



HAL
open science

Deciphering Text from Touchscreen Key Taps

Haritabh Gupta, Shamik Sural, Vijayalakshmi Atluri, Jaideep Vaidya

► **To cite this version:**

Haritabh Gupta, Shamik Sural, Vijayalakshmi Atluri, Jaideep Vaidya. Deciphering Text from Touchscreen Key Taps. 30th IFIP Annual Conference on Data and Applications Security and Privacy (DBSec), Jul 2016, Trento, Italy. pp.3-18, 10.1007/978-3-319-41483-6_1 . hal-01633681

HAL Id: hal-01633681

<https://inria.hal.science/hal-01633681>

Submitted on 13 Nov 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

Deciphering Text from Touchscreen Key Taps

Haritabh Gupta¹, Shamik Sural¹, Vijayalakshmi Atluri², and Jaideep Vaidya²

¹Department of Computer Science and Engineering,
Indian Institute of Technology, Kharagpur, India
{haritabh,shamik}@sit.iitkgp.ernet.in

²Management Science and Information Systems Department,
Rutgers University, USA
{atluri,jsvaidya}@rutgers.edu

Abstract. Exploiting acoustic emanations from electronic as well as mechanical devices as a means for side channel attack has recently emerged as a topic of security concern. In this paper, we present an attack methodology that can be used to extract the text typed by a user from the sound recorded by the built-in microphones of a mobile phone. We use signal processing techniques to initially extract a likely set of characters per tap on the touchscreen and then use natural language processing algorithms to find the most probable words and sentences that can be constructed from a given tap sequence. We also discuss the causes that result in this vulnerability and briefly present some countermeasures.

Keywords: Acoustic emanation, Touchscreen, Side-channel attack, Time Difference of Arrival (TDoA), NLP

1 Introduction

In recent times, mobile phones have almost become a necessity rather than a luxury for a large section of people throughout the world. Reports show that mobile phone penetration will reach approximately 70% of the global population by the year 2020. With increased reliance on mobile phones, especially for personal and confidential information exchange like mobile-commerce applications, the level of security ensured by these devices is an important topical issue.

Many users have started using phones for doing almost all of the work that they used to do on computers even a few years back. Although these mobile devices are yet to replace computers in their entirety, still a majority of the work has now shifted to the mobile devices due to various reasons such as mobility, portability, etc. Tasks range from composing e-mails, editing documents and accessing e-banking sites to surfing the net, using various applications (apps) like cab and hotel booking, chatting and participating in online social networking. All these tasks involve text input using a keypad, which is typically touchscreen in nature. In this paper, mobile phones (including smartphones and tablets) are hereinafter referred to simply as phones and they represent mobile devices that have a touchscreen user interface and have features like Internet access, support for recording and storing media files.

Most phones today are equipped with at least two microphones. Since these microphones are attached to the phone body, they pick up even the slightest sound produced on the surface of the phone. As a user taps on the touchscreen while typing, it produces a sound that may not be quite audible to the human ear due to the presence of ambient noise. Such tap sounds can, however, be captured using the microphones of the phone itself. A user hardly ever denies microphone use permission and Internet access to an app while installing the same. Also, an app is not considered to be suspicious unless it asks for some unusual combination of permissions. We show that even the above two seemingly innocuous permissions can be maliciously exploited to extract the text typed on the device and thus cause a security breach.

In the past, there has been some limited work that shows ways to recover text using acoustic emanations from mechanical keyboards [7]. They use external microphones to capture the sound and the recorded sound is used to train classifiers that differentiate between the sounds produced by different keys. It has also been shown that the sensors equipped with mobile phones such as accelerometer and gyroscope can be used for text extraction [10] [8] [11]. While such an approach uses the sensors attached to the phone, still it requires supervised learning. The possibility of exploiting the audio signal captured by the built-in microphones of a phone to decipher the typed text has never been studied, to the best of our knowledge.

This paper presents a novel methodology for carrying out a side-channel attack using acoustic emanations from the touchscreen keypad of a phone and that too using readily available tools. The approach is based on phone geometry and keypad layout, while using signal processing and natural language processing (NLP) techniques.

The rest of the paper is organized as follows: Section 2 describes our methodology and the associated algorithms in detail. Section 3 provides the complete workflow of an implemented system. Section 4 presents the results obtained in terms of accuracy of the proposed methodology. Related work is reviewed in Section 5. Some preventive measures are discussed in Section 6. Finally, Section 7 concludes the paper and presents some directions in which this work can be carried forward in future.

2 Attack Methodology

The complete methodology of the attack can be divided into two phases. The first phase, which is described in Sub-section 2.1, uses signal processing techniques to determine the probability of different characters that might have been entered with each tap using the audio signal received by the two microphones. It exploits the phone geometry and touchscreen keypad layout. In the second phase (Sub-section 2.2), NLP techniques are used to extract the probable word sequence that has been typed by processing the output of the first phase.

