EavesDroid: Keystroke Recovery using Smartphone Accelerometers

Jennifer Guo, Yi-Hsien Lin, Akshay Mittal and Wathsala Vithanage
{jjguo,yihsien,akshay,wathsala}@princeton.edu

Department of Computer Science
Princeton University

***Abstract*—In this paper we demonstrate the usage of a smartphone accelerometer to eavesdrop on a nearby computer user and recover text based on recorded keyboard vibrations. We present EavesDroid, a proof-of-concept application that can be used to record and process accelerometer data and leverage it to reconstruct the original typed text. EavesDroid uses AdaBoost with Decision Stumps and achieves up to 85% accuracy in recovering original words with an expected error rate of 2 letters per word. We also make our dataset of accelerometer recordings available to the public, the first such dataset to the best of our knowledge.**

I. Introduction

Mobile phones are becoming increasingly powerful devices. In addition to being able to run applications such as email clients and web browsers, by equipping a wide variety of sophisticated sensors, these smartphones can actively interface with the world around them.

Unfortunately, the array of sensors in today's smartphones can also be used in malicious ways. For example, malware could potentially gain access to a mobile phone's camera to take photos or video without the notice of the owner [1]. Other possible attempts are activating the device's microphone to record conversations or turn on the GPS to track the target's position [2], [3], [4], [5]. In light of the possible privacy threats mentioned above, operating systems that are implemented in modern mobile phones usually provide mechanisms so that applications must acquire the user's explicit permission before it can access the device's sensors.

However, not all sensors' accesses are tightly regulated by this mechanism, which leads to potential sensor abuse for eavesdropping purposes by malicious apps. One such example is the accelerometer, a sensor that uses about 10 times less power compared to other motion sensors [6]. Access to accelerometers is usually unrestricted on most of the current mobile phone operating systems, due to the frequent use of accelerometer data to determine and adjust window orientation.

In this paper, we present **EavesDroid**, a proof-of-concept application to recover keystrokes using accelerometer data. We demonstrate that EavesDroid is able to record and reconstruct the key presses made on a nearby keyboard with decent accuracy, based solely on the observed vibrations. We develop profiles for different keys by extracting features from the key press signals and classify them using boosted Decision Stumps. This allows us to establish an abstract representation of the relationship between a keystroke signal and its corresponding key. We then recover the typed content by translating from our intermediary form to English words using a number of different dictionaries. While there have been similar efforts to reconstruct keystrokes from recorded audio signals with high accuracy [7], reconstruction from accelerator data has not seen that much exploration. It is more challenging because of the lower sampling rate and because the data points are recorded at uneven time intervals.

By developing **EavesDroid**, we have made the following contributions to the smartphone security domain:

1) **Develop an infrastructure for characterizing keystroke vibrations**: We capture, analyze and develop profiles of key press events on a nearby keyboard based on the vibrations created when they are pressed. Features such as mean, skewness and FFT are first extracted from our input signals. We then use a set boosted Decision Stumps to create an intermediary representation, which is combined with candidate dictionaries to successfully recover words at rates as high as 70%.
2) **Dataset made publicly available**[1]: To the best

[1] https://github.com/naturegirl/EavesDroid

of our knowledge, no public dataset is available that provides signals of individual key presses as recorded by the smartphone accelerometer placed next to the keyboard. In this paper, we present our pipeline for recording data points for keystroke vibrations and we make the dataset of 1000 recorded data points publicly available to enhance further research in this area. Given the widespread attention to privacy concerns and security breaches caused by eavesdropping on device emanations, we perceive this to be of great importance in motivating smartphone vendors to restrict the usage of accelerometers to permitted applications only.

II. Related Work

Researchers have studied malicious code on mobile devices that learn information about the device owner using other embedded sensors. In the most closely related work to our paper, Marquardt *et. al* [8] propose *(sp)iPhone* which uses motion sensors in the iPhone 4 to infer keystrokes of nearby laptop users. *(sp)iPhone* uses 3 labels for each pair of keystrokes in order to determine the correct identification for the letters of the word. The location of the key is modeled using a neural network trained on 150 keystrokes for each letter of the English alphabet. A profile is then constructed for each pair using the neural network and determines and models the distance between the two events. The predictions of the neural network are matched against a dictionary to determine the top matches. The accuracy achieved is roughly 80% but this decreases to 40% as the size of the dictionary increases.

