ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Exploration of Audio Similarity Techniques to Improve the Accuracy of Sound-Proof

Semester Project

January 11, 2016

Martina Rivizzigno

Prof. Dr. Srdjan Capkun
Supervised by Nikolaos Karapanos and Claudio Marfario

Contents

Abstract

Sound-Proof is a two factor authentication mechanism that aims to improve usability by eliminating user interaction in providing the second factor. The second authentication factor is the proximity of the users phone to the device used to log in. Proximity is verified by comparing the ambient sound recorded automatically by the two devices. To compare the audio samples that are captured during each login attempt, Sound-Proof employs an algorithm which is based on the combination of pass-band filtering and cross-correlation of the time-based representation of audio signals. The algorithm computes the similarity score of the two samples. Given a predefined threshold, this value is used to determine if the login attempt is legitimate or not.

The goal of this semester project is to explore alternative audio comparison techniques that could potentially improve the accuracy of the algorithm used in Sound-Proof. We researched several audio comparison methods coming from music recognition applications. We then implemented three different algorithms and evaluated them using the same dataset which was used to evaluate the original Sound-Proof algorithm. Our evaluation results show that none of the alternative approaches that we explored improves the accuracy of the current implementation of Sound-Proof.

Chapter 1

Introduction

Sound-Proof [1] is a two factor authentication mechanism based on ambient noise. Two factor authentication protects online accounts if passwords are leaked because user identification is the combination of two different components. Usually such mechanisms require the user to interact with his phone. For example Google 2-Steps Verification asks the user to provide his credentials as usual. Then, a code is sent to the user's phone via SMS, voice call or Google mobile app. The code must be copied in the browser to successfully log in. Otherwise the user can insert a Security Key into his computer's USB port which requires him to carry around some additional hardware.

Sound-Proof improves two factor authentication usability by eliminating the user-phone interaction. When a user wants to login to a service, he provides username and password as first factor. If this first authentication step is successful, the second authentication factor is the proximity of the user's phone to the computer being used to log in. The user's phone must be previously paired to his credential. The two devices, the computer browser and the phone, will automatically start recording the ambient sound for five seconds. The phone compares the two recordings to determine if it is in the same environment of the computer. If it is, the ambient sound will be the same, and the user is successfully authenticated. Otherwise the login attempt is rejected as malicious.

The current version of Sound-Proof compares audio samples using cross-correlation with an achieved Equal Error Rate of 0.002. The goal of the semester project is to find an alternative audio comparison method that guarantees higher accuracy.

We research several algorithms from the literature for music recognition systems such as Shazam. We then restrict our focus on three methods which we have implemented using MATLAB.
The first one is a Highly Robust Audio Fingerprinting System [2]. The

algorithm compares audio signals by looking at the difference of the energy content of subsequent frequency bands.
Waveprint: Efficient Wavelet-Based Audio Fingerprinting [3] transforms sound comparison into image comparison by working on the audio signal spectrogram. Images are processed using the wavelet transformation and compared using locality sensitive hashing.
The third algorithm we implement, Moving Average, does not come from the literature but it is our own idea. The audio similarity is computed using cross-correlation like Sound-Proof does. The difference is that the cross-correlation is not computed on the raw sample but on the sample moving average.

The report is structured in the following way: Chapter 2 explains tools and methods we are using in the algorithms and it describes the current implementation of Sound-Proof; Chapter 3 sums up the research work and gives an overview of the approaches we decide not to further investigate and implement; Chapter 4 describes the implemented algorithm and analyses the result we got from each one of them; Chapter 5 draws the conclusions on the semester project and hints at possible future work such as algorithm improvement and other research directions.

Chapter 2

Background

2.1 Sound Comparison

In the following sections we describe tools and methods, used to compare audio signals during the semester project.

2.1.1 Cross-Correlation

Given two discrete time sequences, x and y, the cross-correlation measures the similarity between x and a shifted copy of y as a function of the lag, l:

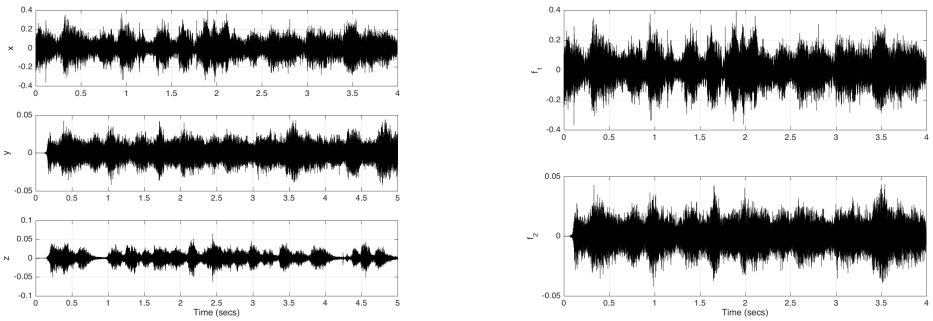
$$c_{x,y}(l) = \sum_{i=1}^{n} x(i) \cdot y(i-l)$$

For example, considering 3 audio signals: two of which are the same audio shifted, x and y, and the third one, z, is a completely different sound. By plotting their time series it is not possible to determine which ones are the matching samples (Figure 2.1a). In order to determine if a signal is present into another, pairs of signals are cross-correlated. The peak in the cross-correlation implies that x is present in y starting after 0.0543s as described in Figure 2.2. Signal x leads y by $fs \times 0.0543 = 2394$ samples, where $fs = 44100\frac{samples}{second}$ is the sampling frequency. In the other plot in Figure 2.2 there is no peak, meaning that the signals are not correlated.

The cross correlation can be normalize as

$$c^*_{x,y}(l) = \frac{c_{x,y}(l)}{\sqrt{c_{x,x}(0)c_{y,y}(0)}}$$

where $c_{x,x}$ and $c_{y,y}$ are the auto-correlation of the two series at 0. c^* takes value between -1 and 1. If $c^*(l) = 1$ the two signals have the same shape, if



