboard typed on, keeping constant the number and location of phones used and the table. We compared the results from the original laptop keyboard to two new keyboards: a laptop keyboard and an ergonomic keyboard. The new laptop used was a different model from the original laptop, but used the same chiclet style keyboard. The ergonomic keyboard differed from all of the keyboards seen in the training data: it is split down the middle such that there is a sizable gap between the left-hand and right-hand sides of the keyboard. Table 11 summarizes the performance of the model on the three different keyboards. The results indicate that the model has some difficulty generalizing to the new laptop keyboard and considerable difficulty generalizing to the new ergonomic keyboard. These results are unsurprising for a number of reasons. First, the new laptop keyboard uses the same style of keys as the old laptop keyboard, so it is similar, but the size of the keyboard and the resistance of the keys are both slightly different. The ergonomic keyboard, on the other hand, is entirely different from the keyboards used in the training data. The location of each key on the keyboard is an important factor for the model, and these locations are altered drastically by the unconventional layout of this keyboard. We conclude that an attacker can achieve success if they have knowledge of the general style of keyboard.

7 DISCUSSION

We had a number of different goals in testing different models. In particular, we wanted to determine which inputs yield the best convolutional output; whether or not the recurrent network improved on the convolutional output; how changing the keyboard model and the typist location effected results; and how well our best model performs on the word-level.

Based on the results from general training on the entire dataset, the MFCCs are the best input to the convolutional neural network. We observed that MFCC processed audio performed better than FFT processed audio, which in turn performed better than unprocessed raw audio. This indicates that inputs perform better using traditional pre-processing. Considering that MFCCs were designed for speech recognition rather than keystroke detection, it is possible that, with more data and more training, the network could extract features from raw or FFT processed audio that are even more relevant to keystroke detection than MFCCs.

We further found, via gridsearch, that an RNN with a sequence length of 11 performed the best. We found that the accuracy of the RNN first increases, then decreases for larger sequence lengths. We speculate that the accuracy increases because longer sequence lengths allow the RNN to uncover longer-term dependencies in the text. Eleven keystrokes, the optimal sequence length we found, is large enough to encompass multiple words, allowing the RNN to learn frequent n-grams. The decrease in accuracy for larger sequence length is likely due to available training data. Larger sequence lengths require more data for proper training, and (when not overlapping) reduce the number of total sequences to train the network. The best performing RNN with a sequence length of 11 improved the CNN results from 26.3% accuracy to 41.8% accuracy. It is clear from the magnitude of this improvement that using the RNN is worth the additional training time and model complexity. With more data, it is possible that a larger sequence length could be used with even better results.

Additionally, the best performing CNN/RNN combination combines input signals from 4 different mobile devices. Prediction accuracy decreased dramatically when using fewer devices, indicating that an attacker will have much better results by using an array of devices. These findings thus suggest that keyboard snooping is a more realistic attack in settings that involve multiple mobile devices, such as meetings.

Table 11. Results for Varying Keyboards

Keyboard	Model	Character Accuracy	Word Accuracy	Word Accuracy, Top-5
Original Laptop	RNN	42.2%	15.1%	28.0%
	Language Model	32.8%	25.0%	30.9%
New Laptop	RNN	30.8%	11.2%	26.3%
	Language Model	26.5%	19.4%	25.3%
Ergonomic	RNN	14.8%	3.9%	12.6%
	Language Model	12.2%	8.9%	15.2%

By splitting the data based on keyboard type we observed that the laptop keyboard is most vulnerable to attack whereas the bluetooth keyboard is least vulnerable. Even so, the RNN is able to achieve 28% typing accuracy, indicating that using a sound dampening keyboard can help to mitigate the likelihood of a successful eavesdropping attack, but does not guarantee that the attack will fail. These results are consistent with our source-filter model considering the greater amount of dampening employed by the bluetooth keyboard.

Splitting the data based on typist position yielding similar results to training and testing irrespective of position; thus, we cannot confidently say that varying the position of the typist affects model accuracy. This indicates that the model is able to generalize to different typist positions.

Finally, we investigated the accuracy of our model at the word level. We tested using the top n predictions from the recurrent neural network for varying n, two different word segmentation methods, and three different word scoring methods, achieving 26.7% accuracy in the best case. We achieved this accuracy score by using a dictionary constructed from words typed during our experiment (a vocabulary size of around 15,000 words), which may or may not be feasible in a realistic attack scenario. We did not restrict our typists to a known vocabulary, but did suggest topics for discussion, leading many of our participants to use similar keywords. In situations where the attacker can guess at which keywords might be typed, an attacker can feasibly achieve similar accuracy to this study. When using an unrestricted vocabulary of English words, the word level accuracy drops to 19.4%, which is still reasonably accurate. Note: this accuracy is somewhat arbitrarily low because many typists had typographical word errors, which are not recoverable using a standard dictionary.