In a very similar, but orthogonal, use case, Owusu *et. al* [9] present the threat of applications which extract the keystrokes of the user's typing on the smartphone screen itself. Again, due to no restrictions on the usage of the accelerometer, they are able to extract 6-character passwords in as few as 4.5 trials(median). TouchLogger [10] uses orientation of the smartphone device to infer the keystrokes.

III. Threat Model

Our attack is based upon the following two observations. Different from most of the sensors on a smartphone such as microphone, camera and GPS, the access of the accelerometer is not protected by the permission mechanism mentioned earlier. That is to say, any application is able to access the accelerometer feed if they wish to. However, due to the frequent usage needs of the accelerometer to determine window orientation and update screen orientation, we envision that this problem will not be fixed in the near future. Therefore, the number of smartphones that are vulnerable to this threat will only increase in the near future.

Our second observation is that smartphones are often placed very closely to a laptop or the keyboard of a computer. This is mainly because smartphone users tend to put their phone within their reaching area while working so they can monitor the phone should any text or notification pop up. This user behavior enables the accelerometer within the smartphone to record keystroke signals that are transmitted from keyboard to smartphone through the underlying table surface. This clearly exacerbates the threat that accelerometer data is not protected since malicious applications can exploit the signals to reconstruct typed keys.

IV. Framework Description

A. Dataset Capturing

Our experimental setup is shown in Figure 1. We placed a Dell USB keyboard and a LG Nexus 5 smartphone on a wooden table. No other items were placed on or were touching the table. All keys were pressed with the index finger of the right hand. The hands do not touch the keyboard or table, and the fingers do not pause on the keyboard but only touch it during the brief duration of the key press. While this method of typing might seem arbitrary, we strongly believe that a unified dataset should be created first before more advanced typing patters are explored.

We recorded 1000 data points in 40 sessions with 25 letters in each sessions. For each session we generate a sequence of 25 random letters and display the letters in 3 second intervals to ensure an even recording of the letters and to ensure that there is no bias toward a specific letter. For the data capturing we developed an Android app that reads from the hardware accelerometer and syncs with our laptops, where we further process the data. The raw accelerometer data as returned by Android contains the x, y and z-direction components of the acceleration of the vibration received. We calculate the G-force magnitude as $\sqrt{x^2 + y^2 + z^2} - g$ where $g = 9.81 m/s^2$ is the acceleration due to gravity. The accelerometer records at the granularity of microseconds.

Figure 2, 3 and 4 show the G-Force values varying with time for the letters 'a', 'b' and 'p' respectively. It is clear from the plots that there is a significant difference between the features of the signal for the different letters and we aim to exploit this difference to recover the typed

letters by a user. Note that the original recordings (not shown in the interest of space) contain a large spike at the beginning and end due to the pressing of "Start" and "Stop" buttons on the recording app. Therefore, we wrote a module, Clipper, to cut off these noise signals and further segment the data into 25 individual data points. We will elaborate on this in Section VI-A.

The recorded dataset is made publicly available at the project website of EavesDroid at `https://github.com/naturegirl/EavesDroid`.

B. Feature Extraction

We extract features from the recorded signals to use in the subsequent training and testing steps of our framework. We employ a combination of time-domain and frequency-domain features as mentioned in [8] to construct our feature vector. The time-domain features we calculated include mean, root mean square (rms), skewness, variance, and kurtosis. As for the frequency-domain feature, we use 30 coefficients extracted from a Fast Fourier Transform (FFT) in order to represent the frequency spectrum of a signal. At the end of this step, we output a 40 dimensional feature vector for each single letter recording.