(a) Time Series before alignment

(b) Time Series after alignment of matching files

Figure 2.1: Cross-correlation to study signal similarity

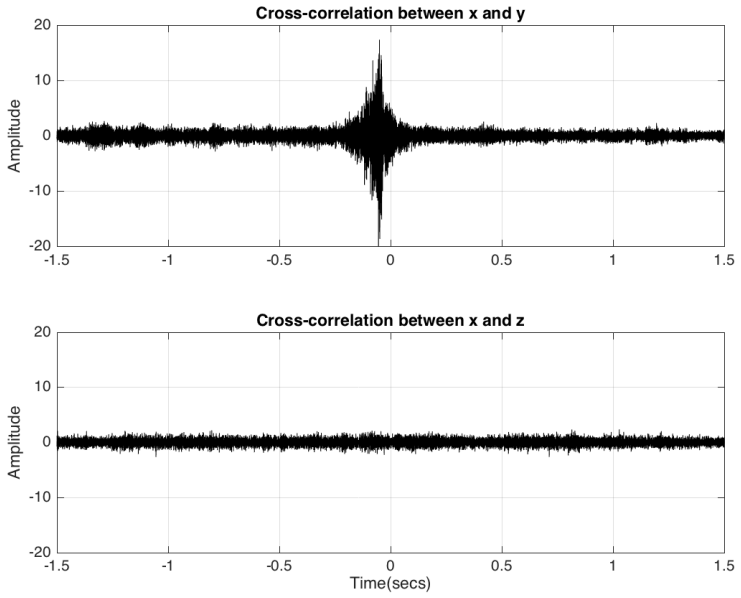


Figure 2.2: Cross-correlation peak

$c^*(l) = -1$ the shape is the same but the sign is opposite. $c^*(l) = 0$ the two signals are uncorrelated. Continuing with the previous example, the peak of the normalized cross-correlation is 0.2045 for signals x and y. For x and z is close to zero, 0.0242 (Figure 2.3).

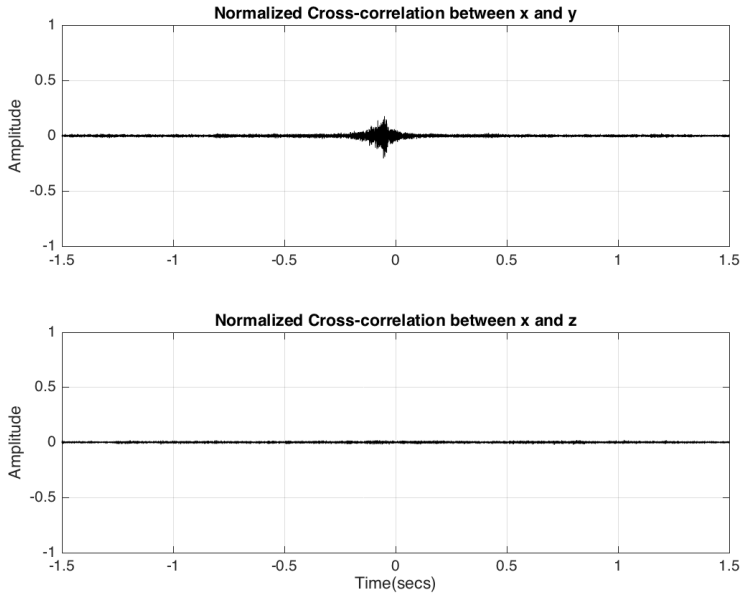


Figure 2.3: Normalized Cross-correlation peak

2.1.2 One-Third Octave Bands

Sound spectrum is usually represented in octave frequency bands to more naturally group frequencies of audio signals as they are perceived by the human hear. The audible frequency range, from 20Hz to 20kHz, is divided in 11 non overlapping logarithmically spaced octaves. Each one is represented by its central frequency which is twice as much as the one of the previous band.
To increase resolution the octave bands are broken into 32 one-third octave bands. Starting from the reference frequency, $f_{centr} = 1000Hz$. The upper band frequency f_{high} and the lower, f_{low} are related as $f_{high} = 2^{1/6} f_{centr}$ and $f_{low} = 2^{-1/6} f_{centr}$. The central frequency, f_{centr} is the geometric mean of the upper and lower frequency limits. The fractional bandwidth per band is constant, $BW = \frac{100*(f_{high} - f_{low})}{f_{centr}} = 23.2\%$.
One third octave bands are widely used in acoustics so their frequency ranges have been standardized [4]. Their frequencies are reported in Table 2.1.

2.1.3 Wavelets

Wavelets are a mathematical tool for hierarchically decomposing functions. In this case the function of interest is an image, the spectrogram.

Band	f_{low}	f_{centr}	f_{high}	*Band*	f_{low}	f_{centr}	f_{high}
1	14.1	16	17.8	17	562	630	708
2	17.8	20	22.4	18	708	800	891
3	22.4	25	28.2	19	891	1000	1122
4	28.2	31.5	35.5	20	1122	1250	1413
5	35.5	40	44.7	21	1413	1600	1778
6	44.7	50	56.2	22	1778	2000	2239
7	56.2	63	70.8	23	2239	2500	2818
8	70.8	80	89.1	24	2818	3150	3548
9	89.1	100	112	25	3548	4000	4467
10	112	125	141	26	4467	5000	5623
11	141	160	178	27	5623	6300	7079
12	178	200	224	28	7079	8000	8913
13	224	250	282	29	8913	10000	11220
14	282	315	355	30	11220	12500	14130
15	355	400	447	31	14130	1600	17780
16	447	500	562	32	17780	20000	22390

Table 2.1: Standard One Third Octave Band frequencies

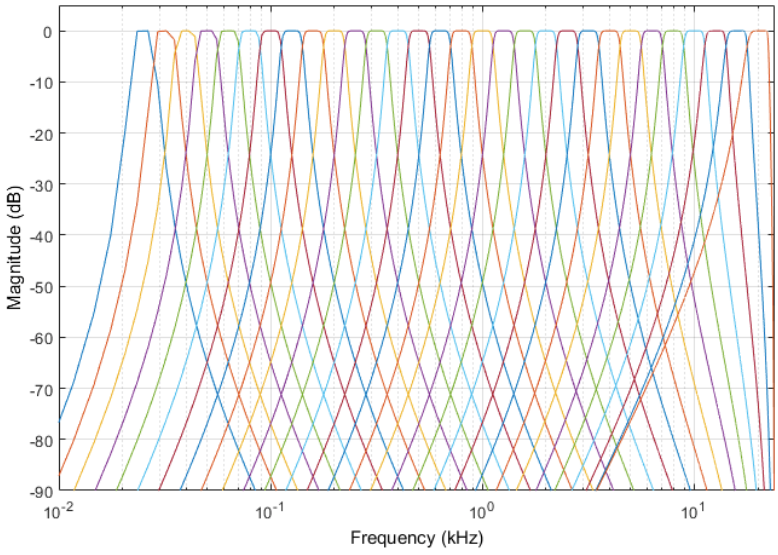


Figure 2.4: One third octave band filter bank

If we consider a one dimensional image composed of 4 pixels having the values

$$\begin{bmatrix} 3 & 1 & 7 & 9 \end{bmatrix}$$

the wavelet transform is computed by averaging the pixels pairwise to get a new lower resolution image

$$\begin{bmatrix} 2 & 8 \end{bmatrix}$$

In order not to loose information in the averaging process, detail coefficients are stored. The coefficients enables to recover the original four pixel values from the average. In the example the first detail coefficient is 1 because 2 is 1 less than 3 and 1 more than 1, $2 + (+1) = 3$ and $2 - (+1) = 1$. The second one is -1 because $8 + (-1) = 7$ and $8 - (-1) = 9$. Iterating the described process, the full decomposition is the following:

