Keyboard Acoustic Emanations

Nicolò Fornari
Coding Theory and Cryptography
University of Trento

***Abstract*—In ground-breaking research, Asonov and Agrawal showed that is possible to differentiate the sound emanated by different keys when typing on a keyboard. This finding leads to a new type of keylogger which is absolutely non invasive being neither software or hardware. In this paper I discuss my implementation of Asonov and Agrawal's research.**

***Keywords*—Acoustic emanations, keylogger, security, sidechannel attacks**

I. Introduction

Emanations produced by electronic devices have long been a source for attacks on the security of computer systems. Past attacks have exploited electromagnetic emanations [4] as well as optical emanations [5],[6].
In [1] Asonov and Agrawal investigated the acoustic emanations of PC keyboards, in order to eavesdrop upon what is being typed. Their attack is based on the hypothesis that the sound of clicks can differ slightly from key to key, although the clicks of different keys sound very similar to the human ear. Experimental results show that a neural network can be trained to differentiate the keys to successfully carry out this novel form of eavesdropping.
The attack is inexpensive and non-invasive. It is inexpensive because in addition to a computer the only other hardware required is a parabolic microphone. It is non invasive because it does not require physical intrusion into the system; the sound can be recorded from substantial distance.

A. Paper Layout

Section II regards the outline of the attack. In Section III I discuss the characteristics of a key stroke, how clicks can be automatically extracted from a recording and which features are selected.
Section IV is about neural networks: it gives a brief introduction to the topic by explaining what are neurons and weights, how training works and finally explaining how the network is used to distinguish different clicks.
In Section V I present my experimental results. I discuss related work in Section VI and conclude in Section VII.

II. The attack

The attack works in two distinct steps: first a recording of a user typing is needed, second a neural network is trained over recordings of key strokes coming from a keyboard of the same model of the victim's one.
With these two pieces of information it is possible to distinguish different clicks by using the neural network as a classifier. The attack is based on the hypothesis that the sound of clicks might differ slightly from key to key, although the clicks of different keys sound similar to the human ear. Before proceeding with technical details of the attack it is fundamental to understand the structure of a click or key stroke.

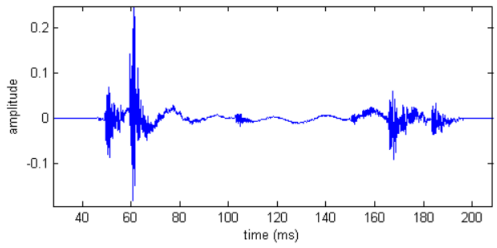


Fig. 1. The acoustic signal of one click

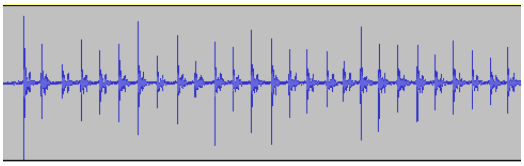


Fig. 2. The acoustic signals of some training clicks

III. Sound processing

A. The key stroke

A key stroke lasts for about 100-150ms. As shown in Figure 1 the acoustic signal has two distinct peaks corresponding to pushing and releasing the key. Moreover the *push peak* can be observed to consist of two distinct active intervals corresponding respectively to a finger touching the key (*touch peak*) and the key hitting the keyboard supporting plate (*hit peak*).

B. Click extraction

The process of automatically extracting every single click from a recording is not trivial at all, considering that it has to be unsupervised.
My first approach to this matter was very naive: I recorded the test clicks as in Figure 2. Then I defined the following algorithm:

1) Start copying the values of a click when they are above a threshold
2) Copy until a whole click is reached (this is possible because the sampling is known and so the click length)
3) Go to step 1)

This algorithm works but has two major drawbacks:

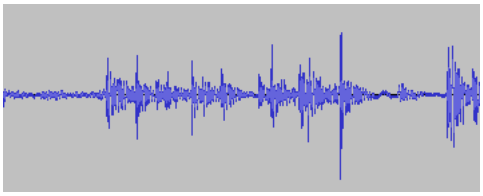


Fig. 3. The acoustic signal of a user typing. In this case clicks are harder to extract respect to Fig. 2