2.1 Character sequence extraction using microphone data

Most of the phones nowadays are equipped with at least two microphones with stereo recording capability. As a user types on the phone touchscreen, it produces a sound for each tap. The microphones, which are quite close to the touch surface, capture this sound. There is hardly any damping as the microphones are solidly attached to the phone body. The two microphones are almost invariably positioned at unequal distances from the point of touch. As a result, from the recorded sound of key taps, the time difference of arrival (TDoA) of peaks in the two microphones can be computed. For each TDoA, the locus of the touch point such that the difference of its distance from the two microphones is constant, forms a hyperbola, with the two microphones at its two foci. Depending on the sign of TDoA, it can be inferred which half of the hyperbola is meaningful for a given key tap. We compute TDoA using cross correlation [2] between the two audio signals recorded at the two microphones as explained below.

The co-ordinate system is defined such that the top left corner of the phone, while holding it upright and facing front, coincides with the origin and the left edge is parallel to the x-axis. As a result, one microphone (denoted as m_1) lies on the y-axis and the other (denoted as m_2) is in the first quadrant. Let the coordinates of the microphones be $(0, p)$ and (l, q) , respectively. Then the phone can be represented as shown in Figure 1a. Thus, the vertex of the hyperbola, as mentioned above, also lies in the first quadrant. Hence, we use the general equation of a hyperbola with vertex at (x_0, y_0) . The equation to represent this hyperbola from the obtained TDoA is as follows:

$$\frac{(x - x_0)^2}{a^2} - \frac{(y - y_0)^2}{b^2} = 1 \quad (1)$$

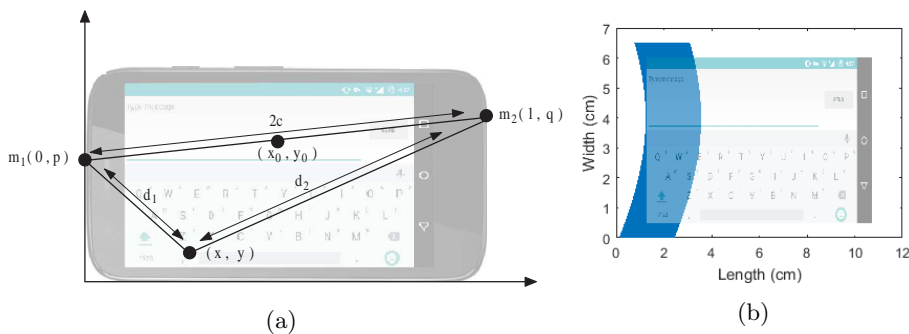


Fig. 1: (a) Hyperbola coverage on keyboard with $k = 3$. (b) Hyperbola coverage on keyboard with $k = 6$.

In Equation 1, (x, y) represents any arbitrary tap point while a and b denote parameters of the desired semi-hyperbola. The following equations are used to compute these parameters in terms of other known and measurable quantities.

$$a = \frac{\Delta d}{2}, \text{ where } \Delta d = |d_1 - d_2| \quad (2)$$

Here, d_i is the distance of the i^{th} microphone from the tapped point on the touchscreen, $i \in \{1, 2\}$. From Figure 1a, it is seen that

$$c = \frac{\sqrt{(l-0)^2 + (q-p)^2}}{2} \quad (3)$$

Here, l, p, q are phone parameters of which l denotes the length of the phone (dimension of the longer edge). p and q denote the distances of the microphones m_1 and m_2 , respectively from the base of the phone. Hence, we can write:

$$b = \sqrt{c^2 - a^2} \quad (4)$$

$$x_0 = c \quad (5)$$

$$y_0 = \min(p, q) \quad (6)$$

Simplifying and solving for x in terms of y from Equation 1, we get:

$$x = x_0 \pm \left(\frac{a}{b} \sqrt{b^2 + (y - y_0)^2} \right) \quad (7)$$

We use this equation to obtain the values of x corresponding to the values of y , where $y = [0, w]$, w being the width of the phone. Δd is computed using the TDoA obtained and the parameters of the recorded sound, namely, the sampling frequency f_s and the sample offset Δs . Sample offset is the value by which one of the audio signals needs to be shifted so that the correlation between the two signals is maximum. The rest of the parameter values are computed using Equations 2 to 7. Δd is obtained from Δs and f_s using Equations 8 and 9 as follows:

$$TDoA = \frac{\Delta s}{f_s} \quad (8)$$

$$\Delta d = v_{\text{sound}} \cdot TDoA \quad (9)$$

Thus, by computing the value of TDoA, we can obtain the desired hyperbola. Hence, the problem of determining the hyperbola parameters reduces to estimating the TDoA (or sample offset Δs) between the signals received by the two microphones. In the rest of the text, the terms *TDoA* and Δs are used interchangeably because from Equation 8, it is evident that we can compute one from the other as f_s is kept constant.

From the recorded audio signals of the microphones, the first task is to detect the tap instants (peaks in the signal). We present an automated scheme for peak detection, which is shown as the *Detect_Peak_Intervals* algorithm (Algorithm 1). It takes the audio signals as input and returns a set of time intervals, each containing only one tap. A detailed description of the algorithm follows.