Resolution	Averages	Detail Coefficients
4	[3 1 7 9]	
2	[2 8]	[1 -1]
1	[5]	[-3]

The wavelet transform of the four pixels image is the single coefficient representing the average of all the image pixels plus the detail coefficients in order of increasing resolution. In the example the wavelet decomposition is:

$$\begin{bmatrix} 5 & -3 & 1 & -1 \end{bmatrix}$$

In this process no information is gained or lost. The initial image has 4 coefficients and so does the transformation. Furthermore any level of resolution can be reconstructed by adding or subtracting the detail coefficients starting from the lowest resolution.
An image, in our case a spectrogram, has two dimensions. To obtain the standard decomposition, first the one dimensional decomposition is applied to every row of pixel values. The result is one average value and detail coefficients for each row. Then the one dimensional transformation is done on each column. The final result is one overall average coefficient. The non standard decomposition processes the rows and the columns alternatively. First one transformation of each row is done, then one for each column. The process is repeated in all the quadrants containing averages in both directions.

2.1.4 Min-Hash

Min-hash [5] is a locality sensitive hashing scheme used to estimate how similar two data sets are. It uses a randomized algorithm to estimate the Jaccard similarity of sets.
The Jaccard similarity of sets S and T is $\frac{|S \cap T|}{|S \cup T|}$, that is, the ratio of the size of the intersection of S and T to the size of their union. The Jaccard similarity of S and

T is denoted by SIM(S, T). For example, considering S1 and S3 from Table 2.2 the Jaccard similarity is:

$$SIM(S1, S3) = \frac{|S1 \cap S3|}{|S1 \cup S3|} = \frac{|\{2\}|}{|\{1,2,3,4,5,6\}|} = \frac{1}{6}$$

Element	*S1*	*S2*	*S3*	*S4*
1	1	0	0	1
2	1	0	1	0
3	0	1	1	0
4	0	0	1	1
5	1	0	0	0
6	0	0	1	1

Table 2.2: Binary matrix representation of sets: S1={1,2,5}, S2={3}, S3={2,3,4,6}, S4={1,4,6}

Sets S1={1,2,5}, S2={3}, S3={2,3,4,6}, and S4={1,4,6} are binary represented in a matrix. The rows are randomly permuted as shown in Table 2.3 and the position of the first one in recorded. So by denoting the Min-hash function, $h(\cdot)$, the result is $h(S1) = 2$, $h(S2) = 3$, $h(S3) = 2$, $h(S4) = 6$.

Element	*S1*	*S2*	*S3*	*S4*
2	1	0	1	0
5	1	0	0	0
6	0	0	1	1
1	1	0	0	1
4	0	0	1	1
3	0	1	1	0

Table 2.3: Permutation of rows

The probability that the min-hash function for a random permutation of rows produces the same value for two sets equals the Jaccard similarity of those sets. By observing the columns for two sets, for example S1 and S2, the rows can be divided in three classes:

- class X: rows have 1 in both columns
- class Y: rows have 1 in one of the columns and 0 in the other
- class Z: rows have 0 in both columns

If the matrix is sparse the majority of rows are of class Z. However the ratio of the number of rows of class X and Y is what determines the similarity and hence the probability that h(S1)=h(S2).
If x is the number of rows of class X and y the number of class Y, the similarity is:

$$SIM(S1, S2) = \frac{x}{x+y}$$

because x is the size of the intersection between S1 and S2 and x+y is the size of the union.
Starting from the top row randomly permuted, the probability of encountering a class X before class Y row is $\frac{x}{x+y}$. So if the first row from the top belongs to class X, not considering rows of class Z, then for sure h(S1)=h(S2). If the first row other than class Z belongs to class Y, then the set with the one gets that row as min-hash value, while set with a zeros will get some row further down. Thus, h(S1) $\neq$ h(s2) if the first row encountered is of class Y.
Concluding the probability that h(S1)=h(S2) is $\frac{x}{x+y}$ which is also the Jaccard Similarity of sets S1 and S2.

2.1.5 Moving Average

The moving average is a succession of averages derived from successive segments of a series of values, typically of constant size and overlapping. Given a series of numbers and a fixed window, the first element of the moving average is calculated as the mean of the initial fixed subset. Then the window is shifted forward until the second window has the desired overlap with the first one. The new subset is averaged. The process is repeated iteratively over the all set.
Moving average is commonly used with time series data such as stock prices and trading volumes to smooth out fluctuations. Mathematically, it can be seen as a convolution:

$$y[i] = \frac{1}{M} \sum_{j=0}^{M-1} x[i+j]$$

where $x[\cdot]$ is the input signal, $y[\cdot]$ is the output signal, and M is the number of points in the average. The moving average filter is a convolution of the input signal with a rectangular pulse having an area of one.

2.2 Sound-Proof

The audio comparison method used by Sound-Proof is based on the cross-correlation technique.

Each audio signal, x, is input in a one-third octave band filter bank to obtain one signal component per band, x_i. Only 20 bands, from 50Hz to 4000Hz, from the

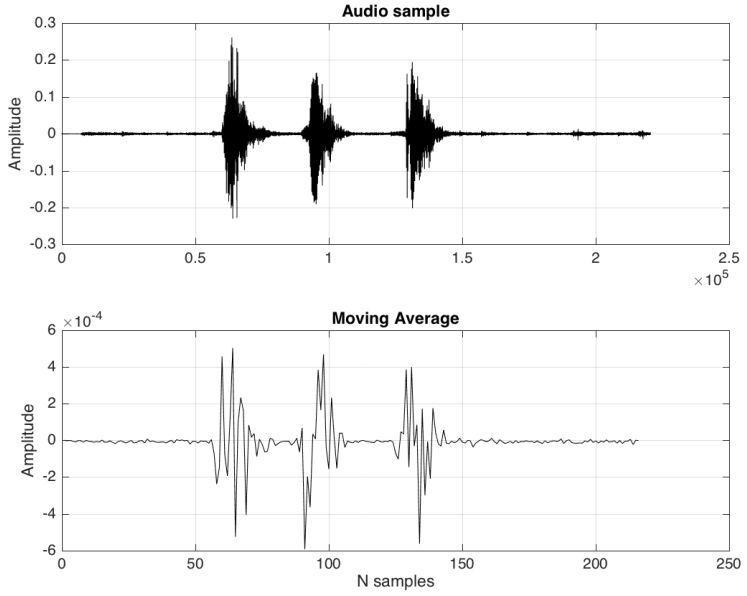


Figure 2.5: Smoothing effect of the moving average on an audio sample

standard ones are considered. Worst results are achieved with the full set of frequencies. The similarity score is computed as the average in the considered bands of the maximum normalized cross-correlation between pairs of signal components, x_i and y_i:

$$S_{x_i,y_i} = \frac{1}{n}\sum_{i=1}^{n} c_{x_i,y_i}(l)$$

where l is bounded between 0 and $l_{max} = 150ms$. The similarity score between two audios captured in a login attempt is compared against a predefined threshold to determine if the login attempt is legitimate or not.