C. L/R, U/D and Triads

We furthermore assign two different labels to each letter, namely *Left/Right* (L/R) and *Up/Down* (U/D) respectively. The *Left/Right* label indicates whether the received signal belongs to a key that is on the left or right half of the keyboard. Keys that are at the left of the keys (including) *T, G, B* are labeled as *Left* and *Right* otherwise. As for *Up/Down* labels, it indicates whether a signal is associated with a key that belongs to the upper region of the keyboard or the lower region. Keys that belongs to the row of Q-P are labeled as *Up* and *Down* otherwise. The *Left/Right* assignment is based upon the observation that keys closer to the smartphone will have a stronger and faster propagated vibration, while the signals further on the right will have a weaker response. The *Up/Down* assignment follows a similar logic: since the keyboard is slightly tilted with the upper keys further away from the surface, the upper rows will have a stronger vibration than the lower rows.

In addition to L/R, U/D labels, we also tried to group different keys together in order to lower the classification space. We made the assumption that keys which are close to each other generate similar vibration signals. Therefore, keys were grouped into triads (with one in pairs) based on their location and labeled with their corresponding group number. Our triad assignment is as follows: {qwa}-1, {szx}-2, {erd}-3, {fcv}-4, {tyg}-5, {hbn}-6, {uij}-7, {pol}-8, {km}-9.

V. Analysis & Implementation

We present the implementation details of the algorithms used to generate a prediction model on the training. The given dataset is broken into training (66%) and testing (33%) sets. We then generate the prediction model by 10-fold cross-validation on the training set. In [8], Marquardt *et. al* use a neural network to build the signal profiles of the letters typed by the user. Our analysis builds upon their paper, but compares the accuracy of multiple algorithms. We use AdaBoost with Random Forests as the weak learner, AdaBoost with Decision Stumps as the weak learner, and Neural networks to determine the accuracies of the respective models on the testing set (shown in Table I). The results corresponding to Random Forests are using 10 decision trees. The accuracy of using Decision Stump is better than that of Random Forests because of over-fitting that takes place in the Random Forests. When the number of trees in the Random Forests was increased, the training error decreases and the test error increases. Since Random Forests represent a multitude of decision trees, the higher number of trees increases the testing error. On the other hand, Decision Stumps are one node trees and are less prone to over-fitting on the training data and provide the similar accuracies as Random Forests with less complex hypotheses/model.

Neural Network achieves almost similar accuracies as the AdaBoost implementations, but we do not use this as our choice of algorithm for the experimentations (Section VII). This is because firstly we do not have the MFCCs features extracted from the keystroke signal and these which provides better representation of cepstral information about the signal and capture by the neural network. Secondly, Marquardt *et. al* have presented a solution to the threat model using neural networks and in order to benefit the research community, we attempt to provide an orthogonal analysis with less features and using a different machine learning approach. Thirdly, by the principle of Occam's razor, we prefer to use the classifier which uses a simple hypothesis and does not overfit the data.

The cepstral features corresponding to the FFTs have significant impact on the predictions of the model. Table II shows the test accuracies using AdaBoost with Decision Stumps. FFTs alone provide 10% higher correct predictions compared to the rest of the features in the

case of L/R dataset. The combined prediction accuracy when using all the features is better than when using just the FFTs and this demonstrates that the frequency of the vibration signal received from the different keystrokes has significant impact in making correct alphabet predictions.

In addition to L/R and U/D datasets, we compute the predictions of the algorithms on the Triads dataset. Table I shows that the models generated make poor predictions about the triad to which a keystroke belongs. This is, firstly, due to the assumption that the alphabets grouped into the same triad do not necessarily have similar keystroke vibrations. Careful triads construction is expected to give better predictions but this analysis is beyond the scope of this project study. Therefore, for the purpose of experimentation, we use L/R and U/D label predictions only for determining the keystrokes of the user. Additionally, we built a neural network on the alphabet labeled dataset and achieved 41.34% accuracy for the 26 letter labels. This is significantly better than random classification but is not sufficient for word recovery from the typed key strokes.

VI. Framework Work Flow

In this section, we elaborate the flow of data and features leading to the predictions made by the model.

A. Learning Phase

In the learning phase, as shown in Figure 6, we use the 1000 data points collected (Section IV-A) for training the model. Each session of the collected data results in one raw accelerometer readings' file. Since the readings have noise peaks at the start and end of the signal, it is passed through the Clipper which strips out the ends. The clipped file is then processed by our SignalBreaker module to get the raw accelerometer data for the letters that constitute the session. The segregated letter files are used to generate two labeled datasets - L/R labels and U/D labels - for the letters. This allows for generating two trained models in the learning. The labeled letters are passed through the Feature Extractor module (Section IV-B) to the get the labeled feature vectors. This is followed by the supervised learning of the AdaBoost classifier which Decision Stumps as the weak learners on the labeled feature vectors and giving the two models for L/R classifier and the U/D classifier.

