Single-stroke Language-Agnostic Keylogging using Stereo-Microphones and Domain Specific Machine Learning

Sashank Narain, Amirali Sanatinia and Guevara Noubir
College of Computer and Information Science
Northeastern University, Boston, MA
{sashank, amirali, noubir}@ccs.neu.edu

ABSTRACT

Mobile phones are equipped with an increasingly large number of precise and sophisticated sensors. This raises the risk of direct and indirect privacy breaches. In this paper, we investigate the feasibility of keystroke inference when user taps on a soft keyboard are captured by the *stereoscopic* microphones on an Android smartphone. We developed algorithms for sensor-signals processing and domain specific machine learning to infer key taps using a combination of stereo-microphones and gyroscopes. We implemented and evaluated the performance of our system on two popular mobile phones and a tablet: Samsung S2, Samsung Tab 8 and HTC One. Based on our experiments, and to the best of our knowledge, our system (1) is the first to exceed 90% accuracy requiring a single attempt, (2) operates on the standard Android QWERTY and number keyboards, and (3) is language agnostic. We show that stereo-microphones are a much more effective side channel as compared to the gyroscope, however, their data can be combined to boost the accuracy of prediction. While previous studies focused on larger key sizes and repetitive attempts, we show that by focusing on the specifics of the keyboard and creating machine learning models and algorithms based on keyboard areas combined with adequate filtering, we can achieve an accuracy of 90% - 94% for much smaller key sizes in a single attempt. We also demonstrate how such attacks can be instrumentalized by a malicious application to log the keystrokes of other sensitive applications. Finally, we describe some techniques to mitigate these attacks.

Categories and Subject Descriptors

K.6.5 [**Security and Protection**]: Unauthorized access; Invasive software; K.4.1 [**Public Policy Issues**]: Privacy

Keywords

Smartphone Security; Side-Channel Attacks; Sensor Malware; Tap Detection with Motion Sensors; Keystroke Inference using Gyroscope; Keystroke Inference using Stereo-Microphones; Machine Learning; Privacy

WiSec'14, July 23–25, 2014, Oxford, United Kingdom

ACM 978-1-4503-2972-9/14/07 ...$15.00.
http://dx.doi.org/10.1145/2627393.2627417.

1. INTRODUCTION

Mobile devices have proliferated across the globe with an estimated 1.75 billion smartphone users world-wide by the end of 2014 [9]. These smartphones are used for various day-to-day and business activities, many containing sensitive information such as Personally Identifiable Information (PII), banking and credit card numbers, passwords, health records, and location information. This means that a malicious application installed on the smartphone can siphon off sensitive information and leave millions susceptible to data theft. Operating systems on these smartphones prevent unauthorized access to application / user information by implementing security mechanisms such as sandboxing and permissions. However, side-channels such as sensors can bypass such security restrictions. Most modern smartphones contain several sensors such as microphones, cameras, gyroscopes and accelerometers that provide a better user experience but these sensors, however useful, leak information that provides an adversary opportunity to covertly infer and steal sensitive data.

In this work, we demonstrate that by recording the tap sounds and vibrations from the stereo-microphones and gyroscope of a smartphone while a target application is running, it is possible to successfully infer user typed keys with high accuracy. We also show that the audio data can be combined with the gyroscope to further boost the accuracy. Android OS is popular among users and smartphone manufacturers with more than million activations daily [15]. Owing to this popularity, we chose to perform our experiments on two popular Android smartphones and a tablet: Samsung S2, Samsung Tab 8 and HTC One. While there have been related work, to the best of our knowledge, we are the first to break the 90% accuracy barrier on the standard Android QWERTY and number keyboard. For instance, some previous work that demonstrated key inference based on Android sensors, Aviv et al. [1] reached an accuracy of 43% and 73% on PIN and pattern passwords, respectively, requiring 5 attempts, Owusu et al. [23] were able to infer 6 passwords out of 99 six-character passwords in an average of 4.5 trials on the QWERTY keyboard in landscape mode, and Miluzzo et al. [21] predicted 4x5 icon taps on iPhone and Nexus S with an accuracy of 80%. To achieve an accuracy of 90%, they relied on more than 20 repetitions. To achieve a high accuracy on the standard keyboard, we developed an algorithm

and framework based on statistical methods and machine learning that can predict keystrokes without repetition or multiple attempts. Our framework is language agnostic as we do not use any lexical properties of languages, however we do assume that the adversary knows the keyboard layout. We demonstrate the algorithm using data collected at an office and in a restaurant. A malicious application and a weak permission model for Android sensors coupled with data modeling techniques make our attack feasible and consequential. We make the following contributions:

- We show that by recording the tap sounds from the stereoscopic microphone and the tap vibrations from the gyroscope on a smartphone, it is possible to infer user's typed keys with a reasonably high accuracy of above 90%. We also show that this accuracy can be boosted by combining the audio data with the gyroscope data. We design an automated system that can process this raw keystroke data, perform noise filtering, build training models and use these models to make language agnostic keystroke predictions on unknown test data.
- We develop a specialized meta-algorithm that divides the keyboard into specific areas and trains models using audio and gyroscope data for those areas. The algorithm combines character-specific and area-specific models to make more accurate predictions. We show that by combining our algorithm with meta-algorithms such as Bagging and Boosting [8], we were able to achieve higher per algorithm accuracy than an elementary use of the machine learning algorithms on unknown test data.
- We demonstrate the feasibility of a Trojan that regularly queries the Android system for the foreground application and covertly records the microphone and gyroscope when a sensitive application is used.

The rest of the paper is organized as follows. In section 2, we describe a scenario of how a stealthy attack can be launched against a sensitive Android application by a victim inadvertently downloading a malicious application. In section 3, we describe the high-level architecture of our system and discuss why the gyroscope and microphone sensors leak information about a user's typing activity. In section 4, we describe our automated keystroke inference system, data collection process and the meta-algorithm. In section 5, we present the results of our experiments. In section 6, we describe techniques that can be used to mitigate these attacks. In section 7, we describe previous related work and in section 8, we conclude and discuss future research directions.

2. ATTACK VECTOR

To perform a successful attack, the adversary follows the set of steps described below. The adversary develops and distributes a malicious application usually as a Trojan (step 1). They trick the victim into installing the application (step 2) through techniques such as social engineering (e.g., a malicious application disguised as a game or a note taking application or through USB connection to bogus devices [2]). Previous works have found examples of such applications with backdoors in the Android marketplace [30], [32]. After compromising the victim's smartphone, the application performs two roles. First, it presents a custom keyboard to the user to collect typing behavior for training models. Second, after training, it listens in the background for keypresses from sensitive Android applications.

To successfully perform the first role, the malicious application starts collecting user's typing behavior by using a custom keyboard and recording the stereo microphones and gyroscope (step 3). The microphone requires explicit permissions during installation and the adversary needs to declare it in the *manifest* file. To avoid raising the victim's suspicions, the adversary justifies such permissions by providing functionality such as: note taking, supporting voice commands and voice recognition. The application must serve an actual purpose (e.g., a Todo app) so that the user actually uses the application. After monitoring the victim's behavior, the Trojan uploads the collected training data to a remote server to build models specific to the victim (step 4). The adversary can also use generic training data and prediction models as a trade-off for performance and stealth.