The samples to test the algorithm were recorded in six different settings: a office without ambient noise, labeled as *Office*, or with a computer playing music, *LaptopMusic*; a living-room with the TV on, *HomeTV*; a classroom during a lecture, *Lecture*; a train station, *TrainStation*; a cafe, *Cafe*. The user could be silent, talking, coughing, or whistling. In the Office environment the user cannot be silent because otherwise there is not enough background noise to perform the comparison. On the contrary, in the Lecture environment the user can only be silent since all the other activities are not allowed in a classroom. In the Cafe environment whistling is not tested since it could be an awkward activity for some users. The phone could be on a table or on a bench near the user, in the trouser pocket or in a purse. In total 4101 login attempts, corresponding to 8202 audio samples.

To evaluate the algorithm two parameters are taken into account: the false

rejection rate (FRR) and the false acceptance rate (FAR). When a legitimate login is rejected a false rejection occurs. On the contrary, a false acceptance occurs when a fraudulent login is successful. The achieved Equal Error Rate is 0.0020 at the threshold 0.13 (Figure 2.6).

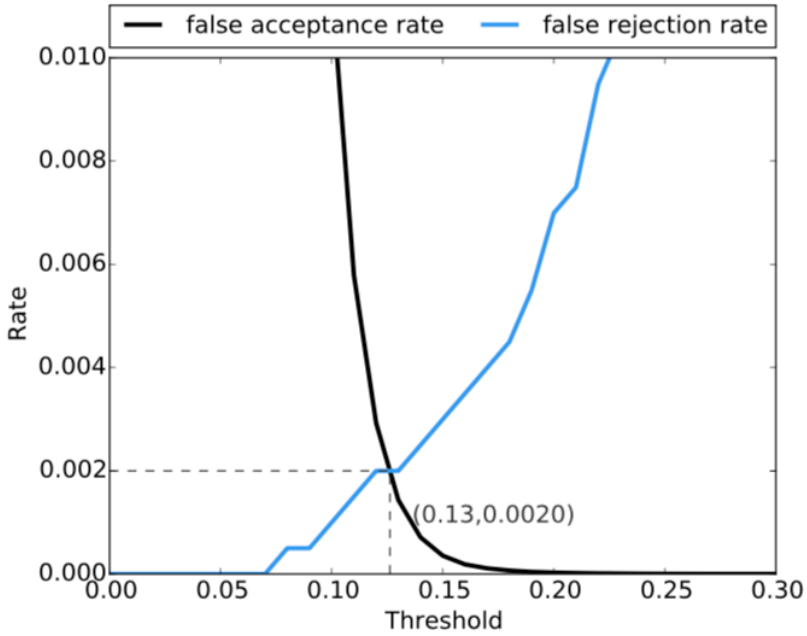


Figure 2.6: Sound-Proof False Rejection Rate and False Acceptance Rate as a function of the threshold, T. The Equal Error Rate is 0.0020 at T=0.13

The performance does not vary significantly in relation to different environments as Table 2.4 shown. The are not false rejections in the LaptopMusic, Lecture and HomeTV environments. The FAR is 0.003 in both the Office and TrainStation environments with 1 rejection over 310 and 1 over 330 respectively. The worst performing environment is the Cafe with 2 rejections over 338 attempts, yielding a 0.006 FRR.

The False Rejection Rate is then analyzed based on the user activity. When the user makes some noise the accuracy increases. In fact the FRR is 0.005 (3 logins over 579) when the user is silent, 0.002 (1 logins over 529) when the user is coughing. There are not rejections when the user is speaking or whistling.

The phone position is then taken into consideration. The FRR is 0.001 (1 logins over 667) when the phone is outside on a table or on a bench, 0.001 (1 logins over 675) when it is in the pocket, and 0.003 (2 logins over 660) when it is in a purse.

The False Acceptance Rate is analyzed when the two non matching audio come from the same environment (Table 2.5). The FAR is low for all environments:

Environment	*rejection*	*login*	*FRR*
LaptopMusic	0	432	0
Lecture	0	122	0
HomeTV	0	430	0
Office	1	310	0.003
TrainStation	1	330	0.003
Cafe	2	338	0.006

Table 2.4: Sound-Proof False Rejection Rate based on the environment

0.001 for the Lecture, Cafe, and TrainStation environments with 8 malicious logins accepted over 7242, 32 over 566994, and 44 over 67098 respectively, 0.003 (311 logins over 90992) in the HomeTV environment, 0.012 (1063 logins over 91960) for LaptopMusic, and 0.025 (1194 logins over 47250) for Office.

Environment	*acceptance*	*login*	*FAR*
LaptopMusic	1063	91960	0.012
Lecture	8	7242	0.001
HomeTV	311	90992	0.003
Office	1194	47250	0.025
TrainStation	44	67098	0.001
Cafe	32	56994	0.001

Table 2.5: Sound-Proof False Acceptance Rate in similar environments

Chapter 3

Related Work

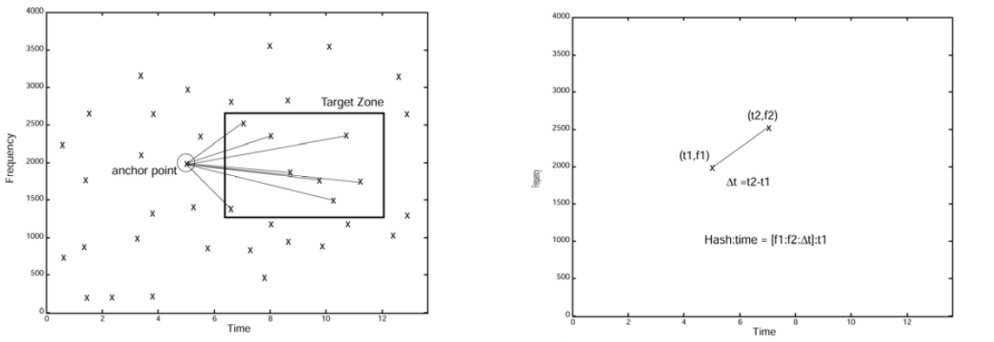
The starting point of project is finding literature on audio comparison. There is a wide literature on song retrieval services, similar to the well known Shazam. A user hears a song playing, records a small snippet with his phone and the app is able to match the sample to the song it belongs to. The user is provided with the title, the author and the album of the song.