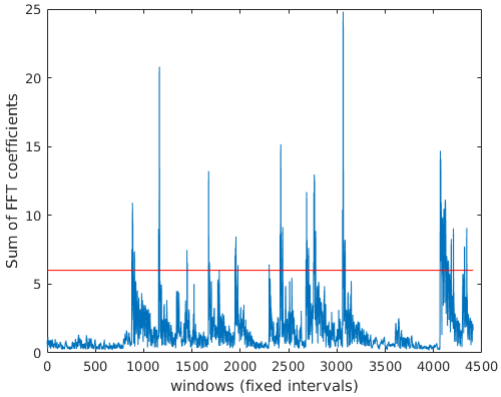


Fig. 4. Energy levels over the duration of 10 keystrokes. The red line is the threshold, one can count the number of spikes above the threshold. Such spikes correspond to recognized clicks

- The touch peak is likely to be lost since only the hit peak will be over the threshold. On the other hand if we set a lower threshold we are not able to distinguish clicks from noise)
- This method can only be applied to recorded training clicks, there is no way it will work with the recording of a real user typing (compare Fig. 3 with Fig. 2).

Having to deal with those issues I looked for a better method which I found in [2]. The idea is simple and extremly effective: calculate windowed discrete Fourier transform of the signal and use the sum of all FFT coefficients as energy. Then set a threshold to detect the start of keystrokes, see Fig. 5. This procedure is described by the pseudo code below:

```
sound ← Open("recording.wav")
ws ← 40 // window size
windows ← Size(sound)/ws
bins ← zeros(windows)
for i ∈ 0..windows do
    transform ← FFT(sound[ws×i,ws×(i + 1)])
    for j ∈ 1..ws do
        bins[i] ← bins[i] + transform[j]
```

In order to further clarify the algorithm look at Figure 4: the signal is partitioned in equal parts, called *windows*, represented on the x axis. The sum of the FFT coefficients belonging to the same window is plotted on the y axis. Spikes clearly point out where the clicks begin. However the program can not look at the graph as a human would do, it needs a threshold to determine when a click begins. For the victim's recording I

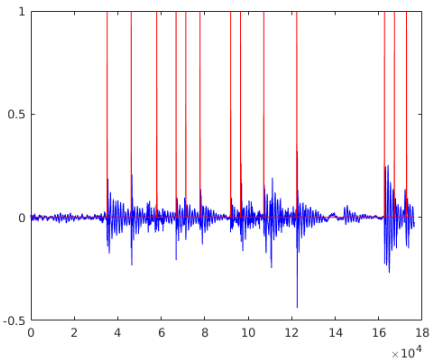


Fig. 5. The vertical red lines mark the beginning of each click in the sound recording of Fig. 3

ask the user to enter an appropriate threshold by looking at the graph. I chose this option because I do not know in advance the number of total clicks. On the other hand when dealing with training clicks I already know the total number of clicks I am expecting. Thus I managed to determine an appropriate threshold dynamically without user interaction: simply look at the pseudo code below:

```
sound ← Open("recording.wav")
expectedClicks ← 100
working ← true
while working do
    clicks ← extractClicks(sound, threshold)
    if size(clicks) == expectedClicks then
        working ← false
    threshold ← threshold-1
```

C. Feature extraction

The raw sound produced by key clicks is not a good input for training a neural network. Neural networks are recommended to be trained with an input consisting of several dozen to several hundred of numeric values between 0 and 1, which corresponds up to 1kB input. On the other hand, the size of the acoustic signal corresponding to a keyboard click is about 10kB. Therefore relevant features must be extracted from the raw sound.

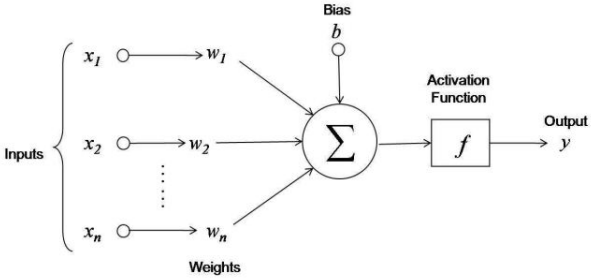
In my project I followed Asonov and Agrawal's choice of taking the Fast Fourier transform of the entire push peak as feature. Of course there is no rule nor guideline: one could choose just a portion of the push peak (namely the touch peak or the hit peak), although harder to isolate, and apply to it melfrequency cepstral coefficients (MFCC) instead of FFT as in [3].
It all comes to experimental results. However I would like to point out that looking for the highest accuracy possible is not a good strategy as it is likely to produce a model that has been overfit, hence with poor predictive performance.