Algorithm 1 *Detect_Peak_Intervals*

Input: Sound signal $x_i(t)$ where $i \in \{1, 2\}$, half window size w_h , threshold θ

Output: Set of intervals containing taps: *intervals*

```

1: for t=1 to n do ▷ Calculate Energy
2:    $A_i(t) = \sum_{n=t}^{t+10} x_i^2(n)$ , where  $i \in \{1, 2\}$ 
3: end for
4: for t = 1 to A.length do ▷  $A \in \{A_1, A_2\}$ 
5:    $s_k$  and  $e_k$  store beginning and end of a window around the selected time  $t$ 
6:    $[M, I] \leftarrow \max(A_i(s_k : e_k))$  where  $i \in \{1, 2\}$  ▷  $I$  : index of the maximum value
7:   if  $t = I + s_k$  and  $M > \theta \cdot \max(A)$  then
8:      $tmpIntr \leftarrow tmpIntr \cup (s_k : e_k)$ 
9:   end if
10: end for
11: intervals contains time readjusted and center positioned peaks
12: return intervals

```

In Algorithm 1, the *for loop* of Lines 1 - 3 computes the energy levels of the signal by accumulating 10 sample points together. Next, in the *for loop* of Lines 4 - 10, for each sample point, first the beginning and end points of the interval window are set in variables s_k and e_k for the k^{th} sample point. Lines 6 - 9 compute the maxima of the signal in the interval. If this value is greater than the threshold computed according to θ , then the corresponding interval is added to the set *tmpIntr*. Finally, the computed time intervals are readjusted so that the peaks are positioned at the center of each interval.

Algorithm 1 runs in $O(n)$ time as there are three loops in the algorithm of complexity $O(n)$, $O(n)$ and $O(k)$, respectively. Here, n is the number of sample points of the audio signal, k is the number of taps in the recording and $n \gg k$.

The output of Algorithm 1 is a set of intervals, each of which contains a tap. These intervals are further processed to get the TDoA corresponding to each tap. We next propose an algorithm (*Compute_TDoA*) to find the sample offset Δs of the two signals in a given interval, which is shown in Algorithm 2. It takes the audio signals (x_1 and x_2) from the two microphones (m_1 and m_2) and for each of the intervals obtained from Algorithm 1, finds their cross-correlation. Then it takes the median of the top p peaks.

The *for loop* of Lines 3 - 13 in Algorithm 2 selects each interval from the set of intervals one at a time. Next, the *for loop* of Lines 7 - 11 selects the top p peaks (empirically set to a value of 3) from the cross correlation. Line 12 adds the computed median to the set *lagMedian*. The final *lagMedian* set is returned by the algorithm.

The set of time lags thus obtained is used to determine the hyperbola corresponding to each tap. The locus of the valid semi-hyperbola obtained using Equation 7, when overlayed on the keyboard layout as shown in Figure 1b, returns the shortest distance of each key from the semi-hyperbola. The probability with which a key is pressed, i.e., the individual character entered by the user, is considered to be inversely proportional to the shortest distance of the key from

Algorithm 2 *Compute_TDoA*

Input: Set of intervals *intervals* each containing one peak, audio signal $x_i(t)$ where $i \in \{1, 2\}$ **Output:** Set of TDoA corresponding to each peak: *lagMedian*

```

1: lagMedian  $\leftarrow \phi$ 
2:  $p \leftarrow 3$  ▷ Median of  $p$  peaks is computed
3: for  $n = 1$  to intervals.length do
4:    $s_i = x_i(s_n : e_n)$ , where  $(s_n, e_n) = \text{interval}[n]$  and  $i \in \{1, 2\}$ 
5:    $[acorr, lag] \leftarrow \text{xcorr}(s_1, s_2)$  ▷ Compute cross-correlation
6:   lagTemp  $\leftarrow \phi$ 
7:   for  $i = 1$  to  $p$  do
8:      $[M, I] \leftarrow \text{max}(|acorr|)$  ▷  $I$  is the index of the maximum value
9:      $acorr[I] \leftarrow 0$ 
10:    lagTemp  $\leftarrow \text{lagTemp} \cup M$ 
11:   end for
12:   lagMedian  $\leftarrow \text{lagMedian} \cup \text{median}(\text{lagTemp})$ 
13: end for
14: return lagMedian

```

the semi-hyperbola. Thus, we obtain the top k most probable characters for each TDoA and hence, for each key tap.

2.2 Word and sentence inferencing from probable character sequence

This sub-section describes the methodology used to construct sentences from the sequence of characters obtained using Algorithm 2 of Sub-section 2.1.

To carry out the attack, uni-gram, bi-gram and tri-gram probabilities are used. The character-level n -grams as well as word-level n -gram counts are learned from the e-mail corpus¹ in a pre-processing step.

The probability of occurrence of an n -gram is the count of that n -gram divided by the total number of n -grams. However, since the number of n -grams is very large, the probability value comes out to be too small. To compute the probability of a string, we need to multiply these n -gram probabilities. If we use the actual probability values, the product might cause an underflow. On the other hand, taking the product of the count of n -grams results in an overflow. Hence, to avoid both underflow and overflow, instead of using the probabilities directly, we introduce the notion of *pseudo-probability*, which is the \log of the count of n -grams. The steps for computing pseudo-probability are shown in Algorithm 3 (*Compute_pseudo-probability_of_a_string*).