We have performed the research keeping in mind the different specification of the problem. A music retrieval application has to search for a song in a database of around 11 millions of songs. Furthermore a small portion of the song, in the order of seconds, has to be compared against the full length original which is on average 4 minutes long. On the other hand, Sound-Proof works with two temporally aligned samples that are 5 seconds long at most. All the efforts to be computationally and memory efficient done by the analyzed methods are not a primary concern and are not taken into consideration. Another difference is that music is a continuum, characterized by a tempo and a number of beats per minute for example, while ambient noise includes moments of silence. Music can be described relying on human audible frequencies ranging from 300Hz to 2000Hz while to capture background noise in everyday environments the range of frequencies must be enlarged.

We have taken into consideration the following papers:

- A Highly Robust Audio Fingerprinting System [2]
- Waveprint: Efficient Wavelet-Based Audio Fingerprinting [3]
- An Industrial-Strength Audio Search Algorithm [6]
- Computer Vision for Music Identification [7]
- Pairwise Boosted Audio Fingerprint [8]

We have implemented the first two algorithms and they are explained in details in Chapter 4. The third approach had been already implemented by the Phds who



(a) Hash genation (b) Hash details

Figure 3.1: Shazam database creation

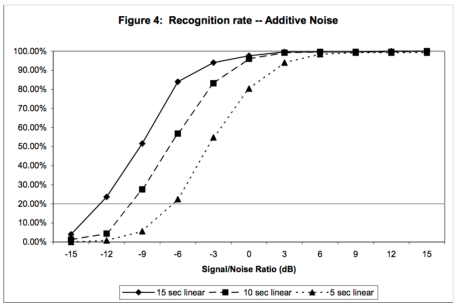
are working on the project and it did not improve the performance. Therefore it was discarded. Nonetheless it is described in section 3.1. We have not implemented the last two approaches therefore they are briefly described in sections 3.2 and 3.3.

3.1 An Industrial-Strength Audio Search Algorithm

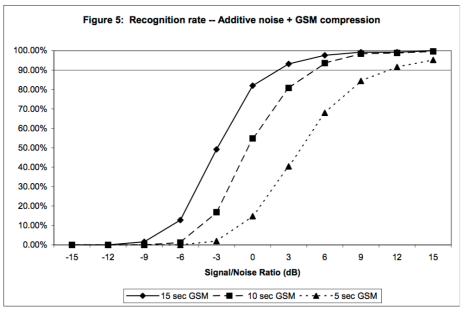
The audio file is transformed into a spectrogram. Time-frequency points which have higher energy than all their neighbors in a region centered around the point are selected. The higher amplitude these spectrogram peaks have the better it is because they are more likely to survive distortions. Once the spectrogram is reduced to a set of peaks, the amplitude is discarded. The resulting image is called constellation map. The pattern of dots in the constellation should be the same for matching audio fragments. Intuitively, sliding the constellation map of a song snippet over the all song constellation will have a significant number of points matching when the proper time offset is determined.
Anchor points and their associated target zone are chosen in the constellation (Figure 3.1a). Each anchor point is paired with all the points in its target zone yielding two frequency coordinates, f_1 and f_2, and the time difference between the points, Δt, as shown in Figure 3.1b. These values are hashed together and they are also associated with the time offset from the beginning of the file to the anchor point, t_1 and the track ID, $trackID$.

$$[f_1 : f_2 : \Delta t] : t_1 : trackID$$

When an audio snippet must be matched to a song in the database, for each hash match a time pair is created. A time pair consists of the time offset from the beginning of the sample and from the beginning of the database file. Pairs are



(a) Additive Noise



(b) Additive Noise and GSM compression

Figure 3.2: Shazam Recognition Rate

distributed into bins according to the track ID from the database hash. For each bin time pairs are plotted. Files are a match if there is a cluster of points forming a diagonal line in the plot. A straight line means that the relative offsets from the beginning of the clip are similar.

The algorithm performance is tested on 250 samples of different length and noise level against a database of 10,000 songs. Noise was recorded in a pub, normalized to the desired signal to noise ratio and linearly added to the sample. The same analysis is carried out when the music and noise mixture is also subjected to GSM 6.10 compression. The recognition rate is reported in Figure 3.2a and 3.2b respectively.

3.2 Computer Vision for Music Identification

The audio signal is converted into a spectrogram, a time-frequency image. It represents the power in 33 logarithmically spaced frequency bands using 372 ms long slices, taken every 11.6 ms. To distinguish different songs small sets of filters are applied to the spectrogram. The filters responses should be robust to distortions while preserving the information needed to distinguish different audio. Filters with large bandwidths and time widths are more robust to distortions but short bandwidth and time widths better capture discriminative information. A class of filters that meets the requirements is the Haar wavelet-like filters [9]. The filters are shown in Figure 3.3. To apply them the spectrogram is seen as a 2-D greyscale image. A filter sums the values of the pixels that lie in the white region and subtracts this value from the sum of pixels in the black region.

From the candidate set M discriminative filters and their thresholds are selected to create a M-bit vector that represents the audio. A single descriptor does not have enough information to match a query to a database of songs. Therefore descriptors are computed every 11.6ms.