B. Attack Phase

In the attack phase, as shown in Figure 6, EavesDroid receives the raw accelerometer data for a word typed in by the user on a nearby keyboard. The clipping of the signal and signal breaking works in the same manner as in the learning phase except there is no labeling in the attack phase. The unlabeled features obtained from the Feature Extractor module are passed to the classifiers (obtained from the learning phase) and the L/R and U/D label predictions are obtained. The attack module also takes in L/R and U/D labeled dictionary words from the 72 Harvard sentences. The feature matching module, in the attack module, gives the top closest matching words corresponding to the raw accelerometer data received by EavesDroid.

The original word signal corresponding to the typed word "juice" is as shown in Figure 5. We see distinct signals for the 5 letters of the word and the noise for the "start" and "stop" is clipped from the ends by the Clipper module. Figure 7 shows the attack phase for recovering the word typed word "juice" from the signal in Figure 5.

VII. Experimentation

We extensively use the R Project [11] tool for data processing and feature extraction and for building the prediction models. R Project provides access to the RWeka [12] library which has implementations of most machine learning algorithms. The AdaBoost and Neural Networks implementations are leveraged from the RWeka package for the purpose of the experiments.

We demonstrate the accuracies of our prediction model on the Harvard sentences [13]. We chose this list of sentences because it is phonetically-balanced, *i.e.* it uses specific English phonemes with a uniform distribution. The words, when passed through the SignalBreaker, show distinct peaks corresponding to each key press and the duration of a key press is approximately 100ms [8]. SignalBreaker module achieves an accuracy of 100% in determining the peaks and constructing the letter signals from the corresponding word signal. To test the accuracies of the L/R and U/D classifiers, the letter signals for an unknown word signal are fed to the classifiers. The L/R and U/D predictions from the classifiers are matched against the dictionary and the closest matching words are predicted as possibilities. The automation of this process, from typing of a word (from a sentence) and matching the occurrence of the typed word in the predicted words, is a non-trivial task and requires an extensive time beyond the scope of this project study.

We provide a workaround to this by breaking the collected dataset of letters into 66% training set TR and the remaining 33% as the test set TS. The recorded alphabet signals in TS (Section IV-A) are unseen by the classifiers. For any sequence of words, our module

SignalGen generates a signal for that word from the set TS of the recorded alphabet signals. This is done by randomly picking a signal corresponding to each letter of the word and concatenating them together to form the word signal. For example, for the word "juice", the signals corresponding to 'j', 'u', 'i', 'c', 'e' are chosen randomly from the corresponding unseen signals for those letters. Thus we can now automate the process of testing the classifiers by feeding large articles to SignalGen and generating predictions for the corresponding words.

We conduct two experiments in order to evaluate the performance of EavesDroid. In the first experiment, we use the 72 Harvard sentences (constituting 4490 words) as the user's typed data which we want to recover. Using the SignalGen module, we construct raw accelerometer data from the TS letters data for the 72 sentences. Using the work flow of the attack phase (as explained in Section VI-B), we get the dictionary words with the least error for all candidate words in those sentences. We notice that EavesDroid is able to correctly recover 85.67% of the 4490 words by making at most 2 letter mispredictions. Table III shows the percentage of words recovered with at most the corresponding expected number of mis-predicted letters. Note that 97.27% of the words have at most 3 mis-predicted letters using the EavesDroid's classifiers. 3 mis-predictions indicates that for a 6 letter word, 12 labels are predicted and out of those at most 6 are incorrect. If we list the top word predictions for a given candidate word, using English context information, a reader can very well guess the word from the available choices. For example, Figure 8 shows the possible word choices for most of the words of the sentence, but using the context information, a human reader can reconstruct the sentence.