To accomplish the second role, the Trojan runs in the background and queries the smartphone OS for the current location and the application. To reduce battery drain, these queries run at predefined conservative intervals. On inferring that the victim is at a place of interest using the GPS or cellular and wireless networks (e.g., bank or residence garage door) or when the victim opens a sensitive application (e.g., a bank application), the Trojan starts to actively collect the microphone and gyroscope data (step 5). The current application can be found by using the `ActivityManager` class in Android SDK if the application has the `android.permission.GET_TASKS` permission set in its manifest file. The data is filtered and keystrokes are extracted by the application and evaluated using the training models to infer the user's typed keystrokes.

3. APPROACH

In this section, we describe the architecture of our key inference system that performs audio and gyroscope keystroke noise filtering and extraction, data consolidation for machine learning, model training, evaluation of test instances and evaluation of accuracy.

3.1 Architecture

The architecture of our keystroke inference system is composed of the components shown in Figure 1. The *Application* component is the trojan application that secretly records the gyroscope and stereo microphones when a user types in our application (training data) or another security sensitive application (test data). The raw gyroscope and audio data is uploaded to a remote server that runs the remaining components of our system to process this data, create training models and perform evaluations on the test data. This raw data contains user tap vibrations and sounds mixed with pauses and noise that occur due to external factors such as unstable hands (gyroscope noise) or background music (audio noise), see Figures 6 and 7. The *Pre-processing* component removes noise from the data using adequate filtering, extracts the user taps, and performs fitting to resample the data. The techniques used for filtering and extracting gyroscope and audio data are different and require two separate pre-processing components. Once the data has been processed, it is converted to a format that is understood by Weka [16]. During conversion, the gyroscope and audio data may also be combined for inference. This means that the two need to be synchronized with each other. The

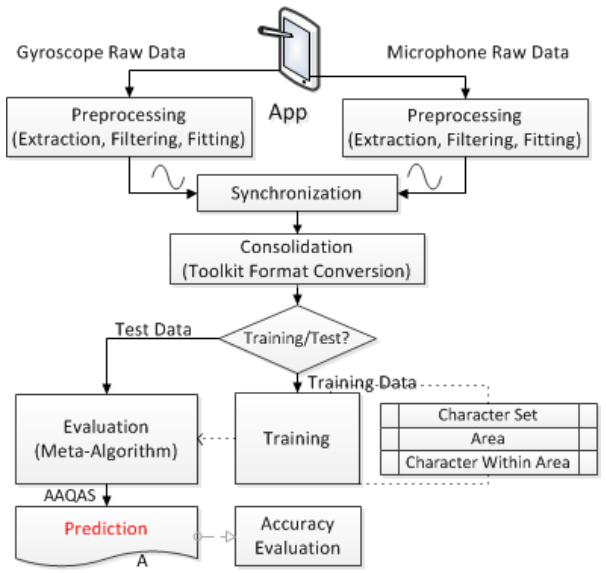

Figure 1: Architecture of the key inference system.

Synchronization component synchronizes the gyroscope and audio data and the *Consolidation* component converts the extracted and filtered inference ready data to a machine learning toolkit format. In order to classify unknown test data, the consolidated training data is used to build models. The *Training* component analyzes the data for errors, randomizes them and uses several machine learning algorithms to build different inference models for the entire character set as well as specific areas on the keyboard. These models are then used by the *Evaluation* component to perform predictions on the unknown test data. The component uses a meta-algorithm that makes a final keystroke prediction. The meta-algorithm uses a multi-step approach based on the specific layout of Android QWERTY and number keyboards to optimize the inference accuracy of test keystrokes. Once predictions have been made for all unknown test data, the *Accuracy Evaluation* component compares them with the expected keystroke to evaluate our system's performance.

3.2 Sensors

3.2.1 Gyroscope & Accelerometer

Gyroscope and Accelerometer sensors on smartphones can detect vibrations for every keystroke when a user types on a soft keyboard. Figure 2 shows the location of the gyroscope (in red), the location of keys 'I', 'Q' and 'V' on a standard QWERTY keyboard on the HTC One, and the co-ordinate system of Android sensors. The magnitude and orientation of these vibrations vary depending on the tap location with respect to the two axes which can be mapped to a standard fixed keyboard layout. In Figure 3, we see that key 'Q' shows significant vibration in the y axis and key 'V' shows significant vibration in the x axis. As the key 'I' is close to both the axes, it does not show considerable vibration in both the axes.

Gyroscope sensors are attractive targets for building smartphone keyloggers. They are easy to use in Android using the Android SDK [14] APIs defined in the `Sensor` class. They do not require special permissions and they can run in the background without prompting or notifying the user. A potential issue with using the SDK API is that the sampling rate is not fixed and may reduce when more processor intensive services are running, thereby reducing the inference accuracy. A solution for obtaining high and constant sampling rate on a Android smartphone even when other high priority services are running on the system is to use the Android NDK [13] APIs defined in `sensors.h`.

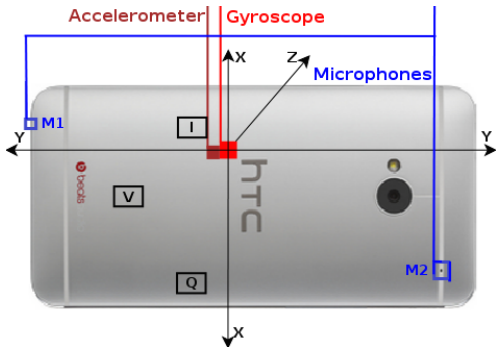

Figure 2: Location of the Accelerometer, Gyroscope and Microphones on HTC One; Approximate location of keys 'I', 'Q' and 'V' on standard QWERTY keyboard.

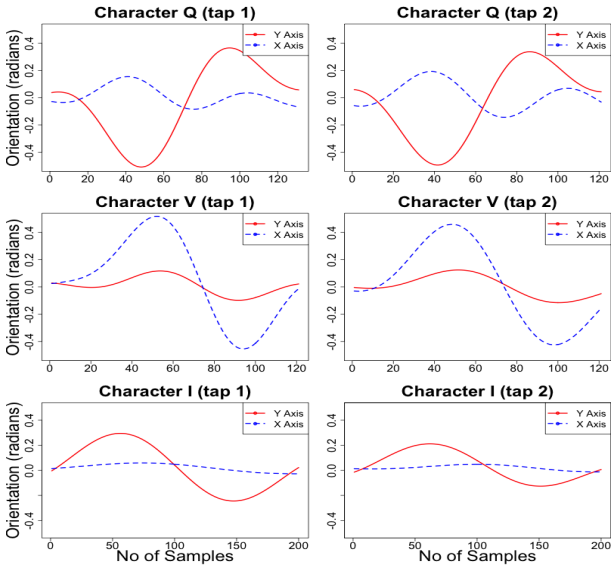

Figure 3: (Top, middle and bottom) Similarity between two keystrokes for letters 'Q', 'V' and 'I'; Each letter pattern is visually different from other letter.