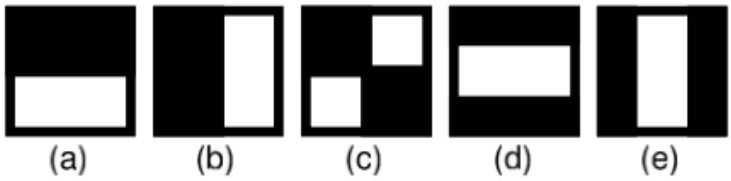


Figure 3.3: Candidate filter set designed by Viola and Jones for face recognition

To determine the filter set and the threshold a learning classifier is trained using AdaBoost. The classifier $H(x_1, x_2) \rightarrow y = \{\pm 1\}$ takes as an input two spectrogram images and labels them as +1 if both images derive from the same audio source. Otherwise it labels them -1. The classifier H is an ensemble of M weak classifiers, $h_m(x_1, x_2)$:

$$H(x_1, x_2) = sgn\Big(\sum_{i=1}^{M} c_m h_m(x_i, y_i)\Big)$$

where c_m is the weight assigned to each weak classifier which represents the confidence on its correctness. Each weak classifier is composed of a filter and its threshold t_m:

$$h_m(x_1, x_2) = sgn\big(f_m(x_1, x_2) - t_m)(f_m(x_2) - t_m)\big)$$

Samples are labeled as matching by the weak classifier if the filter response they generate is on the same side of the threshold.
Initially in AdaBoost all the n samples in the training set have the same weighs $w_i = \frac{1}{n}$. The weights are updated after the training of each weak classifier so that misclassified samples have increasing weights. This process enables the subsequent weak classifier to specialize in the most difficult to classify samples. On the contrary, for this application only matching pairs are re-weighted and the weights of matching and non matching pairs are normalized such that the sum of each is one half. The reason why the modification is necessary is that no weak classifier can perform better than chance on the non matching pairs.

Pairwise Boosting Algorithm

---

input: n samples $\{(x_{11}, x_{21})..(x_{1n}, x_{2n})\}$ and their labels $y_i \in \{\pm 1\}$
initialize: $w_i = \frac{1}{n}, i = 1, ..., n$
for m=1:M

1. find $h_m(x_1, x_2)$, where $h_m(x_1, x_2) = sgn\big(f_m(x_1, x_2) - t_m)(f_m(x_2) - t_m)\big)$, that minimizes the weighted error

2. calculate the weighted error $err_m = \sum_{i=1}^{n} w_i \cdot \delta(h_m(x_{1i}, x_{2i}) \neq y_i)$
3. assign confidence to h_m, $c_m = log(\frac{1-err_m}{err_m})$
4. update weights for matching pairs: if $y_i = 1$ and $h_m(x_{1i}, x_{2i}) \neq y_i$ then $w_i \leftarrow w_i \cdot exp(c_m)$
5. normalize weights: $\sum_{i:y_i=-1}^{n} w_i = \sum_{i:y_i=1}^{n} w_i = \frac{1}{n}$

output: $H(x_1, x_2) = sgn\Big(\sum_{i=1}^{M} c_m h_m(x_i, y_i)\Big)$

Given a query signature $x^r = (x_1^r, x_2^r, ..., x_n^r)$ composed of n descriptors, the matching song is the one maximizing the conditional probability $P(x^r|x^o)$, with $P(x^r|x^o) > T$, where x^o is the original snippet signature and T is a threshold.

The system achieved 90% recall at 96% precision when the audio probe was 10s long and recorded at low volume and with a distorted microphone. 80% recall at 93% precision was achieved with probes captured in very noisy environments, where music was barely audible to humans. The database used for both tests consisted of 1892 songs from 145 albums of different genres.

3.3 Pairwise Boosted Audio Fingerprint

An audio signal is divided into frames L_w long with an overlapping of L_s. Then each frame is divided into M critical bands and for each band the spectral centroid is computed. The result is an M-dimensional SSC vector describing each frame. N consecutive SSC vectors, a SSC image, create an $M \times N$ matrix x which is a binary fingerprint. The Viola and Jones filters (Figure 3.3) are applied to compute the difference in sum of the white (R_W) and black (R_B) regions of the SSC features in different time-frequency supports.

$$F(x) = \sum_{(m,n)\in R_W} x(m,n) - \sum_{(m,n)\in R_B} x(m,n)$$

The filter thresholds are selected from a candidate set as the ones that minimize the mean square quantization error of the filtered outputs of the training data. The quantizer, $Q_T(\cdot)$, is defined as:

$$Q_T(a) = \begin{cases} 0 & \text{if} \quad a < t_1 \\ j & \text{if} \quad t_j \leq t_{j+1} \\ T & \text{if} \quad t_T \leq a \end{cases}$$

Once the filters and their thresholds are selected, their outputs are quantized and transformed into a Grey encoded binary form using $B(\cdot)$. The pairwise boosting algorithm iteratively selects the classifiers, filters with their quantizers, to

minimize the classification error. After I iteration, I classifiers

$$h_{F,T}(x_1, x_2) = \begin{cases} +1 & \text{if} \quad Q_T(F(x_1)) = Q_T(F(x_2)) \\ -1 & \text{if} \quad otherwise \end{cases}$$

are selected.

Pairwise Boosting Algorithm

---

input: D samples $\{(x_{11}, x_{21})..(x_{1D}, x_{2D})\}$ and their labels $y_d \in \{\pm 1\}$
initialize: $w_d = \frac{1}{D}, i = 1, ..., D$
for m=1:I

1. chose the i-th classifier $h_{F,T}(\cdot)$ that minimize the weighted error $\epsilon^{(i)} = \sum_{d=1}^{D} w_d^{(i)} \cdot \delta(h^{(i)}(x_{1d}, x_{2d}) \neq y_d)$
2. update weights $w_d^{(i+1)} = w_d^{(i)} \cdot (\frac{1-\epsilon^{(i)}}{\epsilon^{(i)}})^{-h^{(i)}(x_{1d}, x_{2d})y_d}$
3. normalize weights $\sum_{d=1}^{D} w_d^{(i+1)} = 1$

output: $F^{(i)}$ and $T^{(i)}$ $for\ i = 1, ..., I$

The binary fingerprint for each SSC image is extracted by concatenating

$$B(Q_{T^(i)}(F^{(i)}(x)))$$

for $i = 1, ..., I$.
To find matching fingerprint two distance measure are defined: $D_H(\cdot)$, the hamming distance (the number of bit errors) and $D_Q(\cdot)$, the distance between two quantized outputs.

$$D_Q(b_1, b_2) = \sum_{i=1}^{I} \left| Q_T^{(i)}\left(F^{(i)}(x_1)\right) - Q_T^{(i)}\left(F^{(i)}(x_2)\right) \right|$$

To test the algorithm the parameters used are: $L_w = 371.52$ms, $L_s = 185.76$ms, I=16 or 32 and T=3. Therefore an 5s long audio clip is identified using 16×32 or 16×64 bits fingerprint.
Tests are performed on 11000 matching and 220000000 non matching pairs of audios. All audios are 96 kb/s MP3 encoded (MP3). Moreover audios can be subjected to the following additional distortions: time delay (TD) of 92.9ms, octave bands equalization (EQ1) where adjacent bands are attenuated by -6 and +6 dB in an alternating fashion, volume change (V), echo (E), bandpass filtering (BPF) from 400Hz to 4kHz, 64 kb/s WMA encoding (WMA), one third octave bands equalization (EQ2), and sampling rate change (SR) to down-sample to 16kHz and up-sample to 44.1 kHz. Furthermore tests are performed in a comparative

way. On the same dataset the performance of [7], denoted as "SP+APB", and [10], "SSC+APB" , are calculated. The results are reported in Figure 3.4 and Figure 3.5 for 32 and 64-bit fingerprints respectively. The plots show the Receiver Operating Characteristics (ROC) curve which plots the false negative (FN) rate versus the false positive (FP) rate.