In the second experiment, we fed a New York Times article [14] with 766 words to EavesDroid. The articles has 395 unique words with 138 words that occur in the dictionary. The module is able to successful recover 121 out of the 138 dictionary words with at most 2.5 expected number of letter mispredictions. This demonstrates the practical utility of EavesDroid in recovering text a from victim's typing patterns.

VIII. Challenges and Future Work

Despite achieving good text recovery accuracy for practical use, we faced many challenges in this project which we aim to resolve in the future. For one, the accelerometer is extremely sensitive to surrounding noises. In addition, advanced signal filtering techniques can be used to profile consecutive key presses into the model. Clustering on the letters' signals and labeling them based on the frequency of each letter's average appearance might provide better accuracy. Also, building the model on a word's signal instead of the letter's signal might capture word signatures in a better way.

Furthermore, when using the dictionary to generate candidate words for an input word signal, the number of candidate words increases with the dictionary size. This leads to additional work in contextual identification of the correct word. To mitigate the effect of increased dictionary size, we could group the similarly vibrating keys into triads. This serves to reduce the search space. Another direction for future work is to use Hidden Markov Models (HMMs) to model contextual information of the text in order to more accurately predict words.

IX. Conclusion

Today's smartphones contain an array of powerful sensors which can be used to build increasingly interactive devices, but also to offer potential exploitation by malicious applications. The accelerometer is one such sensor with currently unrestricted access for developers, which we deem particularly problematic due to the security and privacy risks this poses. One particular threat involves using a smartphone accelerometer to eavesdrop on a nearby typing computer user by recording the vibrations of the keyboard and employing it to reconstruct the typed text.

We present EavesDroid, a proof-of-concept application to recover keystrokes using accelerometer data. We demonstrate that EavesDroid is able to record and reconstruct words typed on a nearby keyboard with up to 85% accuracy with an expected error rate of 2 letters per word. While previous approaches used only Neural Networks, we evaluated the performance of 3 different models (Neural Networks, AdaBoost with Random Forests and AdaBoost with Decision Stumps) and conclude that AdaBoost with Decision Stumps yields the highest accuracy.

Our contributions in developing EavesDroid include an full pipeline for recording and processing keystroke vibrations. We believe that this framework will be helpful for future research, as the modular design allows parts to be easily exchanged and improved. Furthermore, we provide a dataset of 1000 recorded keystroke vibrations, the first such dataset to the best of our knowledge.

The smartphone security domain is a continuously evolving area and we hope that this paper helped raise awareness of the potential misuses of accelerometer data and the need to carefully calibrate sensor access.

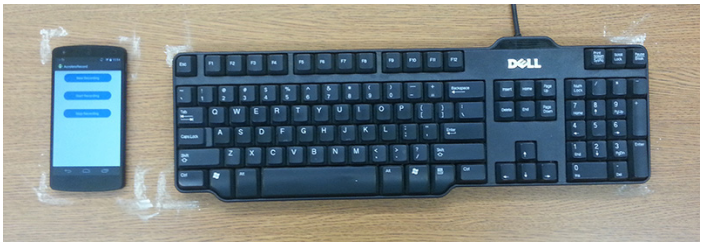

Fig. 1: Threat model of a smartphone placed next to a keyboard. Used in our experimental setup to capture the dataset.

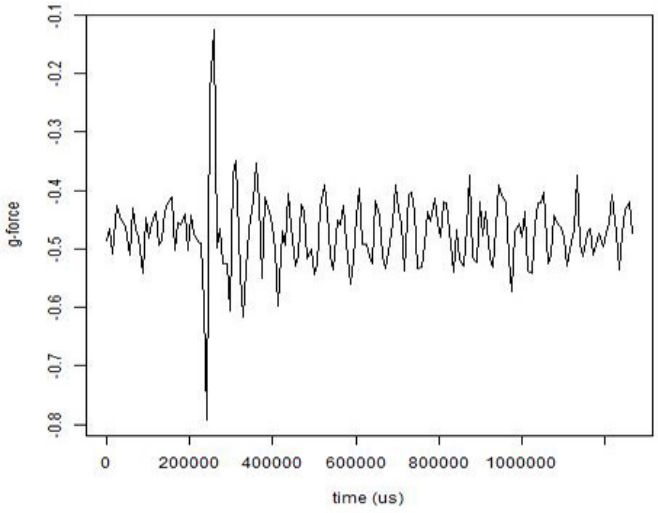

Fig. 2: Sample Recording for letter 'a' showing one distinct peak.