¹ Enron Mail corpus: <https://www.cs.cmu.edu/~./enron/>

Algorithm 3 *Compute_pseudo-probability_of_a_string***Input:** Candidate string str , weight factor ω , maximum n -gram length $nGrams$ **Output:** Pseudo-probability of string: $prob$

```

1:  $prob \leftarrow 0$ 
2: for  $l = 1$  to  $nGrams$  do
3:   if  $l > str.length$  then
4:     break
5:   end if
6:    $prob_t \leftarrow 0$ 
7:   for  $i = 1$  to  $str.length$  do
8:      $subStr \leftarrow str[i : i + l]$ 
9:      $prob_t \leftarrow prob_t + \log(gramCount(subStr)) + 1$ 
10:  end for
11:   $prob \leftarrow prob + (prob_t) \cdot \omega^l$    $\triangleright$  weighted sum of  $n$ -gram pseudo-probabilities
12: end for
13: return  $prob$ 

```

In this algorithm, the outer *for loop* of Lines 2 - 12 runs the inner loop considering i -grams in its i^{th} iteration. The inner *for loop* of Lines 7 - 10 picks up an i -gram from the string, computes its *pseudo-probability*, and adds it to the temporary variable $prob_t$. Finally, Line 11 adds the temporary probability ($prob_t$) to the total probability variable $prob$ of the string after multiplying it with the appropriate weight factor.

There are two loops in Algorithm 3. The outer loop runs n times if n -grams are used for the computation, while the inner loop runs in the order of the length of the string, denoted here as l . Hence, the overall time complexity is $O(n \cdot l)$.

From the computed score of strings, we build an inference tree to determine the most probable text that was typed. The language model is based on n -gram and inference based tree [13] [4]. The character-level n -grams are used to construct a word-level inference tree. At each level, we include the next set of probable characters and recompute the probabilities of the strings thus obtained. At every level, only the top k nodes are retained and the rest are pruned. This prevents the tree from expanding exponentially. In the last level of the tree, we are left with k most probable words that can be constructed using the character sequences obtained from the algorithms presented in Sub-section 2.1.

The node used to create the tree in Algorithm 4 is a three-tuple represented as $node = \langle string, prob, children \rangle$, where $string$ is the partial word that is stored in the node. $prob$ stores the *pseudo-probability* of this partial word computed using Algorithm 3. The children for this node is stored in the list $children$. In this tree, for any child node c , if p is its parent node, then $p.string$ is a prefix of $c.string$ (*Property 1*). Also, the tree is constructed in such a way that at any level l of the inference tree, the maximum number of nodes does not exceed n , where n is a user-defined input parameter (*Property 2*).

We propose an algorithm (*Generate_Word-level_inference_tree*) that builds the word-level inference tree whose properties and node structure are defined in the previous paragraph. It is shown in Algorithm 4.

Algorithm 4 *Generate_Word – level_inference_tree*

Input: Probable characters of a word in 2-d array *alpha*, child count *c*
Output: set of *c* most probable words: *words*

- 1: initialize *root* node and set all parameters to ϕ
- 2: **for** $i = 1$ **to** *alpha.length* **do**
- 3: *allLeaves* \leftarrow *getLeaves*(*root*)
- 4: *trLeaves* \leftarrow *truncateLeaves*(*allLeaves*, *c*)
- 5: **for all** *curr* \in *trLeaves* **do**
- 6: **for all** *ch* \in *alpha*[*i*] **do**
- 7: *t* \leftarrow ϕ
- 8: *t.string* \leftarrow *curr.string* \cdot *ch* \triangleright append *ch* to the string of parent node
- 9: *t.prob* \leftarrow *stringProb*(*t.string*)/*t.string.length* \triangleright Compute pseudo-probability of string *t.string* and normalize by string length
- 10: *curr.children* \leftarrow *curr.children* \cup *t*
- 11: **end for**
- 12: **end for**
- 13: **end for**
- 14: **return** *getLeaves*(*root*)

This algorithm builds the inference tree one level at a time. Each level of the tree adds to the tree, the set of probable characters corresponding to the next tap. The outermost *for loop* of Lines 2 - 13 selects the set of characters corresponding to the i^{th} tap in the i^{th} iteration. Next, the leaf nodes in the tree are pruned so that the tree retains Property 2. The *for loop* of Lines 5 - 12 selects each of the remaining leaves one at a time. Pruning of leaves is done using the *TruncateLeaves* procedure as shown in Algorithm 5. The *for loop* of Lines 6 - 11 adds new child nodes to the current node and sets the *string* so that they satisfy Property 1. The *prob* value in new nodes (denoted as *t*) is computed using Algorithm 3. Algorithm 4 terminates when all the probable characters corresponding to all the key taps have been added to the tree.

There are three nested *for loops* in Algorithm 4, with time complexity of $O(n)$, $O(c)$ and $O(k)$, respectively. Here, n is the total number of taps, k is the number of probable characters returned per tap and c is the number of allowed child nodes per level. Invocation of *TruncateLeaves*() contributes to $O(l)$ time, where l is the number of nodes at a level prior to pruning. So, the time complexity of Algorithm 4 is $O(n \cdot (O(l) + (c \cdot k)))$. c and k being fixed parameters and independent of the input size n , the time complexity becomes $O(n \cdot l)$.

A similar approach is used to build a sentence-level inference tree based on the most probable word sequence. As in the word-level inference tree, at each level, we prune all the nodes except the top k (based on the pseudo-probability value). In the last level, we are then left with the top k most probable sentences that can be constructed from the set of probable words and hence from the set of probable characters.