3.2.2 Microphones

Microphone arrays are becoming commonplace in smartphones. Some smartphones, such as the iPhone 5s, are equipped with three microphones. Other smartphones, such as the HTC One, are equipped with dual membrane microphones that focus on capturing different sound levels (one for sensitivity and the other for distance) on a single microphone. In both these arrangements, the audio captured by the arrays are combined and processed to provide high quality and distortion free audio recording to users with the capability to detect feeble sounds. In addition, the HTC One also supports stereo recording. We observed that a soft

tap on a smartphone, by a user, is audible. This tap can be covertly recorded and processed to infer a user's typing activity with relatively higher accuracy than the gyroscope. There are two techniques to infer keystrokes, one that uses the amplitude of audio signals at the two microphones and the other that uses the time delay between signals reaching the two microphones. We use a combination of both the techniques to build our inference models. Figure 2 shows the location of the microphones (in blue) on the HTC One. Since the microphones are synchronized, for a tap at location T, the delay in tap detection between the two microphones M1 and M2 can be computed using the following formula.

$$\text{Number of Samples} = \frac{(\text{Distance(T,M1)} - \text{Distance (T,M2)}) * \text{Sampling Rate}}{\text{Speed of Sound}} \quad (1)$$

The distance between the two microphones on the HTC One is about 0.134 m. The current maximum supported sampling rate for Android is 48000 Hz and the speed of sound in air is about 340 m/s. Using these values with the formula, a difference of 18 samples is obtained for taps in close proximity to the microphones. This means that taps on different keys will produce varying sample differences based on their distance from the two microphones. The difference in samples will increase when smartphones start supporting higher sampling rates such as 192 KHz, currently supported by Blue-ray. Using a rate of 192 KHz with the formula, a difference of 75 samples can be obtained for taps in close proximity to the microphones. This will significantly improve the accuracy of inference and is indicative of the impact of the sampling rate on the accuracy.

To illustrate the time delay between the microphones, we recorded multiple samples for keys 'Q' and 'V' on a standard QWERTY keyboard on the HTC One at a sampling rate of 48KHz. Figure 4 shows a single tap for the two keys. We found that, for multiple taps, the two keys always have the same delay of 8 and 15, respectively, between M1 and M2. The figure also shows that the amplitudes at M1 were significantly lower than M2. This is because our application uses M2 as the primary microphone, and requests the Operating System to perform noise cancellation when supported. The lower amplitudes at M1 are a result of noise cancellation by the system.

Microphone functionality is easy to implement in Android using the Android SDK APIs defined in the `AudioRecord` class. Even though they initially require special permissions from the user, once installed, they can run in the background without prompting the user or showing a notification. This makes the microphones an attractive target for building Android keyloggers. Stereo recording is a relatively new concept in smartphones and we expect a lot of new smartphones to adopt this technology. We believe that this capability can be used maliciously and should be addressed.

4. SYSTEM DESIGN AND ALGORITHMS

In this section, we describe the hardware and software we developed, the data collected, the gyroscope and microphone noise filtering and extraction process, the training process and the meta-algorithm that we developed to make predictions on unknown test data.

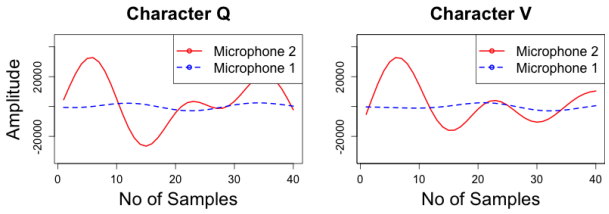

Figure 4: (Left) Sound waves received by the two microphones of HTC One for key 'Q'; (Right) Sound waves received by the two microphones for key 'V'.

4.1 Data Collection

4.1.1 *Hardware & Software*

In this work, we decided to evaluate our algorithms and system on two popular Android smartphones and a tablet: Samsung S2 (Android 4.1), Samsung Tab 8 (Android 4.1) and HTC One (Android 4.4); all three using the stock Android Operating System provided by the vendor. We did not make any modifications to the operating system and its underlying mechanisms. We developed an application to collect accelerometer, gyroscope and microphones data for training the models and for generating test samples for evaluation. It runs in two separate modes for training and test data. The data collected in the training mode is used for building the inference models and the data collected in the test mode is used for making predictions using these models. In case of the test mode, the application invokes a service that runs in the background. These two modes are completely independent of each other and do not overlap. Our application implemented a custom keyboard that has the same layout and capabilities as the standard Android QWERTY and Number keyboard. The only difference is that our keyboard has the capability to detect a key press and inject a key press event in the gyroscope data. Figure 5 shows the screenshot of the custom keyboard. The user presses the 'Start Recording' button and types data on the keyboard. Whenever they are done typing their data, they press the 'Stop Recording' button that is visible after recording is started. During this typing session, the accelerometer, gyroscope and microphones are activated in the background and their data are recorded and stored on the smartphone's internal memory.

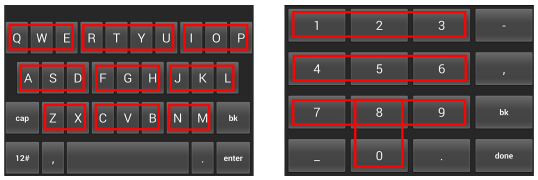

Figure 5: QWERTY and Number Keyboard with Area divisions.

4.1.2 *Gyroscope & Microphones*

We collected accelerometer, gyroscope and audio training and test samples for both the number and QWERTY keyboard in Portrait mode on the Samsung S2, the Samsung Tab 8 and the HTC One. The data was collected by seven participants who were asked to take samples in their normal typing style. We observed their typing behavior and all of

them held the device in one hand and typed using the index finger or thumb of the other. One participant (User4) held the device in their right hand and the remaining in their left hand. The main difference in typing behavior was the intensity with which the finger touched the screen and the angles at which these devices were held. These participants were asked to type anything they wanted (random characters and words) and at least 30 samples per character / number in the training mode.

The data on the HTC One was collected in two environments. Five participants typed in an office environment and two of them also typed in a restaurant. The office environment consisted of a cubicle with three computers and a server running all the time with additional noise from keyboard typing, doors opening and closing, people talking and faint noises of vehicles from on a nearby street (noise level around 49-52 dB). The restaurant was much more noisier with background music, several people talking and noise from utensils (noise level around 72-76 dB).

The data collected from the gyroscope are the smartphone's time, gyroscope accuracy, x axis orientation, y axis orientation and the z axis orientation at the maximum sampling rate for the device. We additionally collected the key press time using a custom keyboard having the same layout and capabilities as the standard Android QWERTY and Number keyboard. We chose to discard the z axis orientation as vibrations caused by keystrokes mainly affected the x and y planes.

