# Kreeper: Keystroke Acoustic Emanations Revisited (Again)

Jacob Hage
jakehage@umich.edu

Cristina Noujaim
cnoujaim@umich.edu

Diego Rojas
darojas@umich.edu

University of Michigan

December 2019

## Abstract

*We show that keyboards are vulnerable to attacks based on differentiating the sounds emanated by each individual key press. We present* ***Kreeper****, a method to extract secret information from recorded keystrokes. Our attack automatically detects key presses, extracts features, classifies keystrokes using audio data, and corrects classification using a language model. With our automated methods, we achieve 88.8% character level accuracy and 46.1% word level accuracy. With human-intervention methods, we achieve 91.3% character level accuracy and 88.2% word level accuracy.*

## 1 Introduction

Keyboard side-channel attacks are a growing field of research. The hypothesis behind this area of study is as follows: given enough information about the user's keystroke timing, an attacker can retrieve a significant amount of information about what was typed. With simple timing information, Song et. al.[6] were able to reduce the entropy of passwords such that the number of passwords required to check in a brute-force attack is reduced by 50x. Although this information is imperfect, it makes discerning what was typed by a user significantly easier.

Retrieving timing information can happen in many ways: over SSH [6], through Javascript [4], and even through recordings of acoustic emanations [7]. Once the attacker has timing information, they can perform the same (or similarly structured) attack of predicting what was typed. In order to make this conversion, either statistical analysis of the user's typing patterns or machine learning is applied. With enough data on any given user, Zhuang et. al. [7] shows that one can retrieve up to 96% of typed characters, and 80% of 10-character passwords in under 75 attempts.

We attempt to revisit and iterate on these findings in a new scenario. If the attacker is sitting next to a target, and can discreetly record the typing of that individual, can they recover a significant amount of English language information?

We propose an approach that uses microphone recordings of a mechanical keyboard. Our method consists of four parts. First, we extract individual keystrokes from the recording of the keyboard. Then, we extract individual frequency features for each keystroke using Mel Frequency Cepstrum Coefficients. With these features (and ground truth for what is actually typed), we train a neural network classifier to detect the different features of each key. Finally, with an unlabeled recording, we refine our prediction using a model of the English language, predicting the most likely sequence of words given the neural network probabilities.

With 25 minutes of recorded labeled data, we are able to correctly recover 88.8% of characters typed by the user, and 46.1% of words. With the addition of human intervention, we achieve 91.3% character level accuracy and 88.2% word level accuracy.

## 2 Related Work

Song et. al [6] first introduced the idea of keystroke timing attacks, through SSH. They showed that the SSH protocol leaks information about the timings of key presses when entering a password, and this timing information may be used to extract the provided text. In particular, the approach exploits two vulnerabilities in SSH for password extraction. The first is that the SSH-transmitted packets are padded only to an eight-byte boundary (if a block cipher is in use), which reveals the approximate size of the original data. The second is that in interactive mode, every individual keystroke that a user types is sent to a remote machine in a separate IP packet immediately after the key is pressed, which leaks inter-keystroke timing information of users' typing. From these two sources of information, a Gaussian Mixture Model is generated to estimate each key pressed, and a Hidden Markov Model is generated for all keys pressed, to infer the most likely sequence of keys pressed by a user.

Zhuang et. al [7] extend these methods using a four-step process. First, they extract timing and audio information from keystroke recordings using Cepstrum features, a common feature extraction method used in voice recognition. Then, they apply unsupervised clustering to the information to understand the distributions for each key. With that clustering information, they can predict the typed key based on the cluster it falls into. They refine their prediction using HMM, which take into account the previous key. This allows keys between clusters to be correctly classified. Then, they apply a simple spell-check algorithm to further correct their predictions. Finally, they use their best unsupervised predictions (the ones that did not require spell-check) to train a simple machine learning model to apply to the data. With all techniques, they were able to achieve 92% character-level accuracy.

Halevi and Saxena [3] studied keyboard attacks for cases in which attackers only utilize raw acoustic information which has been recorded. Their focus is keyboard acoustic emanations specifically for the purpose of retrieving random passwords, where a dictionary attack or an HMM language model would not help. An important part of their work was considering different typing styles (hunt-and-peck vs touch typing); previous work had focused only on hunt-and-peck. In order to analyze the audio, they developed a new signal processing technique called time–frequency decoding, which combines time domain data as well as frequency features for improved detection results.

