EavesDroid: Keystroke Recovery using Smartphone Accelerometers

Jennifer Guo, Yi-Hsien Lin, Akshay Mittal and Wathsala Vithanage
{}

Computer Science Department,
Princeton University

***Abstract*—[to - fill] - smartphones, sensors blabla**

In this paper, we demonstrate accelerometer recording eavesdropping blabla

- make dataset available public, of own recordings

- Results: 70% accuracy, with distance 3 or less

I. Introduction

Mobile phones are becoming increasingly powerful devices. In addition being able to run simple 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 these devices 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 [**?**]. Other possible attempts are activating the device's microphone to record converstaions or turn on the GPS to track the target's position [**?**], [**?**], [**?**], [**?**]. 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. 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. Furthermore, the accelerometer is very energy efficient and uses about 10 times less power than other motion sensors [**?**]. [**citation**]

In this paper, we present **EavesDroid**, an application to recover keystrokes using accelerometer data. We demonstrate that **EavesDroid** is able to record and reconstruct the keypresses made on a nearby keyboard with decent accuracy, based solely on the observed vibrations. We develop profiles for different keys by extracing features from the keypress signals and classify them using boosted Random Forests. 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 effors to reconstruct keystrokes from recorded audio signals with high accuracy [**cite**], 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 keypress 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 Random Forests 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**: To the best of our knowledge, no public dataset is available that provides signals of individual keypresses as recorded by the smartphone accelerometer

placed next to the keyboard. In this paper, we present our pipeline for recording datapoints for keystroke vibrations and we make the dataset of 1000 recorded datapoints 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.

The remainder of the paper is structured as follows: We present related work in recovering keystrokes from keyboard vibrations. We then describe our threat model and our constructed framework. Section V and VI include analysis and implementation, and the framework work flow. We show experimental results in section VII and conclude in section VIII.

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* [**?**] 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 modelled 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* [**?**] 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 [**?**] uses orientation of the smartphone device to infer the keystrokes.

III. Threat Model

Our attack is based upon 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 smartphone 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 tends to put their phone within their reaching area while working so they can monitor the phone should any text or notifications pops up. Therefore, this behaviour enables the accelerometer within the smartphone to record clear keystroke signals before they diminish and get corrupted. This clearly exacerbates the threat that accelerometer data is not protected since malicious applications such as **EavesDroid** would have access to more distinguishable signals

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.

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 $g = 9.81m/s^2$ 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 significant different 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.

B. Feature Extraction

Before actually classifying the signals, we first need to extract the features that can best represent the attributes of the signal. We used a combination of time-domain and frequency-domain features that was mentioned in [**?**] to construct our feature vector. The time-domain features we calculated includes mean, root mean square (rms), skewness, variance, and kurtosis. As for the frequency-domain feature, we calculated 30 FFT feature in order to represent the frequency attribute of a signal. At the end of this step, we have acqured 40 feature vectors, for each corresponding english letters.

C. L/R, U/D and Triads

For each english letters, we assign it with two different labels which are *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 key (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.

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 that are close to each other generates similar vibration signals. Therefore, keys were grouped into triads (with one in pairs) based on their location and labeled with their corresponding group number. Following is the group that each keys belongs to: {qwa}-1, {szx}-2, {erd}-3, {fcv}-4, {tyg}-5, {hbn}-6, {uij}-7, {pol}-8, {km}-9.

V. Analysis & Implementation

In [**?**], Marquardt *et. al* use a neural network to build the signal profiles of the letters typed by the user. We present the implementation details of the algorithms used in order to best 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. 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 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 overfitting on the training data and provide the similar accuracies as Random Forests with less complex hypotheses/model. Neural Networks 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.

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 LR 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 key-

strokes has significant impact in making correct alphabet predictions.