The data collected from the microphones are the amplitudes received by the two microphones. We collected raw audio data instead of pre-processed data. A sampling rate of 48KHz was chosen because Android currently does not support higher sampling rates.

During the initial phase of the experiment, we also collected accelerometer data from the smartphone. Subsequently, we did not consider this sensor because the accuracy was significantly lower than the gyroscope. The reason was that the accelerometer combines acceleration and gravity and the gravity component varied significantly even when the phone was placed stationery on a table. Even when the gravity vibrations were removed by filtering techniques, there was not much improvement in accuracy.

4.1.3 *Synchronization*

The gyroscope and audio data are synchronized by injecting a microphone start event in the gyroscope data. The trojan application invokes two threads to start the gyroscope and microphones, however, the microphones are started only after the gyroscope has completely initialized. This is done to ensure that the gyroscope sensor starts at the highest sampling rate and does not get reduced to a lower rate due to resource consumption by the microphones. Once the microphone starts recording, the application generates a microphone start event that gets injected into the gyroscope data. The two are then recorded simultaneously.

4.2 Pre-processing

4.2.1 *Gyroscope*

Contrary to previous works [29], [21], [5] that extracted features from the gyroscope recordings, we use the raw gyroscope x and y axes orientations as the feature set for machine learning. The Android Sensor API returns the rate of change of orientation in radians/second. We use this rate to construct the gyroscope recording and then use this as our raw data. The raw data is filtered, extracted, re-sampled and converted to a machine learning toolkit compatible format. Figure 6 shows the stages of pre-processing for a gyroscope sample, these stages are described below.

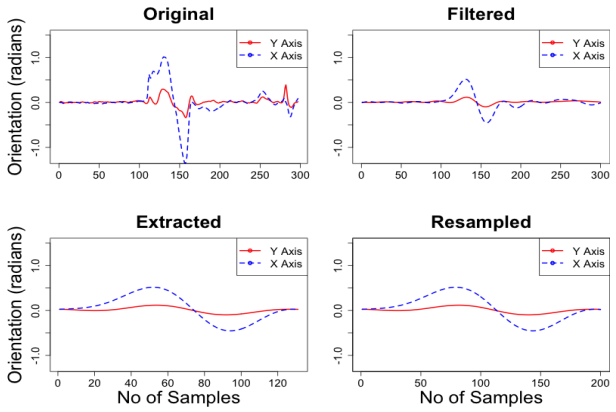

Figure 6: (Top Left) A noisy unprocessed raw gyroscope sample for letter 'V'; (Top Right) Noise removed from the sample after frequency filtering; (Bottom Left) The filtered sample after tap extraction; (Bottom Right) The sample after resampling.

Filtering: The gyroscope noise during a typing session occurs mainly because of the instability of the hand and this noise is typically high frequency. When the change in orientation is much higher than the noise, i.e. high signal-to-noise ratio, the noise can be removed by simple frequency filtering techniques. We use a Fast Fourier Transform filter [11] to detect frequencies corresponding to the sample values (the tap). We keep these frequencies unchanged and remove the amplitudes at the other frequencies. We then apply an Inverse Fast Fourier Transform to obtain the filtered data. The technique works quite well, however, for noisier data advanced filtering schemes such as Kalman Filtering [28] can be applied. When the signal-to-noise ratio is low, then these filters may remove significant tap specific information reducing the accuracy. One option is to use the unfiltered raw data in such cases but we observed that the accuracy with filtered data is much better than raw data. This is because the noise in the data changes the waveform and makes them dissimilar even for the same location. Another option is to observe the gyroscope vibrations and record only when the noise is under a threshold.

Extraction: Our trojan Android application uses a custom keyboard with the standard Android keyboard layout for tap detection. The keyboard injects an event into the gyroscope data when a key is pressed. Our system uses this as the start of the sample and uses a constant time difference to compute the end of the sample.

Resampling: The filtered gyroscope data are of different sizes as the Android SDK does not allow setting a fixed sampling rate for sensors. However, most machine learning toolkits are very specific about file formats and require fixed length training and test samples, therefore the data has to be resampled before consolidation. Another reason for resampling is to increase the size of the data such that

minute changes in data are identified by machine learning algorithms. We use a technique known as Cubic Spline Interpolation [20] to resample the data without changing the waveform of vibrations. Our evaluation results confirm that greater number of samples increase the inference accuracy.

4.2.2 *Microphones*

Instead of extracting specific properties from the audio samples and using them as features, we use the raw audio received at the two microphones. The raw audio data is first filtered to remove noise, then extracted to obtain the tap, re-sampled and converted to a machine learning toolkit compatible format. Figure 7 shows the stages of pre-processing for an audio sample, described below.

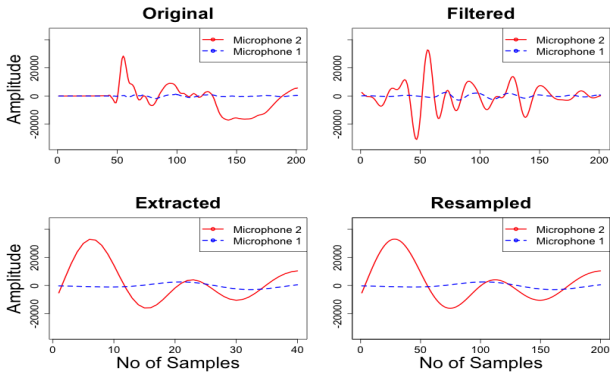

Figure 7: (Top Left) A noisy unprocessed raw audio sample for letter 'V'; (Top Right) Noise removed from the sample after frequency filtering; (Bottom Left) The filtered sample after tap extraction; (Bottom Right) The sample after resampling.

Filtering: The noise in audio signals can be present due to several factors such as background music, conversations, and moving traffic. Audio noise is typically high frequency and can be removed using frequency filtering techniques. We use a bandpass filter to pass all frequencies in the range of 1500 to 3500 Hz and stop the remaining frequencies. Bandpass filtering is an effective technique when the tap and noise frequencies don't overlap. The frequency range was obtained by analyzing recordings on the HTC One such that background noise was removed while retaining the frequencies of the tap sound. A low signal-to-noise ratio can have a significant impact on the inference accuracy and even when filters exist to remove noise, the noise removal algorithm may also change the original waveform and decrease inference. One option is to observe the microphones data and record only when the noise is under a threshold.

Extraction & Resampling: We use the peak amplitude at the two microphones to detect the start of the sample. The channel that has the higher amplitude becomes the base channel and about two wavelengths are extracted to ensure that the peak of the other channel is extracted as well. The audio samples are interpolated so that changes in data are detected as features by the machine learning algorithms.

4.3 Training Process

The system uses the specifics of Android QWERTY and number keyboard and a number of steps and algorithms to develop adequate training models.

Table 1: Accuracies of elementary algorithms for some sample sets.