## 3 Kreeper: The Attack

Our attack, Kreeper, consists of four parts:

1. Extracting keystrokes from an audio recording of a typing user
2. Extract features from individual keystrokes
3. Train a neural network classifier to detect the difference between sounds of each key
4. Refine our classification prediction using spell check

### 3.1 Hardware & Experimental Method Specifications

We describe preliminaries of our experimental setup.

**Keyboard.** To gather data, we used a single Razer Black Widow mechanical keyboard for typing.

**Typing Methods.** One user typed on this keyboard and collected all data within a one-hour period, reducing noise variation. Within that period, the user both typed one key at a time and key sequences, to further reduce noise variation. The user typed using hunt-and-peck typing style as in [7] and [1].

**Microphone.** We used a Brüel & Kjær (B&K) Type 4190 microphone with a B&K Type 2669 preamplifier on a B&K Type 2610 amplifier into a MOTU 828 MkII audio interface. This captured the sound of the keyboard into Audacity, an audio recording & editing interface.

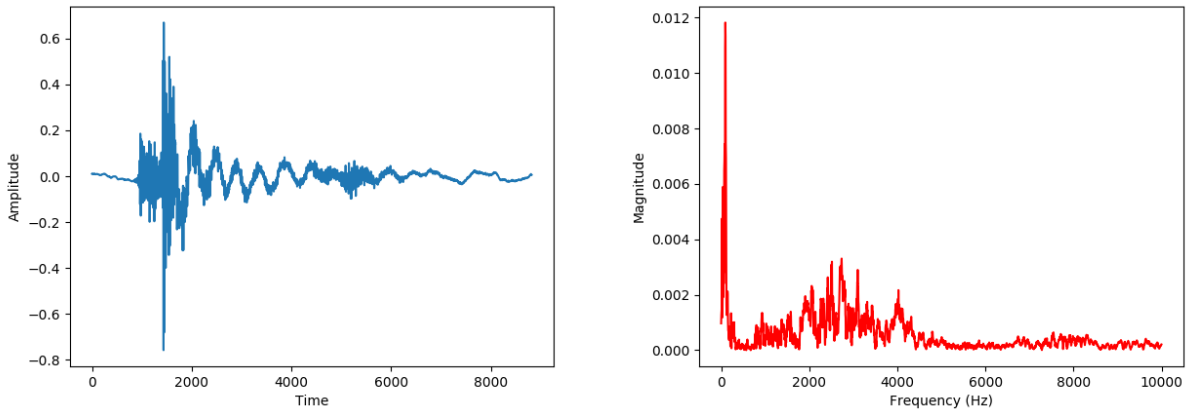


Figure 1: On the left: the acoustic signal of one click. On the right: frequency spectrum of the entire click

### 3.2 Extracting Keystrokes from Audio Recording

Our first step takes a raw recording of a target typing on a keyboard and extracts the individual keystrokes from the recording. Users can type up to on average 300 keys per minute. Each keystroke consists of two major sound bytes - a push peak, where the user presses the key, and a release peak, where the user releases pressure off of the key. Asonov et. al. [1] proved that the average user has about 100ms between the push peak and the release peak. This means that each keystroke must be at least 100ms long. In our experiments, we found that the distance between keystrokes is at least 200ms long. We decided to use both the push and release peak for our experiments. In Figure 1, both the acoustic signal of a single click and the frequency spectrum of that click are displayed.

In order to distinguish between each keystroke, we use a filter that detects high peaks in the audio signal. We found that the minimum keystroke magnitude was above 0.06 and noise magnitude was around 0.03, so we filtered out peaks that exceed noise magnitude. Since each push peak must be at least 100ms from release peak and each keystroke must be 200ms, we only distinguished peaks that were 200ms apart. This gave us 99.95% accuracy on detecting keystrokes, making it nearly perfect at detection, significantly better than Zhuang et. al [7].

### 3.3 Extracting Features from Keystrokes

With each individual keystroke detected, we then could extract features from these keystrokes. As in Zhuang et. al [7], we use Cepstrum features, an expansion on FFT features that is empirically tested and shown to be more effective on voice signals (SPEECH). In specific, we use the Mel Frequency Cepstrum Coefficients (MFCCs). In order to benefit from their experiments, we used the same setup, with 16 MFCC filters computed using 10ms filters shifted 2.5ms each time over our 200ms keystroke recording.