In addition to LR and UD 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 and we leave it as future work in the interest of limited time for this project. Therefore, for the purpose of experimentation, we use LR and UD label predictions only for determining the keystrokes of the user. Section VII will demonstrate that good triad predictions, in addition to LR and UD predictions, will increase the accuracy with which the words typed by the user are identified.

VI. Framework Work Flow

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

A. Learning Phase

In the learning phase (Figure-REF), 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 the SignalBreaker 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 UD classifier.

B. Attack Phase

In the attack phase (Figure-REF), 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 UD label predictions are obtained. The attack module also takes in LR and UD 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.

VII. Experimentation

We extensively use the R Project [**?**] tool for collecting the cepstral features from the data and building the prediction models. R Project provides access to the RWeka [**?**] 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 [**?**]. We chose this list of sentences because it is phonetically-balanced, *i.e.* it uses specific English phonemes with a uniform distribution. For testing the accuracy of text recovery when a user types on the near-by keyboard, we wrote a module, SignalBreaker, to break the vibration signal received corresponding to each word into corresponding set of letters for the particular word. There is a distinct peak corresponding to each key press and the duration of a keypress is approximately 100ms [**?**]. SignalBreaker uses this information to create segments. It 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 hack around this by exploiting the results that the SignalBreaker works with 100% accuracy and that $1/3^{rd}$ of the recorded alphabet signals (Section IV-A) are unseen by the classifiers. For any sequence of words, we wrote a module, SignalGen, which generates the possible signal for that word from the

remaining $1/3^{rd}$ 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 remaining $1/3^{rd}$ 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 5.46% of the 4490 words. Table III shows the percentage of words which at most the corresponding number of errors in the combined LR and UD label predictions. Note that 97.27% of the words have at most 6 mis-predictions using the EavesDroid's classifiers. 6 mis-predictions does not mean that all the label predictions on a 6 letter word are incorrect. Instead 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 [**?**] with 766 words to EavesDroid. The articles has 395 unique words with 257 words that do not occur in the dictionary. The module is able to successful recover 121 out of the 138 dictionary words with at most 5 labeling errors. This demonstrates the practical utility of EavesDroid in recovering text from victim's typing patterns.

[Box sentence: exact matches and possible choices for non-exact matches]

[Negative analysis, word watch, many potential matches when dictionary size increases]

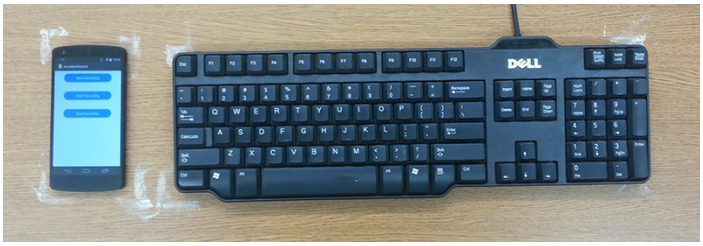

Fig. 1. Threat model of a smartphone placed next to a keyboard. Used in our experimental setup to capture the dataset.

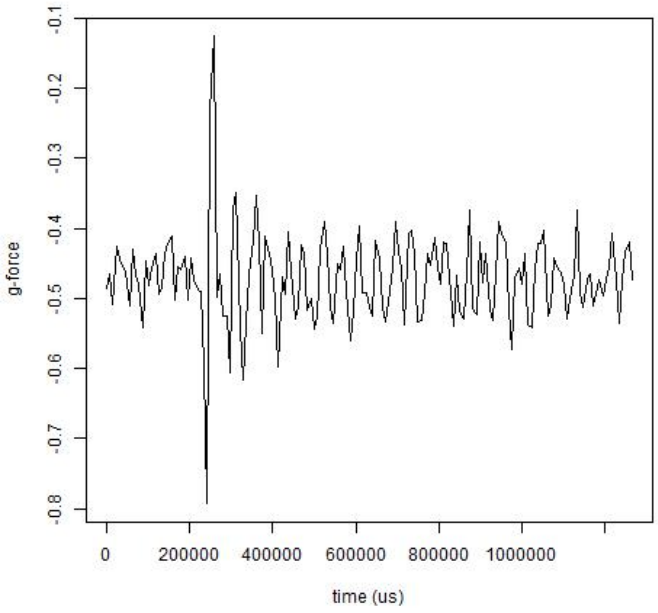

Fig. 2. Sample Recording for letter 'a'