IV. Neural Network

A. An introduction to neural networks

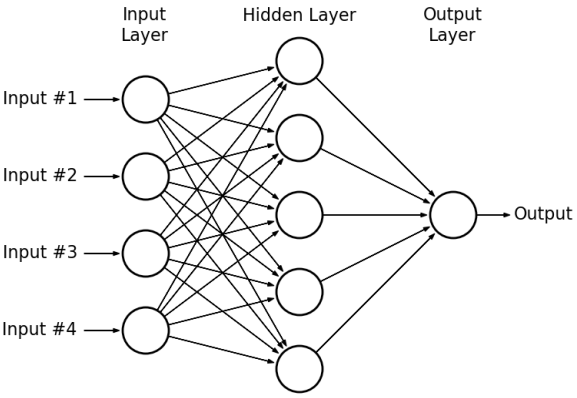
A neural network attemps to mimic the parallel architecture of the human brain. The human brain consists of a massive network of billions of simple neural cells that are interconnected and, given a stimulus, they each may "fire" or not.

A neuron
The neurons are the nodes of the network: they are decision points and can be viewed as a function that takes n input values and mutiplies them by a pre-defined weight (per input), adds a bias and then runs it through an activation function.



The network

A simple neural network has an input layer, a single hidden layer and an output layer (deep neural networks will have multiple hidden layers). The network is fully connected between layers: each node is connected to every node in the next layer up, no neurons are connected to other layers.



The exact setup of the network is largely problem dependent but there are some observations:

- The input layer has to correlate to the inputs: a data set with two input values means two input neurons
- If the neural network is used for a classification problem and you have a fixed number of classifications the output neurons would correlate to that.

The weights
The connection between each node is assigned a weight. Given that the end goal for the network is to learn from data the initial values for the weights do not matter too much, so they are assigned random values at start-up.

Training: back propagation
Backpropagation is a common method of training artificial neural networks used in conjunction with an optimization method such as gradient descent. The algorithm repeats a two phase cycle: propagation and weight update. Propagation (or feed forward) consists in feeding the randomly initialized network with a first training data set. As we are dealing with *supervised learning* we know the expected output for the dataset: this means that we can work out how far off the network is and hence attempt to adjust the weights to perform better.

Here come the second phase of the algorithm: the output of the network is compared to the desired output using a *loss function* and an error value is calculated for each of the neurons in the output layer. Backpropagation uses these error values to calculate the gradient of the loss function with respect to the weights of the network, the aim is to update the weights in order to minimize the loss function.

The importance of this process is that, as the network is trained, the neurons in the intermediate layers organize themselves in such a way that different neurons learn to recognize different characteristics of the total input space. After training, when an arbitrary input pattern is present, neurons in the hidden layer of the network will respond with an active output if the new input contains a pattern that resembles a feature that the individual neurons have learned to recognize during their training.

Algorithm

Phase 1: propagation

- Forward propagation of a training input through the neural network in order to generate the network's output
- Backward propagation of the outputs though the neural network in order to generate the deltas (the difference between the targeted and actual output values) of all output and hidden neurons

Phase 2: weight update

For each weight:

- The weight's output delta is used to find the gradient of the weight
- A ratio of the weight's gradient is subtracted from the weight

This ratio is called *learning rate* as it influences the speed and quality of learning. The greater the ratio the faster the neuron trains but the lower the ratio the more accurate the training is.

The sign of the gradient of a weight indicates whether the error varies directly with, or inversely to, the weight. Hence the weight must be updated in the opposite direction, "descending" the gradient.

Phases 1 and 2 are repeated until the performance of the network is satisfactory.

Derivation

Since backpropagation uses the gradient descent method, one needs to calculate the derivative of the squared error function with respect to the weights of the network.
Assuming one output neuron, the squared error function is:

$$E = \frac{1}{2}(t-y)^2$$

where t is the target output for a training sample and y is the actual output of the output neuron. The factor $\frac{1}{2}$ is included to cancel the exponent when differentiang later.

For each neuron j its output o_j is defined as

$$o_j := \varphi(net_j) = \varphi\left(\sum_{k=1}^{n} w_{k,j} o_k\right)$$

The input net_j to a neuron is the weighted sum of outputs o_k of previous neurons. The variable $w_{i,j}$ denotes the weight between neurons i and j.