### 3.4 Training a Neural Network Classifier

We then train a neural network classifier to distinguish between keys in the recording. This step is crucial, as it allows us to reconstruct the typed information past simple features, and understand human-undetectable difference between keys.

In order to test that there is indeed a tangible difference between the sound of each keys, we trained a two hidden-layer neural network to detect between the ‘a’ and ‘j’ key on the keyboard. With 10 presses of the ‘a’ key and 10 presses of the ‘j’ key, the network was able to perfectly classify a key press as either ‘a’ or ‘j’. This result extended to 5-key classification, as we were able to achieve 100% classification accuracy with 100 presses of 5 keys (‘space’, ‘z’, ‘y’, ‘x’, and ‘w’).

**Typed Sentence**: but what factor or combination of factors may have caused the famines in the eastern mediterranean during these decades remains uncertain elements that might be consoidered include war and plagues of insects but climate change accompanied by drought is more likely to have turned a once verdant land into an arid semidesert

**Predicted Sentence**: but what rrcton or cozbonttior oc fyctorz ran huse rxured tie eatiubd kn rhe psutern uefiterrauetu durink tibze tecadrs renaeuf uncertaen ekbubnts tiat niirt be consxixerer ircluxe wgr rnd plaguex oc insects but critaje charge arcguparied ru ygought is rore liseku to hsrw turuex s once rerdajt kajr into xn arid sbjisesert

Figure 2: A typed sentence and the prediction out of the neural network

We then applied this neural network to all 27 characters (a-z & space), achieving up to 88.8% validation accuracy in under two hours of training. The validation data consists of data both typed one at a time and sequentially with significantly more one at a time data.

When applied to a natural language typed sentence (our test data), we achieved 71.5% character-level accuracy (using the top prediction for each character) and 21.1% word-level accuracy. Although this is lower than the validation accuracy, it still produces a sentence that gives significant information about the users typing, as shown in Figure 2.

In addition, our prediction correctly identified 100% of spaces. Although this is a fraction of typed characters, it is fundamental to readability.

### 3.5 Improve Predictions using Spell Check

After keystrokes are initially classified by our model, the predicted keys undergo a "refinement" process in the form of spellchecking.

Through experimentation, we find that a combination of two different spell-checking methods yields the best word-level accuracy. The first spell checker, pyspellchecker, is a uses a Levenshtein Distance algorithm to find permutations within an edit distance of 2 from the original word. It then compares all permutations (insertions, deletions, replacements, and transpositions) to known words in a word frequency list. Those words that are found more often in the frequency list are more likely the correct results.

Stock spell checkers like pyspellchecker are limited in the kinds of spelling errors they can handle (e.g. at most two letters wrong in a word.) Because these forms of spell checkers are designed to cope well with the common human typist errors, they are limited in their abilities to detect errors that acoustic emanation classifiers make. Hence, we find that a combination of a canonical stock spell checker and a probability-based spell checker yields the best results for extracting acoustic keyboard emanations.

We use a second spell checker, Aspell, in conjunction with pyspellchecker. Unlike pyspellchecker, Aspell uses a posterior probability-based approach to detecting spelling errors, for each character within a misspelled word. That is, the conditional probability of the recognized word $\mathbf{Y}$ (the corresponding string returned by the recognizer, not necessarily a correct word) given each dictionary word $\mathbf{X}$, is computed as

$$p(\mathbf{Y}|\mathbf{X}) = \prod_{i=1}^{\text{length of } \mathbf{X}} p(\mathbf{Y}_i|\mathbf{X}_i)$$

Using these two spell checkers, our refinement approach is as follows:
For every word in the input set:

1. If the word a valid English word, add word to out set, continue
2. Call pyspellchecker correction on word;

**Predicted Sentence:** but what rrcton or cozbonttior oc fyctorz ran huse rxured tie eatiubd kn rhe psutern uefiterrauetu durink tibze tecadrs renaeuf uncertaen ekbubnts tiat niirt be consxixerer ircluxe wgr rnd plaguex oc insects but critaje charge arcguparied ru ygought is rore liseku to hsrw turuex s once rerdajt kajr into xn arid sbjisesert