VIII. Conclusions

A. Future Work

Mobile browsers having allowing access to accelerometer data,

http://blogs.adobe.com/cantrell/archives/2012/03/accessing-the-accelerometer-and-gyroscope-in-javascript.html

http://isthisanearthquake.com/

http://isthisanearthquake.com/seismograph.html

Acknowledgments

Appendix

A. Appendix Subsection

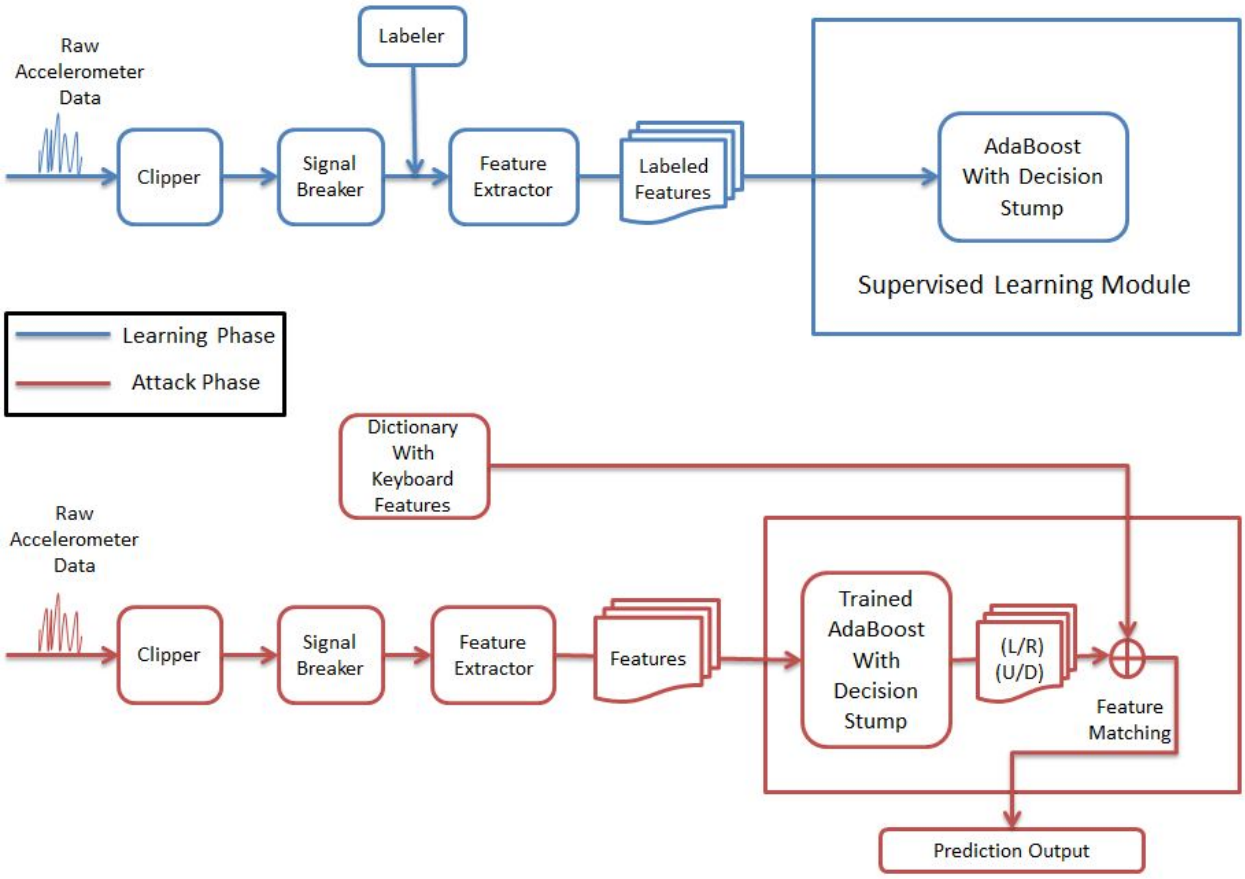