For varying n, the best result did not occur with the largest n value. Rather, as n increased, the word-level accuracy first increased, then decreased (see Figure 5a). We hypothesize that this is caused by mistakes arising for increasing n. For low n values, all possible keystrokes generated by the model are high confidence. For higher n, the model returns lower confidence guesses. When these bad guesses are considered, the accuracy decreases because the model by chance alone guesses a word that is in the dictionary but is not the correct word.

For varying segmentation and scoring methods, we found the best results using windowed segmentation and normalized word scoring. Using the windowed segmentation, we are better able to search for all possible words. The whitespace method is limited because if the model miss-predicted where the space occurs, the word segmentation will fail. Although the windowed method requires greater resources and longer runtime, this added complexity appears to be justified because windowed segmentation outperforms whitespace segmentation by nearly 6% word accuracy overall (Figure 5a).

The real-time analysis of the system revealed a number of key insights into the requirements of a successful attack. We discovered that an attacker requires considerable knowledge of the table, as well as some knowledge about the style of keyboard, while the location of the phones and the room configuration is less critical. Through this analysis, we conclude that keyboard snooping attacks are only a threat to skilled attackers that have knowledge about the setup before hand. This greatly limits the feasibility of the attack not only because the attacker needs knowledge about the table and keyboard beforehand, but also because the attacker must have the requisite skills for processing and deploying a new machine learning model. It is foreseeable that these disadvantages could be mitigated through additional data collection or through additional pre-processing techniques. However, such techniques would require a sophisticated attacker to carry out.

8 LIMITATIONS AND FUTURE WORK

The real-time analysis of the system revealed that the current model has great difficulty generalizing to different tables. It is possible that using a larger set of training data, one that includes multiple tables and rooms, could result in a model that can generalize to different setups more easily. Furthermore, both the real-time analysis and the primary data collection included few users. A larger set of training data in the future would benefit from increasing the number of users, particularly for the real-time analysis.

Though we achieve keystroke accuracy of 41%, it is unlikely that the approach presented here could be used for snooping passwords. This is because the recurrent network takes advantages of common phrases and letter sequences that are typically lacking in strong passwords, so we anticipate that predicting a password would have an accuracy more similar to the CNN model, where about 28% of keystrokes are correct. We assume this accuracy is not sufficient for successful password snooping. It may be possible, however, to reduce the search space of possible passwords with our model, though we have not investigated such a vulnerability in this paper.

Finally, future work could investigate the performance benefit of typist and/or keyboard specific models. In our experiments we did not take advantage of similar typists. For example, some of our participants were experienced touch typists, whereas others were inexperienced and used the hunt-and-peck method. Future studies with more participants could take advantage of these groups by training and testing on different typists within the same typist group, creating group-specific models. Furthermore, we observed varied model performance across the three different keyboard types used in data collection. As per our independent source-filter model, each type of keyboard could be considered a different type of source. Our experiments demonstrate that training one model on data from multiple keyboard types is sufficient, but it is possible that training keyboard-specific models could result in better performance.

9 CONCLUSION

We developed a system that uses sensor data from mobile devices to determine what keys were pressed on a nearby keyboard. In an IRB-approved human subject experiment that involved 20 participants typing over 160,000 characters, we achieved 41.8% accuracy at the keystroke level and 26.7% accuracy at the word-level by using MFCC-processed audio data. Our model generalized to different participants, different keyboards, and different locations. We achieved this accuracy on unconstrained typing data in a noisy environment with commercial smartphone sensors, demonstrating the feasibility of this attack in a realistic setting. However, through building a prototype system, we also highlighted the limitations of the system to data collected outside of a specific room or table, and on a different style keyboards, limiting the practicality of such an attack to sophisticated personnel.

REFERENCES

[1] Apple developer documentation: mach-absolute-time. https://developer.apple.com/documentation/kernel/1462446-mach_absolute_time. Accessed: 2018-07-01.

[2] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10):1533–1545, 2014.

[3] Kamran Ali, Alex X Liu, Wei Wang, and Muhammad Shahzad. Keystroke recognition using wifi signals. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 90–102. ACM, 2015.

[4] Liang Cai and Hao Chen. Touchlogger: inferring keystrokes on touch screen from smartphone motion. In *Proceedings of the 6th USENIX conference on Hot topics in security*, pages 9–9. USENIX Association, 2011.