Dataset	Algorithm	Test Accuracy(%)
L/R	AdaBoost (RandomForests)	68.67
	AdaBoost (DecisionStumps)	69.82
	Neural Networks	68.10
U/D	AdaBoost (RandomForests)	56.60
	AdaBoost (DecisionStumps)	58.68
	Neural Networks	58.62
Triads	AdaBoost (RandomForests)	16.37
	AdaBoost (DecisionStumps)	13.21
	Neural Networks	14.65

TABLE I: Test accuracies of different machine learning algorithms on the test dataset.

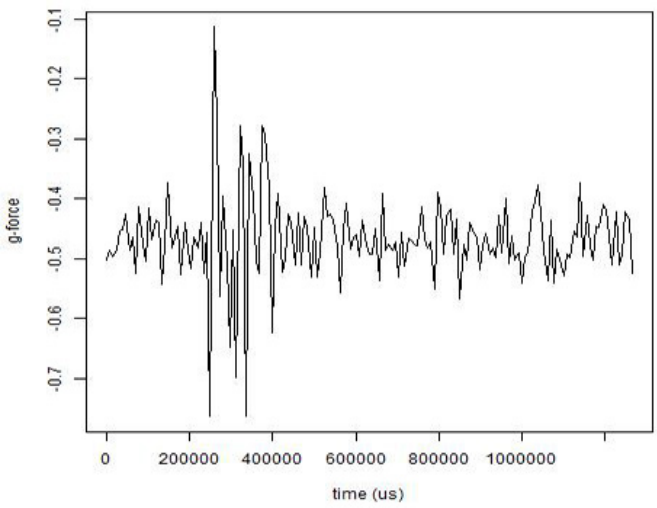

Fig. 3: Sample Recording for letter 'b' showing three distinct peaks.

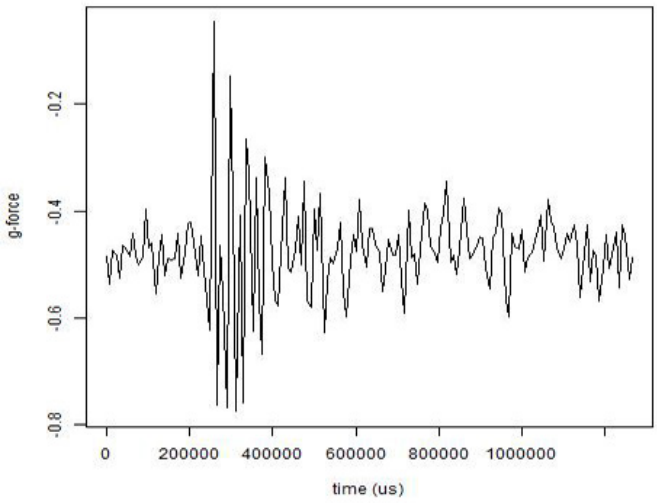

Fig. 4: Sample Recording for letter 'p' showing four closely placed peaks.

REFERENCES

[1] Zhu Cheng. Mobile malware: Threats and prevention. *McAfee Avert*, 2007.

[2] David Dagon, Tom Martin, and Thad Starner. Mobile phones as computing devices: The viruses are coming! *Pervasive Computing, IEEE*, 2004.

[3] Liang Cai, Sridhar Machiraju, and Hao Chen. Defending against sensor-sniffing attacks on mobile phones. In *Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*, pages 31–36. ACM, 2009.

[4] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *OSDI*, 2010.

[5] Manuel Egele, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. Pios: Detecting privacy leaks in ios applications. In *NDSS*, 2011.

[6] Using the Accelerometer. http://developer.android.com/guide/topics/sensors/sensors_motion.html#sensors-motion-accel.