Both the word-level as well as sentence-level inference trees use leaf pruning to prevent exponential expansion of the tree. The pruning algorithm (*TruncateLeaves*)

first takes the list of all children at the current level as shown in Algorithm 5. It then deletes all nodes except the top c nodes and returns the modified list. The time complexity of this algorithm is $O(n \cdot c)$, where n is the number of leaves prior to pruning and c is the desired number of leaves after pruning. Since c is a constant, the overall time complexity is $O(n)$.

Algorithm 5 *Truncate_leaves*

Input: List of leaves l_o , maximum number of child nodes c

Output: Set of c most probable leaves: l_t

```

1:  $d.prob \leftarrow -1$  ▷  $d$  is a dummy node
2: for  $i = 1$  to  $c$  do
3:    $max.prob \leftarrow -1$ 
4:   for all  $t \in l_o$  do ▷ Node with maximum pseudo-probability is stored in  $max$ 
5:     if  $t.prob > max.prob$  then
6:        $max \leftarrow t$ 
7:        $index \leftarrow l_o.indexOf(t)$ 
8:     end if
9:   end for
10:   $l_t \leftarrow l_t \cup max$  ▷ Add  $max$  to modified set of leaves
11:   $l_o[index] \leftarrow d$  ▷ Replace  $max$  by dummy node  $d$ 
12: end for
13: return  $l_t$ 

```

In this section, we have described the two phases of the proposed attack methodology and the algorithms used to carry out the attack. The next section presents the details of implementation of a complete system for carrying out such an attack.

3 Implementation Details

This section presents an implementation flow of the attack starting from the recording of sound to the extraction of estimated typed text. A trojan app is installed on the victim's phone that creates audio files from the recorded tap sounds. It needs only microphone permission during installation and Internet access permission so that the audio files can be sent to the attacker's server for further processing. The processing of audio files needs information extracted from preprocessed text corpus which is quite large to store in the phone and the word extraction will be quite computation expensive for the phone processor. Hence, we propose to perform all computations on the server rather than on the phone.

The first module at the server end processing in the overall block diagram of Figure 2 takes the raw audio file as input and generates the list of time intervals that contain peaks (taps on the touchscreen). Our recording environment emulated any quiet room like library, office, conference room, etc., typical places

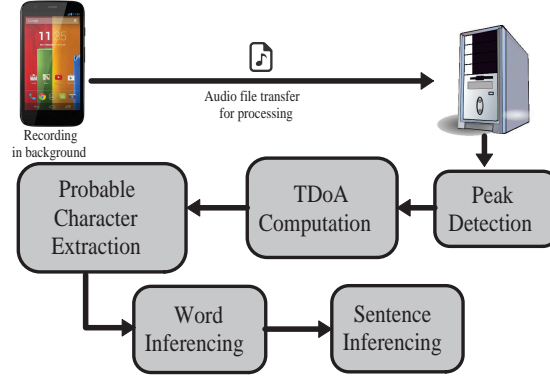


Fig. 2: Block diagram of attack flow implementation

where a user is likely to reveal personal information or carry out online transactions. During the experiments, the source of ambient noise was ceiling fan and air conditioner.

The list of peaks is given as input to the next module along with the audio file. This module, as described in Algorithm 2, computes the cross correlation between the sound waves from the two microphones. Figure 3a shows the energy spectrum of the audio signals as captured by the two microphones corresponding to a tap. It is seen in the figure that there are sharp peaks during the tap, preceded and followed by periods of silence. This property of the microphone captured signal is used to extract peaks from the audio files.

The time domain representation of the two audio signal amplitudes in Figure 3b shows that there is a noticeable shift between them. Figure 4a presents a detailed view of the two audio signals corresponding to a tap. Cross correlation is computed between these two signals. The cross correlation peak is the actual TDoA between the signals. The computed TDoA is used as a parameter for the hyperbola that corresponds to the probable region of tap. It is also observed from Figures 3a-3b that out of the two signals, one has higher amplitude than the other. This difference in amplitude is used to select the valid semi-hyperbola. As an example, if signal from microphone m_1 has higher amplitude, then the tap sound must be emanating from the left half of the keyboard (the half that is closer to microphone m_1).

Note that, the sign of TDoA can also be used to select the correct half of the hyperbola. For example, if the peak in the signal from microphone m_1 occurs before the peak in the signal from microphone m_2 , it can be concluded that the click originated from that half of the keypad which is closer to microphone m_1 .

The magnitude of the TDoA is used next to obtain the probable region on the keyboard layout where the tap occurred. This is shown in Figure 4b. In this example, the key a was tapped. The computed TDoA was used to plot the hyperbola and depending upon the sign of the TDoA, the other half of the hyperbola was discarded. We see that the maximum probability region passes over a . Thus,

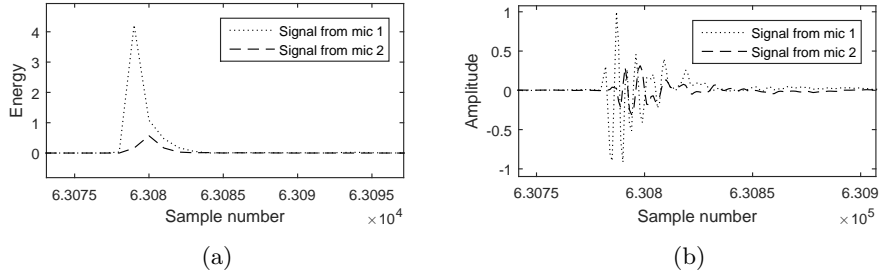


Fig. 3: (a) Energy spectrum of the input signal corresponding to a single tap. (b) Input signals in time domain corresponding to a tap.