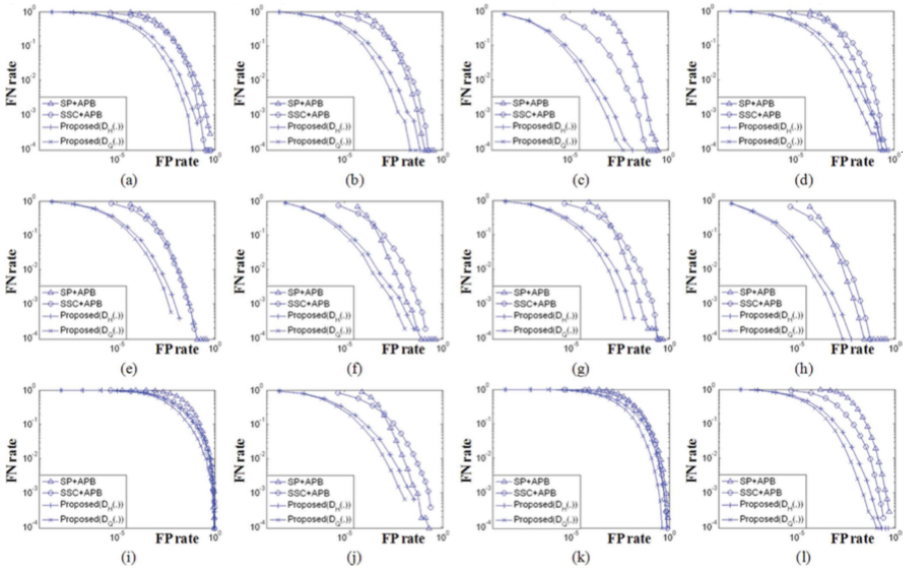


Figure 3.4: Results of a 32 bit fingerprint against various distortions: a) TD + MP3. (b) EQ1 + MP3. (c) V + MP3. (d) E + MP3. (e) BPF + MP3. (f) WMA + MP3. (g) EQ2 + MP3. (h) SR + MP3. (i) TD + EQ1 + V + E + MP3. (j) WMA + EQ2 + SR + MP3. (k) TD + E + BPF + EQ2 + MP3. (l) EQ1 + V + BPF + WMA + MP3.

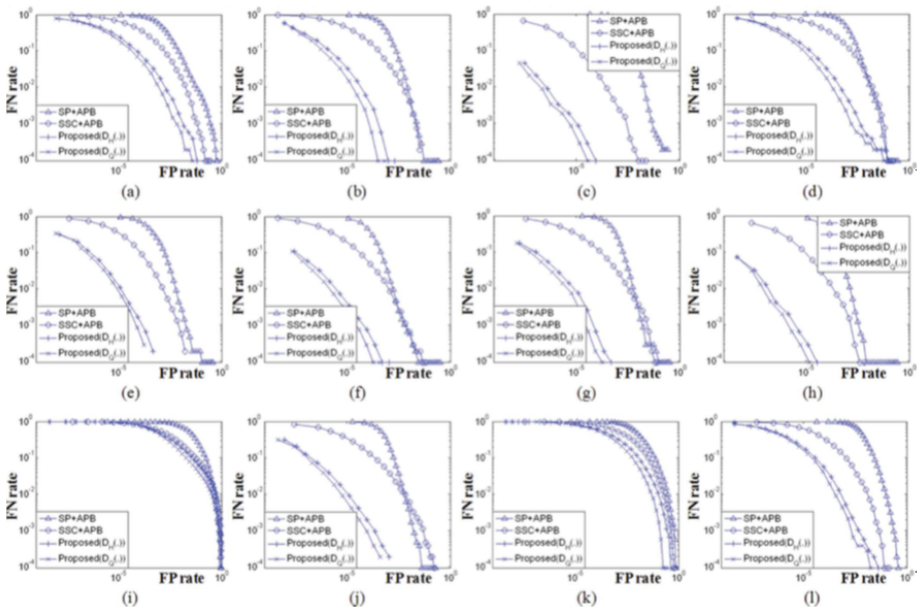


Figure 3.5: Results of a 64 bit fingerprint against various distortions: (a) TD + MP3. (b) EQ1 + MP3. (c) V + MP3. (d) E + MP3. (e) BPF + MP3. (f) WMA + MP3. (g) EQ2 + MP3. (h) SR + MP3. (i) TD + EQ1 + V + E + MP3. (j) WMA + EQ2 + SR + MP3. (k) TD + E + BPF + EQ2 + MP3. (l) EQ1 + V + BPF + WMA + MP3.

Chapter 4

Our Solutions

Each section describes one of the algorithm we have implemented. For each one of them we describe the algorithm as presented in the paper, our implementation comprehensive of pseudo-code and the results we have achieved.

4.1 Robust Fingerprinting

4.1.1 Algorithm

The audio signal is segmented in overlapping frames, 0.37 seconds long. Each frame is weighted by a Hanning window with an overlap factor of 31/32 which means that frames are taken 11.6ms apart. Subsequent frames are largely overlapping producing a slow varying representation of the signal. For each frame a 32-bit sub-fingerprint is extracted. A frame is divided into 33 non overlapping and logarithmically spaced frequency bands between 300Hz and 2000Hz. The bits of the sub-fingerprint are defined as follows:

$$F(n,m) = \begin{cases} 1 & \text{if } E(n,m) - E(n,m+1) - (E(n-1,m) - E(n-1,m+1)) > 0 \\ 0 & \text{if } E(n,m) - E(n,m+1) - (E(n-1,m) - E(n-1,m+1)) \leq 0 \end{cases} \tag{4.1}$$

where $F(n,m)$ is the m-th bit of the sub-fingerprint of frame n and $E(n,m)$ is the energy of band m of frame n. One sub-fingerprint does not contain enough data to identify a audio. The basic unit is composed of 256 subsequent sub-fingerprints and it is called fingerprint-block.
Two audio signals are declared similar if the Hamming distance between the two fingerprint blocks is below a certain threshold T. T is a sensitive parameters because it influences the false acceptance rate, FAR, and the false rejection rate, FRR. Smaller T will decrease the false acceptance probability and will increase the false rejection one. The optimal threshold is set to $T = \alpha n$, where n is the number of bits extracted ($n = 32 \times 256$) and α is the Bit Error Rate. The BER was experimentally set to 0.35 by Haitsma and Kalker after having calculate the BER for 100,000 randomly selected pairs of fingerprints.