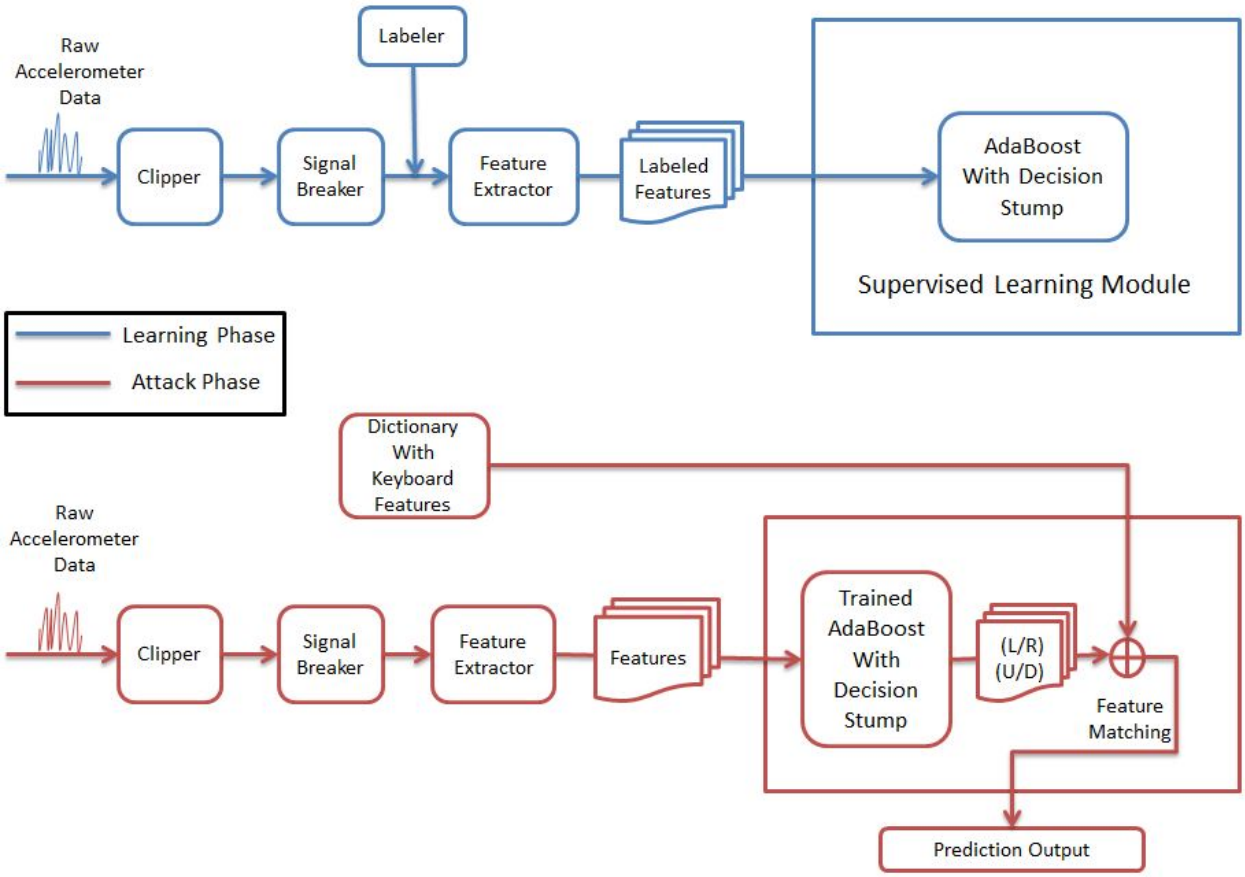

Fig. 6: The Data Processing Architecture. This diagram provides a high level overview of the architecture used for training the classifiers and for analyzing keyboard input during an attack.