**Refined Sentence:** but what reckon or combination ox factors ran huse regret tie Ayyubid kn rhe pattern uefiterrauetu during tubes decades reunify uncertain cabinets that night be consxixerer include war and plagues ox insects but cartage charge accompanied Ru thought is rote Lusaka to hare turned s once verdant Kara into in arid subsidiser

Figure 3: The predicted and the refined sentences

| Stage | One-At-A-Time | English Character-Level | English Word-Level |
|---|---|---|---|
| Random | 3.7% | - | - |
| Frequency | 27% | - | - |
| After Classifier | 88.8% | 71.5% | 21.1% |
| After Refine | - | 74.6% | 46.1% |
| Human Intervention | - | 91.3% | 88.2% |

Table 1: Accuracy of all methods on different types of data. Random is defined as if a character was picked at random. Frequency is defined as if a character was picked at random weighted by the frequency of letters in the English language.

   (a) If the returned word is a valid English word whose length matches the length of the original word, add word to out set, continue

3. Call Aspell correction on word;

   (a) If the returned word is a valid English word whose length matches the length of the original word, add word to out set, continue

4. Add original (misspelled) word to out set.

With this method, we are able to achieve 74.6% character-level accuracy on an English language sentence and 46.1% word-level accuracy.

### 3.6 Adding Human Intervention

While our methods by themselves only achieve at most 46.1% word-level accuracy on an English language sentence, we hypothesized many of the word mistakes made are trivially correctable by the eyes of a human. To test this, we had a human who had not seen the original text try to retrieve the full sentence. With only the predicted sentence and the top 4 likely characters for each letter (as predicted by the neural network), they could retrieve 88.2% of words and 91.3% of characters in exactly 30 minutes (see Figure 4.) With human intervention, an attacker achieves its goal of understanding the maximum amount of information typed by the user.

### 3.7 Results

We present the final table of our results in Table 1.

# 4 Discussion

In the following section, we discuss our experiments, our failed methods, and what did and didn't work.

## 4.1 Changing our Approach

In our project proposal, we proposed recreating Zhuang et. al [7] approach to keystroke extraction. In their approach, they use many of our same approaches to extraction, including extracting keystrokes and features. However, their key deviation is using unsupervised learning instead of supervised learning to understand keystrokes. In addition, their paper claims that with only 10 minutes of hunt-and-peck typing, their methods could extract 60% of characters from the audio simply using clustering & Hidden Markov Models (HMMs). However, when repeating their exact experiment, with the same length of recording, and controlling for noise, we were not able to recreate their results. In fact, our accuracy with their method never reached better than random.

With this discovery (and limited time to recreate their methods) we decided to combine their method results with Asonov et. al [1]. We used the same automatic key extraction and featurization methods, while using the supervised machine learning approach of Asonov et. al [1]. In addition, we added Zhuang et. al's [7] refine approach to improve upon our results. This yielded significantly better results than our original attempt.

In addition, the original methods from Zhuang et. al [7] do not actually produce the idealistic results they present in their conclusion. Unsupervised methods only provided 87% character-level accuracy. Only with supervised methods (inspired by Asonov et. al [1]) were they able to achieve the astronomically high 92% character-level accuracy they claimed. Therefore, it was better for our team to approach the problem from a supervised learning perspective.

Although our approach deviated slightly from our original proposal, it significantly improved our ability to extract information from the user. Not only did we recreate the results of Asonov et. al [1], we also improved on their results using a combination of other methods.

## 4.2 Data Collection

Collecting the proper data for the above attack was a difficult process. In the original paper we tried to recreate, they claimed that 10 minutes of typing data on a typical quiet keyboard would be enough to extract up to 92% character-level accuracy. However, with this setup, we were not able to achieve accuracy better than 27%. Therefore, we decided to collect more data over a longer period of time. To train our model, we collected 25 minutes of typing data on a mechanical keyboard. This yielded significantly better results than a normal keyboard.

## 4.3 Typing Methods

In our experiments, we discovered that one at a time typing methods were significantly easier to classify than sequential typing methods. For example, classifying a sequence such as "aaaabbbbcccc" was much easier than classifying a typed sentence. For one at a time typing, we were able to achieve 88.8% character-level accuracy from the classify stage. However, with sequential typing, we were only able to achieve 71.5% character-level accuracy in the classify stage. This is likely because the majority of our data is one at a time typing, as it is much easier to get a balanced dataset. We believe that with more time and sequentially typed data, we could achieve a higher accuracy on real English language out of the classify stage.