Fig. 6. The Data Processing Architecture. This diagram provides a high level overview of the architecture used for training the classifiers and for analyzing keyboard input during an attack.

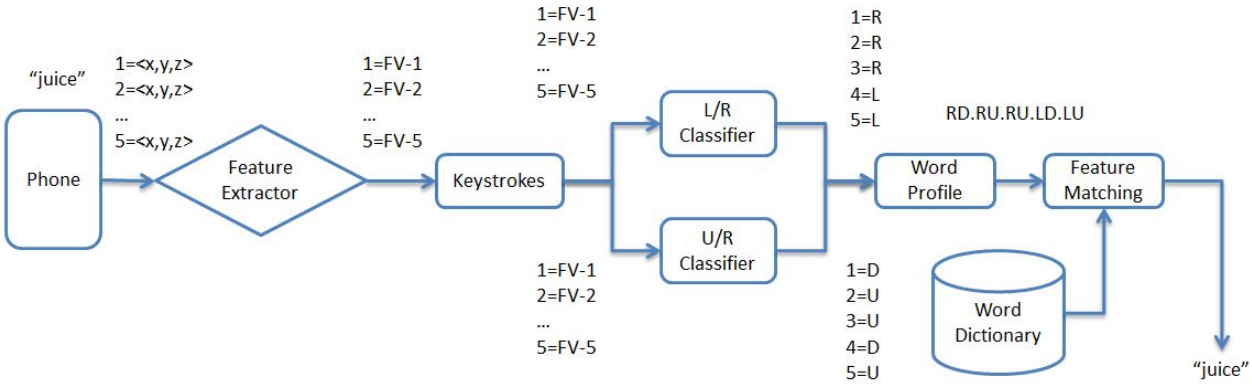

Fig. 7. The Data Processing Architecture. This diagram provides a high level overview of the architecture used for training the classifiers and for analyzing keyboard input during an attack.

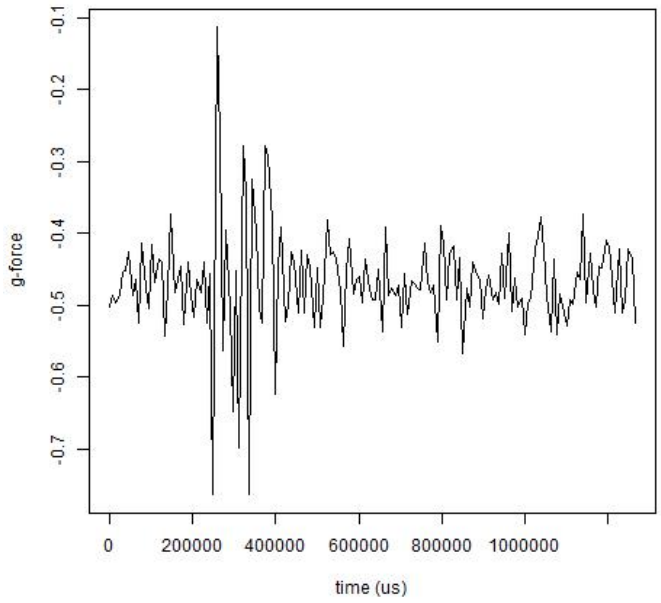

Fig. 3. Sample Recording for letter 'b'