Keyboard	Sensor	DT	NB	NN	10-NN
HTC One					
QWERTY	Mics	86%	85%	90%	80%
QWERTY	Comb	85%	81%	89%	84%
Number	Mics	70%	81%	72%	66%
Number	Comb	68%	73%	78%	71%
Samsung S2					
QWERTY	Gyro	60%	61%	58%	52%
Number	Gyro	74%	74%	82%	72%

Consolidation: The filtered training data are consolidated into a single database file in the machine learning toolkit format. The file can contain only the gyroscope data (for non-stereo microphones), the microphone data or the microphone + gyroscope data combined (for stereo-microphones). Unlike [21], we do not use any techniques to combine the microphone + gyroscope data but simply append their contents. We reason that these sensors have different properties that may be lost if combined.

Area Division: The keyboards were broken down into areas such that all keys in a particular area are distinguishable from another using at least one feature. The area division for a standard QWERTY keyboard in portrait mode is shown in Figure 5. They are chosen such that the x-orientation remains constant and y-orientation varies for all keys in the area. For example, for the area 'QWE', the negative y-orientation will be higher for 'Q', lesser for 'W' and the least for 'E'. The area division for a standard number keyboard in portrait mode is shown in Figure 5 and follows the same technique except for keys '8' and '0' that will have a constant y-orientation and varying x-orientation. We tested other area divisions as well but this division worked better than the others.

Elementary Algorithms: The Weka toolkit offers a large variety of machine learning algorithms. The problem of predicting the keystrokes is a supervised classification problem, therefore we eliminated algorithms that do not apply in this context (e.g., K-Means). Before developing our meta-algorithm, the algorithms we tested were Decision Trees, Naive Bayes, K-Nearest Neighbor (k-NN), Hidden Markov Models, Support Vector Machines, Random Forest and Neural Networks. Of these algorithms, Decision Trees (DT), Naive Bayes (NB), 1-Nearest Neighbor (NN) and 10-Nearest Neighbor (10-NN) performed better and yielded higher accuracy rates. Neural Networks also yielded high accuracy rates but was not chosen due to heavy resource consumption. In the context of our work, instance-based methods such as k-NN yield high accuracy, since they try to find the closest match between the new prediction and the training data. Table 1 shows the performance of these elementary algorithms for three QWERTY and three number keyboard sample sets. As we can see, none of these algorithms perform well on all areas of keyboard because of overlapping instances. This observation drove us to develop a meta-algorithm which considers the areas of keyboard before making predictions.

The Training Process: The goal of the training process is to build inference models based on the entire character set and areas defined for the keyboard.

In step 1, training models are built for predicting areas of the keyboard using a voting [18] algorithm. The voting algorithm uses the prediction of all the algorithms and their confidence values to determine the area of the test data. The model for voting uses ensembling techniques such as Bagging and Boosting to improve the accuracy of algorithms. The Ensembling technique builds multiple models from subsets of the training data, analyzes the accuracy of the subsets to detect incorrectly classified instances, and then uses these instances again with different weights or averaging to build better predictive models.

In step 2, training models are built for the entire character set for all the algorithms. These models also use ensembling. If the training data specified is microphones + gyroscope, then training models for microphones are also built. This is because the gyroscope data for certain areas of the keyboard that are close to the actual hardware may be weaker that other areas and weak gyroscope data may reduced the overall accuracy of the model for that area.

In step 3, training models are built for all character sets within an area for each area. These models also use ensembling. If the training data specified is microphones + gyroscope, then training models for microphones are also built. Our system evaluates these models using multi-fold cross validation using varying fold values. In multi-fold cross validation, a subset of the training data is provided to the model as test data and the accuracy of the model is computed. By using multiple folds, a model can be tested multiple times with different training data and their accuracies are averaged. Our system determines which models are better for an area and uses these for predictions for that area.

In step 4, the two best algorithms for an area (determined in step 3) are combined into a single voting algorithm. This voting algorithm is used by the meta-algorithm to make a final prediction in case all previous prediction mechanisms do not conclude on a single prediction. In case the two models predict different keys, their confidence values are used to determine the final prediction.

4.4 The Area-Based Meta-Algorithm

Initially, we used elementary algorithms and voting schemes to make predictions on test data. Using these elementary algorithms, the accuracy achieved was not high (See Table 1), and different algorithms predicted different keys for the same test data. To address the problem, we developed a meta-algorithm that utilizes our area specific models at multiple levels to infer the key with a higher accuracy rate than traditional algorithms. The evaluation we performed shows that the meta-algorithm yields much better accuracy.

The Inference Process: Figure 8 shows the flow diagram of our meta-algorithm and the levels of evaluations.

In step 1, the test data is evaluated using the area voting model. The goal of this step is to identify the area where a keystroke would belong to and load the appropriate models for that area. Our system evaluates the models built for every area and maintains a list of the models that have yielded high accuracy with the training data for that area. For example, an area 'IOP' on the QWERTY keyboard may have weak gyroscope data implying that the consolidated model will be weak. The system detects this and loads the microphone model for evaluation instead of the microphones + gyroscope combined model.

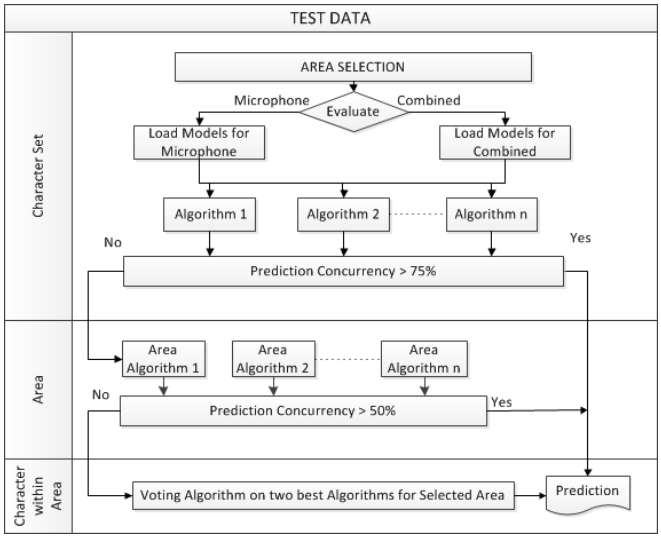

Figure 8: Flow diagram of the Meta-algorithm.

In step 2, the test data is evaluated using the loaded character set models. We use these models before the area-specific models to ensure that a prediction error by the voting model is detected and corrected. For example, the voting model may predict the area of a test sample on a QWERTY keyboard as 'IOP' when the key pressed was 'K'. One reason for this may be weak or noisy gyroscope data in the voting model that is similar to keys in area 'IOP'. Our system will load the microphone models when the gyroscope data is weak. These models can then evaluate the test inputs correctly based on the audio data. If more than 75% of the models predict the same key, then this key is chosen as the final prediction and no additional steps are performed.

In step 3, the test data is evaluated using the area-specific models. This step is only executed when the character models were not successful in predicting the keystroke. One reason can be the prediction of neighboring keys that belong to another area. For example, a character set based model may predict the key as 'A' when the actual key is 'Q'. These keys are neighbors on a standard Android QWERTY keyboard and may contain similar vibrations and audio characteristics. When the test data is evaluated specifically using the models for area 'QWE', then they can only predict a key from the selected area and may predict the correct key.