## 4.4 Using HMMs to Improve Classification

As in Zhuang et. al [7], we tried to use HMMs to improve the accuracy of our predictions. However, at a character level, we found HMM's are not powerful enough to add information to our predictions. Even when applied after our neural network, which provided 71.5% accuracy, and trained on 5 million english words (20 million characters), the HMM only produced random output. We therefore decided to pursue a different method of refining described above.

**Typed Sentence:** but what factor or combination of factors may have caused the famines in the eastern mediterranean during these decades remains uncertain elements that might be consoidered include war and plagues of insects but climate change accompanied by drought is more likely to have turned a once verdant land into an arid semidesert

**Human Generated Sentence:** but what factor or combination of factors may have caused the eatiubd in the eastern mediterranean during these decades renaeuf uncertain ekbubnts that niirt be considered include war and plagues of insects but climate change accompanied by drought is more likely to have turned a once rerdajt land into an arid sbjisesert

Figure 4: The original sentence and the human generated sentence using the output of the neural network and the top 4-Character predictions

### 4.5 Top $n$-Character Prediction with Human Intervention

While analyzing possible strategies to improve the sentences, we revisited the results of the classifier with the aim of improving the characters that were classified. As was presented above, we used the top prediction per character to construct the predicted sentence with 71.5% character-level accuracy. We decided to look at the top $n$ predictions per character given by the classifier in order to understand which ones could be retrievable from the results of our classifier and which ones could not.

For $n = 3$, 9.6% of the characters were non-predictable, with 42.31% of the words having some non-predictable character, and only 11.54% of them having more than 1 non-predictable character.

For $n = 4$, 6.19% of the characters were non-predictable, with 32.69% of the words having some non-predictable character, and only 5.77% of them having more than 1 non-predictable character.

For $n = 3$, 4.02% of the characters were non-predictable, with 25% of the words having some non-predictable character, and 0% of them having more than 1 non-predictable character.

From this information, we see that a human user could theoretically retrieve greater than 90% of the characters when given the predicted texts with the lists of top $n \in \{3, 4, 5\}$ characters. This is shown in Section 3.6.

## 5 Conclusion

In this paper, we present **Kreeper**, an automatic method to extract keystroke information from a recording of a user typing. Our attack automatically detects keystrokes with nearly zero error or need for human intervention, a significant improvement on Zhuang et. al [7]. Then we extract Cepstrum features and create a supervised neural network classifier that can classify keys with 88.8% accuracy on one at a time typing characters and 71.5% accuracy on sentences. Then, we apply a refining technique that utilizes spell checking to improve our character-level accuracy to 74.6% and word-level accuracy to 46.1%. With human intervention, this accuracy reaches 88.2% word-level accuracy.

Although this was a difficult task, and we were not able to achieve some of the same metrics as the papers we were recreating, we were able to achieve novel improvements in certain areas. Our automatic keystroke detector and classifier character-level accuracy significantly improved on their original counterparts [1][7]. In future work, we hope to revisit the assumptions made in these works and see how to accurately recreate their results and improve upon their methods.

## References

[1] Asonov, Dmitri, and Rakesh Agrawal. "Keyboard acoustic emanations." IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004. IEEE, 2004.

[2] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

[3] Halevi, Tzipora, and Nitesh Saxena. "Keyboard acoustic side channel attacks: exploring realistic and security-sensitive scenarios." International Journal of Information Security 14.5 (2015): 443-456.

[4] Lipp, Moritz, et al. "Practical keystroke timing attacks in sandboxed javascript." European Symposium on Research in Computer Security. Springer, Cham, 2017.

[5] SPEECH VISION AND ROBOTICS GROUP OF THE CAMBRIDGE UNIVERSITY ENGINEERING DEPARTMENT. HTK Speech Recognition Toolkit. http://htk.eng.cam.ac.uk/

[6] Song, Dawn Xiaodong, David A. Wagner, and Xuqing Tian. "Timing analysis of keystrokes and timing attacks on ssh." USENIX Security Symposium. Vol. 2001. 2001.

[7] Zhuang, Li, Feng Zhou, and J. Doug Tygar. "Keyboard acoustic emanations revisited." ACM Transactions on Information and System Security (TISSEC) 13.1 (2009): 3.