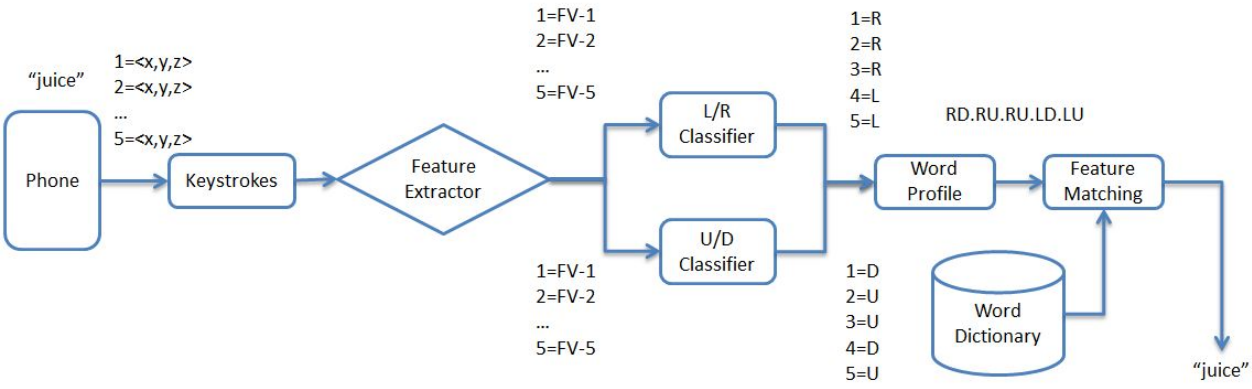

Fig. 7: Attack Phase Example. Using the word signal from Fig. 5, it runs the clipper to remove noise from start and end, then extracts the individual letter signals, followed by feature extraction. These are fed to the prediction model and using the dictionary, the word is found.

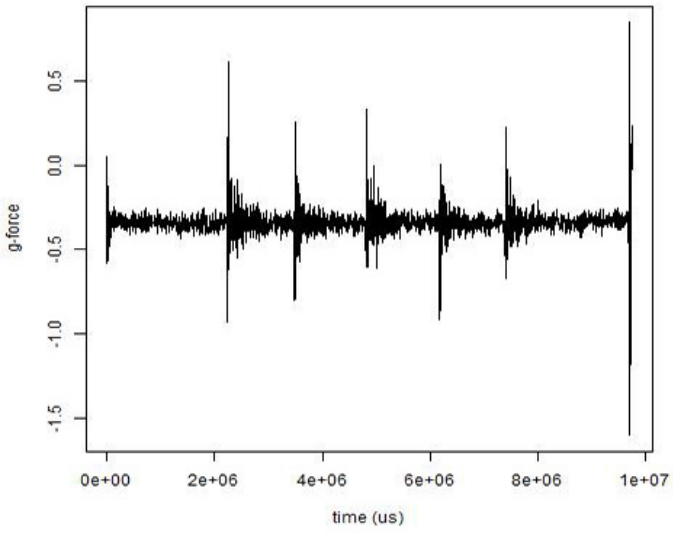

Fig. 5: Sample Recording for the word "juice"

```
    Typed Text: Glue the sheet to the dark blue background.
Recovered Text: Glue the sheet to the dark blue background.
                blue     juice        days glue
                corn     depth        work size
                hard     canoe        lack four
                 .         .           .    .
                 .         .           .    .
                 .         .           .    .
```

Fig. 8: Word options for an Harvard sentence. A word with no options represents the exactly matching word predictions.

Dataset	Features Picked	Test Accuracy(%)
L/R	FFTs	69.54
	non-FFTs	59.48
U/D	FFTs	56.89
	non-FFTs	59.48
Triads	FFTs	19.54
	non-FFTs	15.22

TABLE II: Test accuracies with and without FFT features using AdaBoost with RandomForests.

Expected # of Letter Mis-predictions	Percentage of Recovered Words(%)
0	5.46
0.5	23.41
1	41.64
1.5	70.31
2	85.67
2.5	94.54
3	97.27

TABLE III: Percentage of words recovered with at most the given number of errors in predictions

[7] Li Zhuang, Feng Zhou, and J Doug Tygar. Keyboard acoustic emanations revisited. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 373–382. ACM, 2005.

[8] 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*. ACM, 2011.

[9] Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, and Joy Zhang. Accessory: password inference using accelerometers on smartphones. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*. ACM, 2012.

[10] 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*. USENIX Association, 2011.

[11] R Project. http://www.r-project.org.

[12] RWeka package. http://cran.r-project.org/web/packages/RWeka/index.html.

[13] I. S. on Subjective Measurements. IEEE Recommended Practices for Speech Quality Measurements. IEEE Transactions on Audio and Electroacoustics, 1969.

[14] Protect Our Bats. http://www.nytimes.com/2014/05/12/opinion/protect-our-bats.html, 2014.