The *activation function* φ is in general non-linear and differentiable. A commonly used activation function is the logistic function:

$$\varphi(z) = \frac{1}{1+e^{-z}}$$

which has derivative:

$$\frac{d\varphi}{dz}(z) = \varphi(z)(i - \varphi(z))$$

Calculating the partial derivative of the error with respect to a weight $w_{i,j}$ is done using the chain rule twice:

$$\frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{i,j}}$$

The last factor is straightforward to calculate:

$$\frac{\partial net_j}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}}\left(\sum_{k=1}^{n} w_{k,j} o_k\right) = o_i$$

Now let's assume that the logistic function is used as activation function, we have:

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial}{\partial net_j}\left(\varphi(net_j)\right) = \varphi(net_j)(1 - \varphi(net_j))$$

Finally the first factor is easy to evaluate if the neuron is in the output layer because then $o_j = y$ and

$$\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial y} = \frac{\partial}{\partial y}\left(\frac{1}{2}(t-y)^2\right) = y - t$$

However if j is in an arbitrary inner layer of the network we should consider E as a function of the inputs of all neurons $L = i, v, ..., w$ receiving input from neuron j

$$\frac{\partial E(o_j)}{\partial o_j} = \frac{\partial E(net, net_v, ..., net_w)}{\partial o_j}$$

Taking the total derivative with respect to o_j it is possible to find a recursive expression for the derivative:

$$\frac{\partial E}{\partial o_j} = \sum_{l \in L}\left(\frac{\partial E}{\partial net_l}\frac{\partial net_l}{\partial o_j}\right) = \sum_{l \in L}\left(\frac{\partial E}{\partial o_l}\frac{\partial o_l}{\partial net_l} w_{j,l}\right)$$

Therefore the derivative with respect to o_j can be calculated if all derivatives with respect to the outputs o_l of the next layer are known.

Lets define δ_j as follows:

$$\delta_j := \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j} = \begin{cases} (o_j - t_j) o_j (1 - o_j) & j \in O \\ \left(\sum_{l \in L} \delta_l w_{j,l}\right) o_j (1 - o_j) & j \in I \end{cases}$$

where O and I are the sets of respectively output and inner neurons.

Putting everything together we have:

$$\frac{\partial E}{\partial w_{i,j}} = \delta_j o_i$$

To update the weight $w_{i,j}$ using gradient descent, one must choose a learning rate α. The change in weight, which is added to the old weight, is equal to the product of the learning rate and the gradient, multiplied by -1 (in order to update in direction of a minimum).

$$\Delta w_{i,j} := -\alpha \frac{\partial E}{\partial w_{i,j}} = \begin{cases} -\alpha (o_j - t_j) o_j (1 - o_j) & j \in O \\ -\alpha \left(\sum_{l \in L} \delta_l w_{j,l}\right) o_j (1 - o_j) & j \in I \end{cases}$$

B. Distinguishing n clicks

Our goal is to distinguish different clicks using a neural network as classifier. The network must be first trained over a certain set of known clicks. Let's consider for clarity just a subset of characters of the keyboard: *a,b,c,d*. The purpose of the neural network is to assign to a given unknown character *x* four values corresponding to the probability of *x* of being respectively *a,b,c* or *d*.

The training phase

- Record n clicks for *a,b,c,d*
- For each click extract the features from the push peak of the click
- Provide all the features as *input*
- Specify the number of classes for the output by providing a so called *target* to the network.

Let's consider an example with $n = 2$ for clarity. The *input* for the network is:

$$f(a_1), f(a_2), f(b_1), f(b_2), f(c_1), f(c_2), f(d_1), f(d_2)$$

where $f(a_1)$ is the features extracted from the push peak of the first sample for click *a*. The network needs to know the number of possible classes for the output, in this case 4 classes corresponding to *a,b,c,d*, and to which class each element of

the input must be assigned. All this information is stored in the *target*. Let's see how the target looks like for the current example:

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

The rows correspond to the classes while the columns to the i^{th} element of the input: for instance $f(a_1)$ and $f(a_2)$ are mapped to the first class which means character *a*, $f(b_1), f(b_2)$ are mapped to the second class meaning character *b* etc.