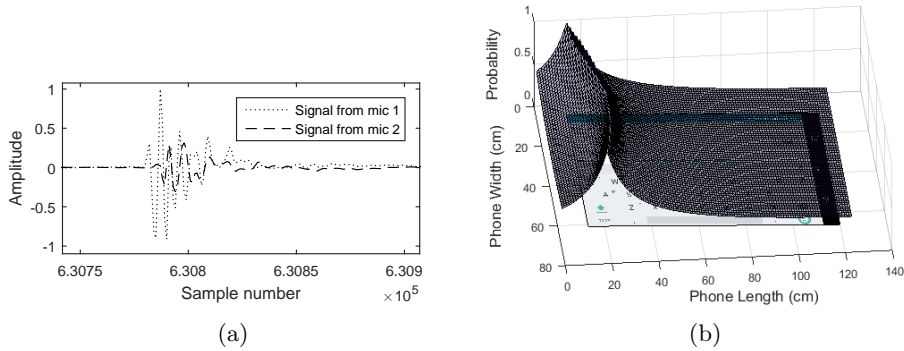


Fig. 4: (a) Shift between two signals captured by the two microphones in time domain. (b) Overlay of hyperbola region on phone keyboard.

for every tap, we determine the most probable region of the tap. To get the k most probable characters, we set the width of the band in such a way that k number of keys fit into the band of the hyperbola. The *Probable Character Extraction* module of Figure 2 then returns the k characters that have the highest intersection area with the plotted region.

Text inferencing from the probable sequence of letters is done by building the inference tree and pruning leaves at each level as explained in Algorithms 4 and 5. This approach is used for both word level as well as sentence level inferencing of Figure 2.

4 Experimental results

We carried out detailed experiments with our system implementation described in Section 3. In this section, we present several important observations that were made during the experiments as well as quantitative result on accuracy. The results consist of two parts. Sub-section 4.1 presents the results obtained

till the extraction of the probable characters from the audio signal captured by the microphones. Sub-section 4.2 then presents the results of the stages related to inferring of words from probable characters and that of sentences from the probable set of words.

The dataset that we used to test the hypothesis and proposed methodology has following statistics. On average, the audio encoding of each character takes ≈ 82 kB per character, while the average typing speed was ≈ 2 characters per second. Moreover, the complete methodology presented in Section 2, takes about 70 seconds to decipher text from a recording containing ≈ 100 taps.

4.1 Audio processing results

Figure 5 shows the plot of various TDoAs measured in number of sample points and the character set that are returned corresponding to each TDoA. In one of the phones used for experimentation (layout shown in Figure 1b), $TDoA \in [-8, 10]$. We plot each TDoA over the keyboard layout keeping $k = 3$ as described in Section 3 and return the character set corresponding to that TDoA. A linear trend is seen in the values of TDoA and the characters returned also shift from the left end of the keypad to the right end. For example, for $TDoA = 10$, the returned character set is $\{q, w, a\}$ and for the minimum value of $TDoA = -8$, the returned character set is $\{p, l, o\}$.

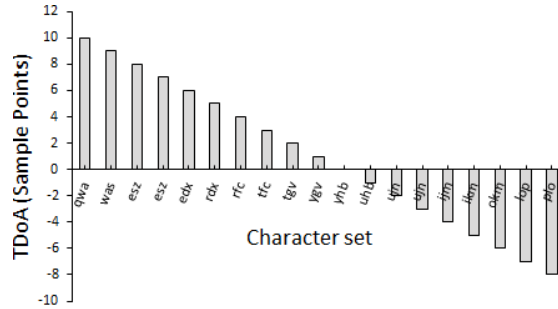


Fig. 5: Character sets for different values of TDoA

In Figure 6a, we plot the distribution of the computed value of TDoA for one particular letter (clicked letter is a). The set of top k most probable characters containing the typed character can be extracted with quite good accuracy. From Figure 6b it is seen that, using $k = 3$, the character that was actually typed is present in the set of probable characters 90% of the time. For the rest 10%, the computed TDoA deviates only by 1. Hence, it can be concluded that TDoA is a robust feature to be used for the proposed attack methodology. This set of probable characters is used as input for the NLP module, the results of which are presented in the next sub-section.