In step 4, the test data is evaluated using a voting model consisting of the two best algorithms for that area. The model determines the final prediction based on the prediction and confidence values of the two algorithms. This step will generally be executed when the test data is quite noisy and difficult to infer. We do not discard the data but attempt to make a final prediction based on the two best algorithms for that area.

5. EVALUATION

We evaluate our keystroke inference system using the following three metrics: The performance of the gyroscope, microphones and microphone + gyroscope (combined) sensors for different areas of the keyboard, the performance of our meta-algorithm applied to different machine learning algorithms in comparison to the traditional use of these al-

gorithms, and the performance of our meta-algorithm on the sample sets that were collected.

5.1 The Meta-Algorithm Evaluation

Table 2 shows the area-wise accuracy of the gyroscope (Gyro), microphone (Mics) and combined (Comb) sensors for a sample set collected using the QWERTY keyboard in portrait mode on the HTC One. The evaluation is based upon the accuracy with which characters were predicted correctly in different areas of the keyboard. We can see that the gyroscope predictions are inconsistent across areas as compared to the microphone which is consistent throughout. The gyroscope results are location dependent because of its hardware location. The areas IOP, ASD and NM are close to the gyroscope and do not exhibit significant vibration on the y axis. The inference depends more on the x axis vibrations yielding lower accuracy for these areas. The areas XZ, ASD, and QWE are further from the gyroscope and exhibit significant vibrations in the y axis. The inference depends on both axes vibrations yielding higher accuracy for these areas. Microphone predictions, on the other hand are location independent as they rely on the speed of sound traveling over the surface. We also see that the accuracy of the microphones is higher than the gyroscope for most of the areas, the only exception being area XZ. Combining the data from the two sensors yields higher accuracy than the individual sensors in some cases when the gyroscope data for an area is not weak. In a situation where the gyroscope data is weak for an area, our system attempts to detect this and performs inference using microphone data for that area.

Table 2: Area-wise accuracy of QWERTY keyboard sample set.

Area	Gyro	Mics	Comb
Q, W, E	84%	90%	92%
R, T, Y, U	86%	86%	92%
I, O, P	79%	90%	99%
A, S, D	84%	94%	97%
F, G, H	70%	89%	92%
J, K, L	71%	84%	89%
X, Z	88%	80%	98%
C, V, B	83%	93%	93%
N, M	77%	90%	100%

Table 3 shows the performance of our meta-algorithm when applied on individual algorithms in comparison to the traditional use of the algorithms (Table 1) for the same sample sets in Table 1. We see that our meta-algorithm can improve the accuracy of prediction for every sample set. We also see that the Decision Tree (DT) algorithm benefits the most from the meta-algorithm, with high increase in accuracies ranging from 8-13%. The Naive Bayes (NB) algorithm does not benefit much from our meta-algorithm, with lower increase in accuracies ranging from 0-4%.

Table 4 shows the final accuracy obtained by using our meta-algorithm with the sample sets that we collected in the office environment. We can see that it is possible to achieve high accuracy of predictions using the stereo-microphones on the device. We achieve an accuracy of 89.5% for the QWERTY keyboard for User3 and an accuracy of 94.5% for the Number keyboard for User2. We also see that in some cases such as the QWERTY keyboard sample for User3, combining the audio data with the gyroscope can boost the inference accuracy and it is possible to reach a higher accuracy of 94% even for the QWERTY keyboard. In some situations, using a combination of sensors may result in decrease of accuracy, such as for the number keyboard for User2. This is possible when the gyroscope data is weak. We built our system to detect such weak gyroscope samples using cross-validation of training samples but we did come across situations when the cross-validation yielded high accuracy for weak gyroscope data and used them to create models. One alternative could be to use some training samples solely for evaluating the models instead of using cross-validation with training samples that were used to create the models. There are some sample sets where the gyroscope inference accuracy is as low as 44-56%. We evaluated them carefully and found that our filtering techniques were not able to handle large gyroscope drifts. These drifts can be compensated by using Kalman filtering on the gyroscope data.

Table 3: Accuracies of meta-algorithm for some sample sets.

Keyboard	Sensor	DT	NB	NN	10-NN
HTC One					
QWERTY	Mics	94%	86%	93%	85%
QWERTY	Comb	95%	80%	93%	91%
Number	Mics	80%	81%	79%	76%
Number	Comb	81%	77%	81%	77%
Samsung S2					
QWERTY	Gyro	68%	61%	60%	55%
Number	Gyro	82%	76%	84%	79%

Table 4: Final single stroke meta-algorithm accuracy for several sample sets.

User	Keyboard	Count	Gyro	Mics	Comb
HTC One					
User1	Number	306	68%	93%	93%
User2	Number	200	44%	94.5%	93%
User3	Number	300	72%	91%	91%
User4	Number	300	75%	94%	95.5%
User5	Number	323	45%	83%	83%
User3	QWERTY	782	80.5%	89.5%	94%
User4	QWERTY	860	56%	83%	83%
User5	QWERTY	877	66%	73.5%	84%
Samsung S2					
User1	Number	137	75.5%	-	-
User2	Number	542	84%	-	-
User3	Number	202	83%	-	-
User4	Number	200	81.5%	-	-
User5	Number	512	81%	-	-
User1	QWERTY	366	63.5%	-	-
User2	QWERTY	620	77%	-	-
User5	QWERTY	312	74%	-	-

We also evaluated our keystroke inference system in environments when such an attack would not work so well. The gyroscope inference accuracy will be affected when the vibrations recorded during typing are noisy such as when typing while on a running vehicle, trembling hands during typing or when the touch is too soft. The microphones inference accuracy will be affected when the background noise is too high or when the touch is too soft for the microphones

to capture. In our experiments, we asked two participants to type in a noisy restaurant environment and achieved an accuracy of 42% and 56% for 212 and 226 test samples using the microphones, respectively. There were two participants who touched the screen very softly, and for them, our system achieved a low accuracy of less than 20% for both the microphones and the gyroscope. This was mainly because the keystroke could not be differentiated from the background noise. We also asked two participants to type on a tablet and achieved an accuracy of 36% and 45% for 106 and 234 test samples, respectively, using the gyroscope. These participants held the tablet in two hands and typed using their thumbs significantly reducing the vibrations caused due to typing.

5.2 End-to-end Attack Evaluation

To illustrate an end-to-end attack, we have also implemented a Trojan-like functionality in our Android application. The application starts a background service that queries for the foreground activity every five seconds. In Android, every UI page is known as an activity. An application may have multiple activities and every activity has a different class name. Using these class names, an adversary can determine the functionality that an application is performing. For example, for the banking application we used, the login activity is named as com.*****.mobile.*****Activity (parts of the class name hidden here for anonymity of application). The Trojan starts recording the microphones and gyroscope during the banking application login activity or when credit card input activity is in the foreground.