Remark 1. *In the real application the number of samples would be far higher than 2, ie.* $n = 100$*, and the number of classes would be at least 36 (26 letters of the alphabet and 10 digits).*

C. Testing the network

After the training phase the network must be tested to know if its results are correct up to a satisfying percentage. In order to do so a second set of *sampling* clicks is recorded, for each click the features are extracted and given as input to the network. The network's result is compared with the identity of the key given in input.

V. Results

During my investigation I trained the neural network with two different keyboards: one integrated in a Dell laptop, with thinner keys, and a standard desktop one by Trust.
A certain key is recognized if the output node corresponding to this key has largest value when the feature corresponding to the key is provided as input to the network. Obtaining good results was not a trivial tasks since there are many factors that affect the network's prediction: precision in extracting sound clicks, recordings quality, features extraction etc. Such factors are related to the input fed to the network however one should also take into account the configuration of the network, eg. number of hidden nodes, learning rate, etc. Table I reports the results for both keyboards. Take for instance the value of the first cell relative to recognition of a on the Dell keyboard: (7,1,2). It means that over 10 clicks that the network was asked to recognized, 7 were correctly recognized, 1 was correctly recognized as second choice (as with second greatest probability) and 2 as third choice. It is interesting to note how the two keyboards produce different features: for instance character m is perfectly recognized on the Dell keyboard respect to the Trust one. Viceversa for character f.

Key	Recognized (D)	Recognized (T)
a	7, 1, 2	9, 0, 1
b	4, 5, 0	5, 3, 1
c	10, 0, 0	5, 2, 1
d	5, 2, 0	3, 4, 2
e	7, 2, 0	5, 2, 1
f	3, 3, 1	8, 0, 1
g	7, 0, 0	7, 1, 1
h	7, 1, 0	6, 2, 0
i	7, 1, 1	8, 2, 0
j	9, 1, 0	7, 2, 0
k	7, 1, 0	6, 1, 0
l	7, 1, 2	7, 2, 1
m	10, 0, 0	4, 5, 0
n	8, 1, 1	3, 1, 1
o	8, 1, 0	5, 2, 0
p	8, 2, 0	5, 1, 1
q	7, 2, 0	6, 1, 2
r	5, 0, 0	6, 3, 0
s	9, 0, 0	5, 3, 0
t	7, 2, 0	4, 3, 1
u	7, 0, 2	6, 3, 0
v	6, 4, 0	6, 1, 0
w	5, 1, 1	7, 1, 0
x	7, 1, 0	8, 2, 0
y	7, 1, 1	6, 2, 0
z	8, 1, 0	3, 3, 0

TABLE I. Results for Dell keyboard (D) and Trust keyboard (T). Values in a cell are the number of times the key was correctly recognized as first second or third choice respectively for a total of 10 samples.

Remark 2. *These values are explicative but not absolute: running the same code twice will not yield the same results since the neural network is initialized with random weights.*

A. Error Classification

When dealing with neural networks it is important to evaluate and improve their prediction capability. A good starting point for this task is to look at the network's errors, which I divide in three classes, in descending order of gravity:

I type: the network predicts a wrong result with high confidence

II type: the network predicts a wrong result but the second choice is the right one

III type: the network predicts the right result but with low confidence. Despite not being a real error such behaviour is risky since the network could predict wrong values when being fed noisy input

Table II reports prediction values for ten samples: the value in cell i, j is the probability that sample i is character j. In this example the correct character is "f" and just the two greatest values are reported in each row for better readibility. The network's prediction is not satisfying: we can see a I type error for samples $1, 5, 8, 10$ and III type error for sample 2. Given the set of characters *a,b,c,d,e,f* the network wrongly predicts just c or e instead of f. It is interesting to note that such keys are close to each other on a QWERTY keyboard. Is it possible then to discriminate every key with a satisfactory probability value? Asonov and Agrawal's answer is positive but neither code or recordings from their research have been published. I tried to address this issue by analyzing the problem one step ahead of the neural network. If the network fails to recognize a key for another, maybe it is not the network's fault but the two keys produce features too similar to each other. I first plotted the features of two keys, f and c for which the network prediction was wrong and compared them with a pair, f and a, for which the network prediction was excellent. Despite being a naive approach Fig. 7 clearly shows that features for keys correctly detected do differ.
Of course this visual intuition had to be formalized with an

Sample	a	b	c	d	e	f	g
1			0.54			0.15	
2					0.39	0.46	
3			0.29			0.53	
4			0.46		0.30		
5			0.95			0.04	
6					0.20	0.59	
7			0.20			0.67	
8			0.75			0.17	
9			0.18			0.61	
10			1.00	0.00			

TABLE II. Network's prediction for 10 samples: the value in cell i, j is the probability that sample i is character j. For readibility just the two greatest values are reported for each row.