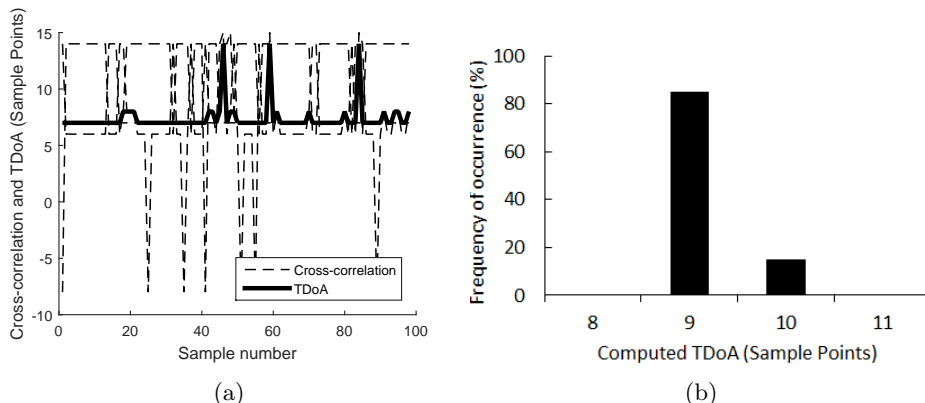


Fig. 6: (a) Computed values of TDoA obtained from cross-correlation for 100 taps of *a*. (b) Distribution of the computed TDoA.

4.2 Text extraction results

Figure 7a shows the variation in word occurrence frequency and word level accuracy with change in word length. We took a test set of 30 messages of varying sizes. The text was deciphered using the proposed attack methodology from the tapping sounds picked up by the two microphones and compared with the actual typed text. The bars in the graph show the fraction of the count of occurrences and the fraction of the number of times they were recovered correctly. The piecewise continuous line represents the accuracy and a linear trend line is also plotted in the graph for reference. From the figure, it is observed that accuracy approaches 1.0 when the word length is 1 or 2. This is because these are usually stop words like *a*, *an*, *be*, *at*, etc. Such words are quite common and thus can be inferred accurately. The next peaks occur at word lengths 3, 4 and 5. This is due to the fact that, in the English language, the average word length is 5.1 letters [3]. Words with greater length occur rarely, as a result of which, there are less number of samples with longer length to train the language model. Therefore, while inferring, the accuracy decreases with increasing word length beyond 5. The average word level accuracy is 0.81.

Figure 7b shows the variation in accuracy of complete text with message size. For this study, we use the same data set as in Figure 7a. It is seen from the figure that the accuracy lies in the range of 0.57 – 0.95, with an average of 0.88. It is also observed that accuracy tends to increase with increasing length of the text. The NLP algorithm assumes that the text is composed of English words only. Hence, such a trend is observed, because with increase in message size, the ratio of valid English words to the total number of words increases.

A few local variations are also seen in the figure. This point was further investigated. It was found that our assumption that the user would be typing in English, i.e., using words from the valid English vocabulary did not always hold. Often the message contained certain words like URLs, dates, proper nouns, etc.,

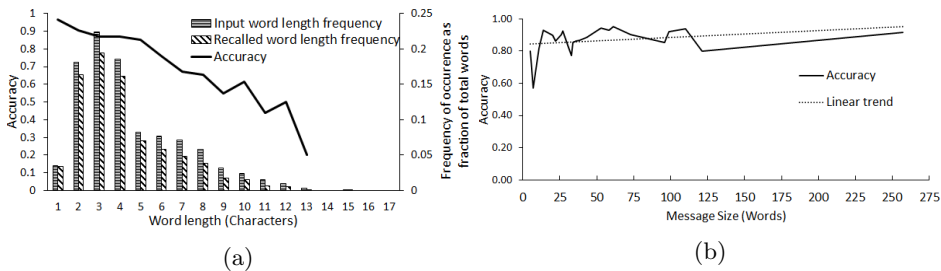


Fig. 7: (a) Accuracy for different word lengths (b) Variation in accuracy with message size

that are not there in the English dictionary, which caused some degradation in performance.

5 Related work

It has been shown in the literature that emanations from devices can be exploited in ways to reveal information associated with the source, thereby breaching security. There are several variations of this type of attack based on the type of emanation [6] [9]. Particularly, acoustic emanations from various devices is an important field of research.

Asonov et al. [1] show that the keyboard strokes can be recovered using the acoustic emanations from mechanical keyboards. They argue that although the emanations from different keys sound similar to human ear, they are actually different and a classifier can be trained to distinguish between the keys. Fast Fourier Transform is used as the feature set of the sound and backpropagation neural network is used for classification.

The above issue was readdressed by Zhuang et al. [14], who presented a novel technique using which even random keystrokes can be recovered. Unlike the attack methodology of [1], which requires labeled data to train the neural network before it can be used as a classifier, this attack uses the statistical constraints of the English language to train a model by unsupervised learning. They use cepstrum features, which were shown to be superior to FFT. Another related work [5] provides a brief explanation and analysis of the possible attacks that can be carried out using mobile phones. It presents a nomenclature for various attacks. Some preventive measures are suggested that can be implemented to minimize the risk from the identified possibilities of attack.

Since our work is aimed at deciphering text from touchscreen mobile phones, we next present some work in this field. Miluzzo et al. [10] discuss the possibility of use of the accelerometer and gyroscope sensors of phones and present a method to recover the tap position as well as the typed text. They propose that each point on the screen, when tapped, produces different values of linear and angular acceleration, which can be captured using the phone sensors.