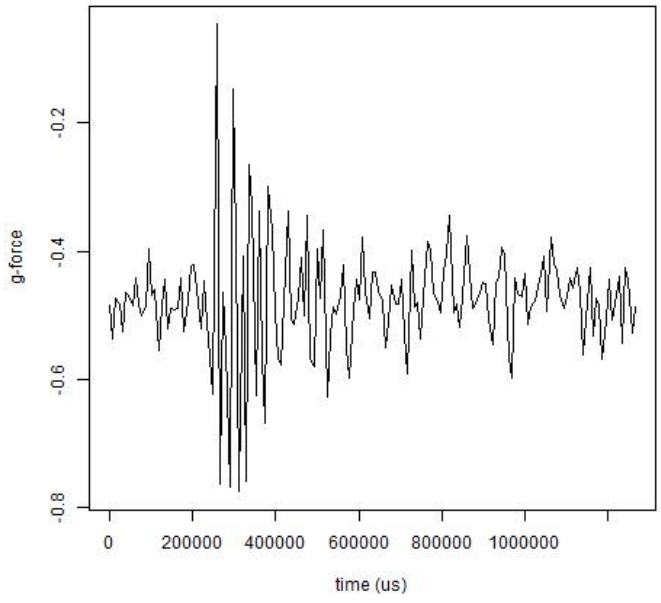

Fig. 4. Sample Recording for letter 'p'

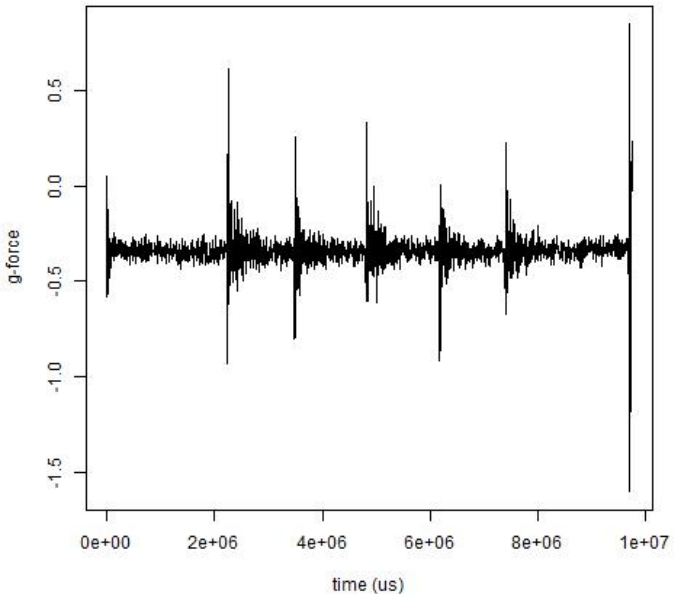

Fig. 5. Sample Recording for the word "juice"

Dataset	Algorithm	Test Accuracy(%)
3*LR	AdaBoost (RandomForests)	68.67
	AdaBoost (DecisionStumps)	69.82
	Neural Networks	68.10
3*UD	AdaBoost (RandomForests)	56.60
	AdaBoost (DecisionStumps)	58.68
	Neural Networks	58.62
3*Triads	AdaBoost (RandomForests)	16.37
	AdaBoost (DecisionStumps)	13.21
	Neural Networks	14.65

TABLE I. TEST ACCURACIES ON $1/3^{rd}$ OF DATASET

```
    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(%)
2*LR	FFTs	69.54
	non-FFTs	59.48
2*UD	FFTs	56.89
	non-FFTs	59.48
2*Triads	FFTs	19.54
	non-FFTs	15.22

TABLE II. TEST ACCURACIES WITH AND WITHOUT FFT FEATURES USING ADABOOST WITH RANDOMFORESTS.

# errors (L/R, U/D)	Recovered Words Accuracy(%)
0	5.46
1	23.41
2	41.64
3	70.31
4	85.67
5	94.54
6	97.27

TABLE III. PERCENTAGE OF WORDS RECOVERED WITH AT MOST THE GIVEN NUMBER OF ERRORS IN PREDICTIONS