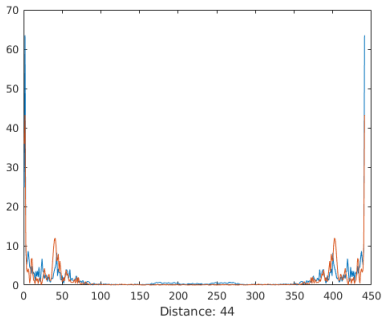


Fig. 6. Features of keys c and f, visually similar and with low distance (as obtained by applying Dynamic Time Warping)

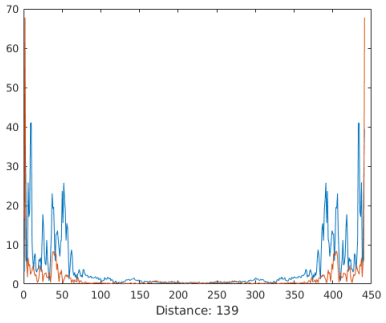


Fig. 7. Features of keys a and f, visually different and with high distance (as obtained by applying Dynamic Time Warping)

approapriate metric. For such purpose I used Dynamic Time Warping.

B. Dynamic Time Warping

Consider the problem of eliminating timing differences between two time series A and B

$$A = a_1, a_2, \cdots, a_i, \cdots, a_I$$

$$B = b_1, b_2, \cdots, b_j, \cdots, b_J$$

Let us consider an $i-j$ plane, shown in Fig. 8 where patterns A and B are developed along the $i-$axis and $j-$axis respectively. Let us consider the sequence

$$F = c_1, c_2, \cdots, c_k, \cdots, c_K$$

where $c_k = (i_k, j_k)$. This sequence can be considered to represent a function which approximately realizes a mapping from from the time axis of pattern A onto that of pattern B. Hereafter it is called a warping function. When there is no timing difference between these patterns, the warping function coincides with the diagonal line $j = i$. It deviates further from the diagonal line as the timing difference grows. As a measure of the difference between two vectors a_i and b_j a distance $d(c) = d(i, j) = ||a_i - b_j||$ is employed between them. Function F is subject to some restrictions:

1) Monotonicity:

$$i_{k-1} \leq i_k \text{ and } j_{k-1} \leq j_k$$

2) Continuity conditions:

$$i_k \leq i_{k-1} \text{ and } j_k \leq j_{k-1}$$

3) Boundary conditions:

$$i_1 = 1, j_1 = 1 \text{ and } i_K = I, j_K = J$$

4) Adjustment window condition (see Fig. 8):

$$|i_k - j_k| \leq r$$

where r is an appropriate positive integer called window length. This condition corresponds to the fact that time-axis fluctuation in usual cases never causes a too excessive timing difference.

As a result of conditions 1) and 2) the following relation holds between two consecutive points:

$$c_{k-1} = \begin{cases} (i_k, j_k - 1) \\ (i_k - 1, j_k - 1) \\ (i_k - 1, j_k) \end{cases}$$

If one is not interested in optimization then the pseudo code for Dynamic Time Warping is very simple:

$D \leftarrow array[0..n, 0..m]$
for $i \in 1..n$ **do**
  $D[i, 0] \leftarrow \infty$
for $i \in 1..m$ **do**
  $D[0, i] \leftarrow \infty$
$D[0, 0] \leftarrow 0$
for $i \in 1..n$ **do**
  for $j \in 1..n$ **do**
    $D[i, j] \leftarrow |A[i] - B[j]| + min(D[i-1, j],$
    $D[i, j-1],$
    $D[i-1, j-1])$
return $D[n, m]$

VI. Related Work

Asonov and Agrawal pioneered a new generation of keyloggers. The year after their publication Li Zhuang, Feng Zhou, and J. D. Tygar published a revisited version of the attack [2]. The novelty of their work lies in the fact that there is no need for training recordings labeled with the corresponding clear text. This is possble by using statistical constraints of the underlying content, English language. However their approach