We collected 100 four digit random numbers and 100 sixteen digit random numbers simulating PIN numbers and credit card numbers from the Trojan service. These were recorded when users opened a particular activity of a banking application triggering the microphone and gyroscope recording. Table 5 shows the accuracy obtained by using our meta-algorithm with these PIN and credit card numbers. For four digit PIN numbers, the system correctly predicted 376 digits out of 400 digits and 8 additional keystrokes were detected by the system. Out of the 100 PIN numbers, 84 were predicted completely correctly in the first attempt. For sixteen digit credit card numbers, the system correctly predicted 1467 digits out of 1600 digits and 12 additional keystrokes were detected by the system. Out of the 100 credit card numbers numbers, 52 were predicted completely correctly in the first attempt.

Table 5: Final meta-algorithm accuracy for 100 PIN numbers and 100 credit card numbers.

Total	Correct	Correct Digits	Accuracy
PINs			
100	84	376	94%
Credit Cards			
100	52	1467	91.5%

We, thus, show that by building area specific models combined with meta-techniques, it is possible to achieve high accuracy of predictions such as 90-94% for the QWERTY and Number keyboard.

6. MITIGATION

The Android platform uses a variety of security and defense mechanisms against application's misbehavior, such as sandboxing and permissions. The security and effectiveness of such techniques have been studied in previous works [26], [17], [3], [12], [22], [10]. Cai et al. [6] discussed properties of a privacy protecing sensors. Although these mechanisms have proved to be effective against a large number of attacks, they are not effective against side channel attacks that bypass them.

We broadly classify mitigation techniques against side channel attacks as blocking or limiting accuracy [24], [7].

Blocking: When a sensitive application starts running, it will obtain a lock on mutually exclusive sensors and hardware that are only accessible to one process (app). Some sensors, such as the microphone and camera, fall into this classification. During this period when the application is using these sensors, no other process will be able to use them. Blocking is straightforward to design and implement, e.g. in the current Android SDK, this can be done by using the system calls in `android.media.AudioRecord` and `android.media.MediaRecorder`. However, this mechanism is not practical against non-mutually exclusive sensors, such as the gyroscope and accelerometer, without significant user experience degradation.

Limiting accuracy: The inference accuracy for both the microphones and gyroscope is highly correlated with the sampling rate. Table 6 shows the impact of the sampling rate on these sensors. We see that by reducing the sampling rate of the gyroscope from 100 Hz to 56 Hz in sample sets collected by the same user, the inference accuracy reduced from 79% to 58%. By lowering the sampling rate to 20 Hz, most keystroke vibrations were not detected yielding a low accuracy of 18%. Similarly, by reducing the sampling rate of the microphones from 48 KHz to 22.05 KHz, the inference accuracy reduced from 91% to 31%. As mentioned in Section 3, the sampling rate of the sensors can be reduced by introducing services that use more processing power, however, an adversary can still obtain high and constant sampling rate by using the Android NDK.

Table 6: Impact of sampling rate on inference accuracy for Number keyboard sample set.

Sampling Rate	Accuracy
Gyroscope	
100 Hz	79%
56 Hz	58%
20 Hz	18%
Microphones	
48000 Hz	91%
44100 Hz	89%
22050 Hz	31%

7. RELATED WORK & DISCUSSION

Cai & Chen [5] were the first to study the feasibility of number keystroke inference attacks using an Android device's orientation sensor. They developed an Android application called TouchLogger and collected three data-sets on a HTC Evo 4G phone using a Number only keypad in Landscape mode. Their experiments achieved a successful inference accuracy of about 70% on all three data-sets and showed that such an attack was indeed feasible.

Owusu et al. [23] studied the feasibility of character and area inference using an Android device's accelerometer sensor. They developed an Android application called ACCes-

sory for collecting data-sets on a HTC ADR 6300 phone from four participants. The participants were instructed to hold the phone and enter keys in a certain manner and several data-sets were collected for screen area and characters using a QWERTY keypad in Landscape mode. The data-sets were used to build a predictive model to evaluate the accuracy of area inference as well as passwords inference. Their experiments showed that, out of 99 6-character passwords, it was possible to successfully infer 6 character passwords in 5 trials.

Xu, Bai & Zhu [29] used two motion sensors, accelerometer and orientation, to study the feasibility of inference of the lock screen password and the numbers entered during a phone call, such as credit card and PIN numbers. They developed an Android trojan application called TapLogger that stealthily logs these numbers by using the accelerometer sensor to detect the occurrence of taps and the orientation sensor to infer which number was typed by the user. They collected data-sets of several tap events from three students using two phones, HTC Aria and Google Nexus (One), and unlike other experiments, performed the training and classification on the smartphone itself. Their experiments achieved an accuracy of about 99% for one user on the Google Nexus (One) and about 70% - 85% accuracy for the other users.

Cai & Chen [4] study the impact of different settings on the accuracy of predictions. They vary different factors in their settings, such as user habits, screen size, device type, layout orientation, etc. Their results show that side channel attacks stay possible and practical regardless of the setting. Although the attacks are feasible, the accuracy of such predictions vary. They use Google Nexus S, HTC Evo 4G, Galaxy Tab 10.1, Motorola Xoom in their experiments with 21 users, and demonstrate that 4 digit PIN can be guessed correctly after 81 attempts, 65% of times.

Aviv et al. [1] examine the possibility of side channel attack on smartphones by using the accelerometer. They demonstrate the possibility of inferring PIN and pattern password on four different smartphones; Nexus One, G2, Nexus S and Droid Incredible. Their results and evaluations are based on 24 users, divided into two groups of 12. Each group performs controlled (seating) and uncontrolled (walking) experiments. In the controlled setting, they reach an accuracy of 43% and 73% on PIN and pattern passwords respectively, within 5 attempts from a set of 50 PINs and 50 patterns. In the uncontrolled setting, they can predict PINs and patterns within 5 attempts 20% and 40% of times respectively.

Miluzzo et al. [21] present a framework called TapPrints that uses the accelerometer and gyroscope to identify icon locations and infer characters typed on a keyboard. They collected a data-set on two Android devices, the Google Nexus S and Samsung Galaxy Tab 10.1, and a iPhone 4. The experiment with icon locations was performed with the device in Portrait mode while other experiments with the character keypad were performed with the device in Landscape mode. By using ensemble machine learning, the author show that on an average, locations of icons can be inferred with 79% and 65% accuracy for the iPhone and Google Nexus S respectively and characters could be inferred with 65% accuracy. They also showed that some icons or characters can be inferred with an accuracy of up to 90% and 80% respectively.

Marquardt et al. [19] demonstrated that an Android application that has access to the device's accelerometer can be used to recover text typed on a physical keyboard the device is placed in close proximity with. They showed that if a device is placed within 2 inches of a physical keyboard and the keyboard is used for typing, then by measuring the relative physical position and distance between the vibration, they could recover words with accuracy as high as 80%.

Templeman et al. [27] propose a proof-of-concept visual malware called PlaceRaider. It opportunistically uses camera and other sensory data from a smartphone to create a 3D model of the user's environment. This 3D model allows the adversary to navigate and zoom in areas of interest to examine the individual images corresponding to that region. Another example of sensory malware is Soundcomber [25] which uses microphone to steal private information such as credit card numbers from phone conversations.