[5] Liang Cai and Hao Chen. On the practicality of motion based keystroke inference attack. In *International Conference on Trust and Trustworthy Computing*, pages 273–290. Springer, 2012.

[6] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[7] Li Deng, Geoffrey Hinton, and Brian Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8599–8603. IEEE, 2013.

[8] Gunnar Fant. *Acoustic theory of speech production: with calculations based on X-ray studies of Russian articulations.* Number 2. Walter de Gruyter, 1970.

[9] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[10] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[12] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
[14] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. Convolutional neural network architectures for matching natural language sentences. In *Advances in neural information processing systems*, pages 2042–2050, 2014.
[15] Biing-Hwang Juang, L Rabiner, and JG Wilpon. On the use of bandpass liftering in speech recognition. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'86.*, volume 11, pages 765–768. IEEE, 1986.
[16] Andrew Kelly. Cracking passwords using keyboard acoustics and language modeling.
[17] Xiangyu Liu, Zhe Zhou, Wenrui Diao, Zhou Li, and Kehuan Zhang. When good becomes evil: Keystroke inference with smartwatch. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1273–1285. ACM, 2015.
[18] Chris Xiaoxuan Lu, Bowen Du, Hongkai Wen, Sen Wang, Andrew Markham, Ivan Martinovic, Yiran Shen, and Niki Trigoni. Snoopy: Sniffing your smartwatch passwords via deep sequence learning. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(4):152, 2018.
[19] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.
[20] Philip Marquardt, Arunabh Verma, Henry Carter, and Patrick Traynor. (sp) iphone: decoding vibrations from nearby keyboards using mobile phone accelerometers. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 551–562. ACM, 2011.
[21] Maryam Mehrnezhad, Ehsan Toreini, Siamak F Shahandashti, and Feng Hao. Stealing pins via mobile sensors: actual risk versus user perception. *International Journal of Information Security*, 17(3):291–313, 2018.
[22] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.
[23] David L Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on communications*, 39(10):1482–1493, 1991.
[24] Ben Milner. A comparison of front-end configurations for robust speech recognition. In *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*, volume 1, pages I–797. IEEE, 2002.
[25] Abdel-rahman Mohamed. *Deep neural network acoustic models for asr.* PhD thesis, 2014.
[26] Lindasalwa Muda, Mumtaj Begam, and Irraivan Elamvazuthi. Voice recognition algorithms using mel frequency cepstral coefficient (mfcc) and dynamic time warping (dtw) techniques. *arXiv preprint arXiv:1003.4083*, 2010.
[27] Alan V Oppenheim and Ronald W Schafer. *Discrete-time signal processing.* Pearson Education, 2014.
[28] Javier Ram i rez, Jos e C Segura, Carmen Ben i tez, Angel De La Torre, and Antonio Rubio. Efficient voice activity detection algorithms using long-term speech information. *speech communication*, 42(3-4).
[29] Bharath Ramsundar, Steven Kearnes, Patrick Riley, Dale Webster, David Konerding, and Vijay Pande. Massively multitask networks for drug discovery. *arXiv preprint arXiv:1502.02072*, 2015.
[30] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *Artificial Neural Networks–ICANN 2010*, pages 92–101. Springer, 2010.
[31] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.
[32] Laurent Simon and Ross Anderson. Pin skimmer: Inferring pins through the camera and microphone. In *Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices*, pages 67–78. ACM, 2013.
[33] Laurent Simon, Wenduan Xu, and Ross Anderson. DonâĂŹt interrupt me while i type: Inferring text entered through gesture typing on android keyboards. *Proceedings on Privacy Enhancing Technologies*, 2016(3):136–154, 2016.
[34] Fikret Sivrikaya and Bülent Yener. Time synchronization in sensor networks: a survey. *IEEE network*, 18(4):45–50, 2004.
[35] Peter Welch. The use of fast fourier transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms. *IEEE Transactions on audio and electroacoustics*, 15(2):70–73, 1967.
[36] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.
[37] Zhi Xu, Kun Bai, and Sencun Zhu. Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors. In *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, pages 113–124. ACM, 2012.
[38] Tuo Yu, Haiming Jin, and Klara Nahrstedt. Writinghacker: audio based eavesdropping of handwriting via mobile devices. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 463–473. ACM, 2016.
[39] Li Zhuang, Feng Zhou, and J D. Tygar. Keyboard acoustic emanations revisited. 13, 01 2009.

Received November 2018; revised February 2019; accepted April 2019