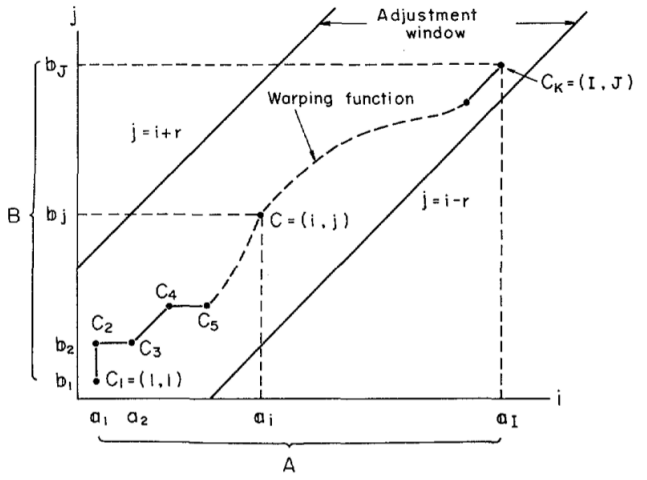


Fig. 8. Warping function and adjustment window definition

requires a 10 minutes sound recording of a user typing English text.
Another really interesting and recent (2017) research work is to be found in [3]. The proposed attack is called *Skype & Type (S & T)* and is based on acoustic eavesdropping on Voice-over-IP. S&T is motivated by the simple observation that people often engage in secondary activities (including typing) while partecipating in VoIP calls. The attack is possible since VoIP software (specifically Skype in the research work) conveys enough audio's information to reconstruct the victim's input.
Moreover S&T relaxes the strong adversary models required by previous attacks, ie. precise profiling of the victim's typing style and keyboard [1] and significant amount of victim's typed information [2].

VII. Conclusions

The work presented in this paper points to many avenues for further research. One could employ a classifier different from neural networks, such as a support vector machine or a decision tree, experiment with MFCC instead of FFT for feature extraction, test different keyboards with multiple users and typing styles.
However what I find matters most is the awereness that keyboard acoustic emanations are a threat to take into account in security critical scenarios. A countermeasure would be typing on a rubber keyboard, despite inconvenient to carry around, or in alternative using a virtual keyboard when entering credentials (such security measure is enforced by Westpac's online banking services as well as TreasuryDirect).

This project can not be seen as a complete automatic attack tool, rather as an educational proof of concept. However I am glad to have achieved satisfying results in terms of network recognition and to have laid out an open source code basis which, to the best of my knowledge, was not available neither for Ason and Agrawal's research nor for related work. I hope that this work will be useful to other people interested in the topic, the whole project can be found on github.com under "Keyboard Acoustic Emanations".

References

[1] Asonov, D. and Agrawal, R. 2004. "Keyboard Acoustic Emanations". *Proceedings of the IEEE Symposium on Security and Privacy.* 3–11.

[2] Li Zhuang, Feng Zhou, and J. D. Tygar. Keyboard acoustic emanations revisited. *In CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, pages 373–382, New York, NY, USA, 2005. ACM.

[3] Compagno, Alberto, Mauro Conti, Daniele Lain, and Gene Tsudik. "Don't Skype & Type!" *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security* - ASIA CCS '17 (2017): n. pag. Web.

[4] R. J. Anderson and M. G. Kuhn. Soft tempest - an opportunity for NATO. In *Proceedings of Protecting NATO Information Systems in the 21st Century, IST Symposium, Washington DC, USA*, Oct. 1999

[5] M. G. Kuhn. Optical time-domain eavesdropping risks of CRT displays. In *Proceedings of IEEE Symposium on Security and Privacy, Berkley, California, USA*, May 2002.

[6] J. Loughry and D. A. Umphress. Information leakage from optical emanations. *ACM Transactions on Information and System Security*, 5(3):262-289, Aug. 2002

[7] Sakoe, Hiroaki; Chiba, Seibi. "Dynamic programming algorithm optimization for spoken word recognition". *IEEE Transactions on Acoustics, Speech and Signal Processing*. 26 (1): 43-49.

[8] Wikipedia, Backpropagation
`https://en.wikipedia.org/wiki/Backpropagation`