Zhou et al.[31] investigate side channel attacks based on the data from different sensor. They look at ARP information, speaker status and per-app data-usage statistics. From these channels they can infer user's identity, his geo-location and his driving route. Their app is also capable of monitoring when a target app is running to stealthily collect data and report back to a remote adversary.

Our experiment is different from previous related works as we are the first in our knowledge to use the stereo recording in smartphones and to combine acoustic and sensor information to infer keystrokes. We use the entire raw data and we use keyboard specific information in our meta-algorithm. By using the combination of acoustics and sensors and a multi-tier approach based on the areas of keyboards, we achieved a higher accuracy on the standard Android keyboard. Our experiment is similar to previous works as we too have focused on predicting keystrokes on the QWERTY and number keyboard but unlike previous experiments, we focused on smaller keys. For example, [5] use a number only keypad in landscape mode, [4] use different settings but mainly in landscape mode, [23] [21] use a QWERTY keypad in landscape mode, [29] [1] use larger keypads such as the lock screen or the number keypad shown during calls. We deduce that the accuracy of these experiments may reduce when tests are performed on the default Android character keyboard in Portrait mode. We demonstrate that by using a simple attack technique, it is possible to obtain a high inference accuracy even for smaller keys. Also, we demonstrate the feasibility of number and character inference using the sounds generated by the keystrokes and recorded by a device's stereoscopic microphones.

8. CONCLUSION

In this paper, we investigated the feasibility of keystroke inference on a smartphone by recording the sounds of key taps by the stereo-microphone and the vibrations by the gyroscope. In the future, we plan on implementing mitigation techniques for these side-channel attacks on Android and evaluate their effectiveness on several smartphones.

9. ACKNOWLEDGMENTS

We would like to thank Professor Kevin Butler for his helpful comments on our paper. This material is based upon work partially supported by the National Science Foundation under Grant No. CNS 1409453.

10. REFERENCES

[1] A. J. Aviv, B. Sapp, M. Blaze, and J. M. Smith. Practicality of accelerometer side channels on smartphones. In *Proceedings of the Annual Computer Security Applications Conference*, ACSAC '12.

[2] A. Bates, R. Leonard, H. Pruse, D. Lowd, and K. Butler. Leveraging usb to establish host identity using commodity devices. In *The 21th Annual Network and Distributed System Security Symposium*, NDSS '14.

[3] S. Bugiel, S. Heuser, and A.-R. Sadeghi. Flexible and fine-grained mandatory access control on android for diverse security and privacy policies. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13.

[4] L. Cai and H. Chen. On the practicality of motion based keystroke inference attack. In *Proceedings of the 5th International Conference on Trust and Trustworthy Computing*, TRUST'12.

[5] L. Cai and H. Chen. Touchlogger: Inferring keystrokes on touch screen from smartphone motion. In *Proceedings of the 6th USENIX Conference on Hot Topics in Security*, HotSec'11.

[6] L. Cai, S. Machiraju, and H. Chen. Defending against sensor-sniffing attacks on mobile phones. In *Proceedings of the 1st ACM Workshop on Networking, Systems, and Applications for Mobile Handhelds*, MobiHeld '09.

[7] S. Chakraborty, K. R. Raghavan, M. P. Johnson, and M. B. Srivastava. A framework for context-aware privacy of sensor data on mobile systems. In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, HotMobile '13.

[8] T. G. Dietterich. Ensemble methods in machine learning. In *Proceedings of the First International Workshop on Multiple Classifier Systems*, MCS '00.

[9] eMarketer. Smartphone users worldwide will total 1.75 billion in 2014. `http://bit.ly/LjwToI`, 01 2014. Last accessed 03/08/2014.

[10] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10.

[11] M. Frigo and S. G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proc. 1998 IEEE Intl. Conf. Acoustics Speech and Signal Processing.*

[12] C. Gibler, J. Crussell, J. Erickson, and H. Chen. Androidleaks: Automatically detecting potential privacy leaks in android applications on a large scale. In *Proceedings of the 5th International Conference on Trust and Trustworthy Computing*, TRUST'12.

[13] Google Inc. Android native development kit (sdk). `https://developer.android.com/tools/sdk/ndk/index.html`, 2014. Last accessed 02/25/2014.

[14] Google Inc. Android software development kit (sdk). `http://developer.android.com/sdk/index.html`, 2014. Last accessed 02/27/2014.

[15] Google Inc. Android, the world's most popular mobile platform. `http://developer.android.com/about/index.html`, 2014. Last accessed 03/08/2014.

[16] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 2009.

[17] H. Hao, V. Singh, and W. Du. On the effectiveness of api-level access control using bytecode rewriting in android. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13.

[18] L. I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. John Wiley and Sons, Inc., 2004.

[19] P. Marquardt, A. Verma, H. Carter, and P. Traynor. (sp)iphone: Decoding vibrations from nearby keyboards using mobile phone accelerometers. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11.

[20] S. Mckinley and M. Levine. Cubic spline interpolation.

[21] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. R. Choudhury. Tapprints: Your finger taps have fingerprints. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12.

[22] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel. Semantically rich application-centric security in android. In *Computer Security Applications Conference, 2009. ACSAC '09. Annual*, 2009.

[23] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang. Accessory: Password inference using accelerometers on smartphones. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*, HotMobile '12.

[24] K. R. Raghavan, S. Chakraborty, M. Srivastava, and H. Teague. Override: A mobile privacy framework for context-driven perturbation and synthesis of sensor data streams. In *Proceedings of the Third International Workshop on Sensing Applications on Mobile Phones*, PhoneSense '12.

[25] R. Schlegel, K. Zhang, X. Yong Zhou, M. Intwala, A. Kapadia, and X. Wang. Soundcomber: A stealthy and context-aware sound trojan for smartphones. In *The 18th Annual Network and Distributed System Security Symposium*, NDSS '11.

[26] J. Sellwood and J. Crampton. Sleeping android: The danger of dormant permissions. In *Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, SPSM '13.

[27] R. Templeman, Z. Rahman, D. Crandall, and A. Kapadia. PlaceRaider: Virtual theft in physical spaces with smartphones. In *The 20th Annual Network and Distributed System Security Symposium*, NDSS '13.

[28] G. Welch and G. Bishop. An introduction to the kalman filter. Technical report, 1995.

[29] Z. Xu, K. Bai, and S. 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*, WISEC '12.

[30] W. Zhou, Y. Zhou, X. Jiang, and P. Ning. Detecting repackaged smartphone applications in third-party android marketplaces. In *Proceedings of the Second*

ACM Conference on Data and Application Security and Privacy, CODASPY '12.

[31] X. Zhou, S. Demetriou, D. He, M. Naveed, X. Pan, X. Wang, C. A. Gunter, and K. Nahrstedt. Identity, location, disease and more: inferring your secrets from android public resources. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, CCS '13.

[32] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. In *Proceedings of the 19th Annual Network and Distributed System Security Symposium*.