Zhu et al. [12] present a context-free attack technique to recover keystrokes using phones as recording devices placed in the vicinity of the keyboard being typed on. With enough number of such external microphones, the keystroke region can be pinpointed with good accuracy. To make the attack even more general, the keyboard orientation is also assumed to be unknown beforehand. Keyboard reconstruction is formulated as an optimization problem and the most likely keyboard layout is taken as the orientation of the keyboard. Finally, the keystrokes are recovered.

There is, however, no work that recovers key taps from touchscreen keypads using built-in microphones of a mobile phone as reported in this paper. Such an attack is more realistic than the ones mentioned above since the previous attacks require external hardware to record the sound of key presses. Setting up of extra hardware in the vicinity of the victim while he is typing is susceptible to detection. Since the proposed methodology uses the microphone already attached to the device, it can silently record and transmit the audio signals to the attacker's server.

6 Preventive measures

There are a few basic preventive measures, which however, might reduce the impact of such an attack, even though it cannot be eliminated fully. For instance, if the user keeps the key-press tone ON, then with each tap, an additional sound will be generated masking the actual sound created by the tap to a certain extent. A similar effect may be achieved if the vibration on each key-press is kept ON. However, both of these counter-measures are inconvenient for the user and they drain the battery as well.

Another approach that may be adopted by the operating system developers is that the OS can display notifications (using icons) to depict which devices are currently being used in the background. Thus, the user will be notified if the microphone or any other sensor is running in the background. The OS may also display the apps which are currently using that sensor data. In this way, the user can become aware of any background app trying to eavesdrop and possibly terminate it to prevent the attack. However, such measures demand a certain level of maturity and knowledge on the part of the user.

7 Conclusion

The primary goal of this paper was to bring to light the vulnerabilities of touchscreen phones, which we have exploited and devised an attack methodology that is reasonably accurate. It may be noted that, any form of encryption will not help to prevent this attack as the proposed methodology records the tap sounds generated at the time of typing itself.

The direction for future work on improving the attack methodology is three-fold: (i) improving accuracy of the signal processing unit (ii) use of other NLP techniques to enhance the performance of the text inferencing phase and also to

detect special characters and numerals. This would allow the attack to detect passwords as well. (iii) Fusion of data from more number of sensors. Sensors such as gyroscope and accelerometer are also present in almost all smartphones. For each key tap, the accelerometer and gyroscope values would vary. Although this data is a bit more difficult to make use of as compared to the microphone data, still it is a possibility that is worth exploring to see how the attack performs.

References

1. Asonov, D., Agrawal, R.: Keyboard acoustic emanations. In: Proceedings of IEEE Symposium on Security and Privacy. pp. 3–11 (2004)
2. Benesty, J., Chen, J., Huang, Y.: Time-delay estimation via linear interpolation and cross correlation. *IEEE Transactions on Speech and Audio Processing* 12(5), 509–519 (2004)
3. Bochkarev, V.V., Shevlyakova, A.V., Solovyev, V.D.: Average word length dynamics as indicator of cultural changes in society. arXiv preprint arXiv:1208.6109 (2012)
4. Brown, P.F., Desouza, P.V., Mercer, R.L., Pietra, V.J.D., Lai, J.C.: Class-based n-gram models of natural language. *Computational linguistics* 18(4), 467–479 (1992)
5. Dagon, D., Martin, T., Starner, T.: Mobile phones as computing devices: the viruses are coming! *Pervasive Computing* 3(4), 11–15 (2004)
6. Halevi, T., Saxena, N.: Acoustic eavesdropping attacks on constrained wireless device pairing. *IEEE Transactions on Information Forensics and Security* 8(3), 563–577 (2013)
7. Hussain, M., Al-Haiqi, A., Zaidan, A., Zaidan, B., Kiah, M.M., Anuar, N.B., Abdalnabi, M.: The rise of keyloggers on smartphones: A survey and insight into motion-based tap inference attacks. *Pervasive and Mobile Computing* pp. 1–25 (2015)
8. Liu, X., Zhou, Z., Diao, W., Li, Z., Zhang, K.: When good becomes evil: Keystroke inference with smartwatch. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 1273–1285. ACM (2015)
9. Meng, W., Lee, W.H., Murali, S., Krishnan, S.: Charging me and i know your secrets!: Towards juice filming attacks on smartphones. In: Proceedings of the 1st ACM Workshop on Cyber-Physical System Security. pp. 89–98 (2015)
10. Miluzzo, E., Varshavsky, A., Balakrishnan, S., Choudhury, R.R.: Tapprints: Your finger taps have fingerprints. In: Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services. pp. 323–336 (2012)
11. Ping, D., Sun, X., Mao, B.: Textlogger: inferring longer inputs on touch screen using motion sensors. In: Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks. pp. 1–12. ACM (2015)
12. Zhu, T., Ma, Q., Zhang, S., Liu, Y.: Context-free attacks using keyboard acoustic emanations. In: Proceedings of the ACM Conference on Computer and Communications Security. pp. 453–464 (2014)
13. Zhu, Z., Bernhard, D., Gurevych, I.: A monolingual tree-based translation model for sentence simplification. In: Proceedings of the 23rd International Conference on Computational Linguistics. pp. 1353–1361 (2010)
14. Zhuang, L., Zhou, F., Tygar, J.D.: Keyboard acoustic emanations revisited. *ACM Transactions on Information and System Security* 13(1), 1–26 (2009)