To test the scheme robustness the BER between the fingerprint-block of an original song and a degraded version of it is calculated. Figure 4.1 shows the BER of four songs subject to different kind of audio degradation. All bit error rates are well below the threshold $\alpha = 0.35$ besides linear speed changes.

Processing	Orff	Sinead	Texas	AC/DC
MP3@128Kbps	0.078	0.085	0.081	0.084
MP3@32Kbps	0.174	0.106	0.096	0.133
Real@20Kbps	0.161	0.138	0.159	0.210
GSM	0.160	0.144	0.168	0.181
GSM C/I = 4dB	0.286	0.247	0.316	0.324
All-pass filtering	0.019	0.015	0.018	0.027
Amp. Compr.	0.052	0.070	0.113	0.073
Equalization	0.048	0.045	0.066	0.062
Echo Addition	0.157	0.148	0.139	0.145
Band Pass Filter	0.028	0.025	0.024	0.038
Time Scale +4%	0.202	0.183	0.200	0.206
Time Scale –4%	0.207	0.174	0.190	0.203
Linear Speed +1%	0.172	0.102	0.132	0.238
Linear Speed -1%	0.243	0.142	0.260	0.196
Linear Speed +4%	0.438	0.467	0.355	0.472
Linear Speed -4%	0.464	0.438	0.470	0.431
Noise Addition	0.009	0.011	0.011	0.036
Resampling	0.000	0.000	0.000	0.000
D/A A/D	0.088	0.061	0.111	0.076

Figure 4.1: BER for different kinds of signal degradations for *"O Fortuna"* by Carl Orff, *"Success has made a failure of our home"* by Sinead o'Connor, *"Say what you want"* by Texas and *"A whole lot of roses"* by AC/DC.

4.1.2 Implementation

Before applying the above described algorithm, we time align pairs of samples to be processed using cross-correlation with a maximum lag of 150ms. Then they are trimmed in order to have an equal amount of samples. Since audio comparison should be carried out on at least one fingerprint-block, audio files shorter than 3s after trimming are discarded.

Both the file coming from the web browser and the phone are divided into 257 frames. They are stored into a matrix where each column represent one frame. Each frame is input to a Hanning window as long as the frame. Figure 4.2 shows the time representation of such a window.

Each frame is then divided into 33 non overlapping and logarithmically spaced

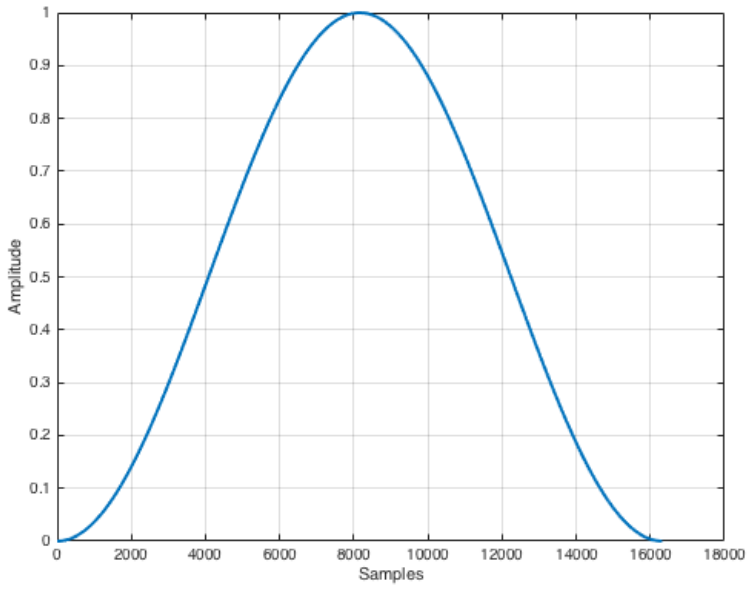


Figure 4.2: 16317-point ($frame\_length * f_s = 0.37 * f_s = 16317$) Hanning window

frequency bands. The range of frequency is enlarged between 16Hz and 4kHz due to the environment of the recordings. Table 4.1 reports the central frequency of each band. The lower and upper bound of each band can be calculated dividing and multiplying the central frequency by 1.09, $f_{low} = f_{centr}/1.09$ and $f_{high} = f_{centr} * 1.09$. In this way the per-cent fractional bandwidth is constant:

$$BW = 100 \times \frac{f_{high} - f_{low}}{f_{centr}}$$

Logarithmically spaced bands between 300Hz and 2000Hz are well suited to represent the most relevant spectral range in music. Sound-proof must be able to capture environmental noise in all possible settings, for example the train whistle in a station. The frequency range currently adopted by Sound Proof, from 50Hz to 4000Hz, resulted in slightly worse performances and therefore it was discarded.

We have implemented the filter bank using Butterworth filters. The amplitude

Band	f_{centr}	*Band*	f_{centr}	*Band*	f_{centr}
1	16	12	106.76	23	712.36
2	19.1	13	126.86	24	846.52
3	22.59	14	150.75	25	1005.94
4	26.84	15	179.15	26	1195.39
5	31.9	16	212.89	27	1420.51
6	37.91	17	252.98	28	1688.03
7	45.05	18	300.62	29	2005.93
8	53.54	19	357.24	30	2388.7
9	63.62	20	424.52	31	2832.62
10	75.6	21	504.46	32	3366.08
11	89.84	22	599.46	33	4000

Table 4.1: Non overlapping logarithmically spaced frequency bands between 16Hz and 4000Hz

response of such a filter is

$$|H(jw)| = \frac{1}{\sqrt{1 + (\frac{w}{w_c})^{2n}}}$$

where n is the filter order and w_c is the cutoff frequency. In order to derive the transfer function it is easier to work with a normalized filter, meaning that the cutoff frequency is 1, $w_c = 1\frac{rad}{s}$.

$$|H(jw)|^2 = H(jw)H(-jw) = \frac{1}{1 + w^{2n}}$$

By substituting $s = jw$, $H(s)H(-s) = \frac{1}{1+(\frac{s}{j})^{2n}}$.
The poles of the transfer function are given by:

$$1 + \left(\frac{s}{j}\right)^{2n} = 0 \Rightarrow s^{2n} = -(j)^{2n} \Rightarrow -1 = e^{j\pi(2k-1)} \quad \forall\ k\ integer$$

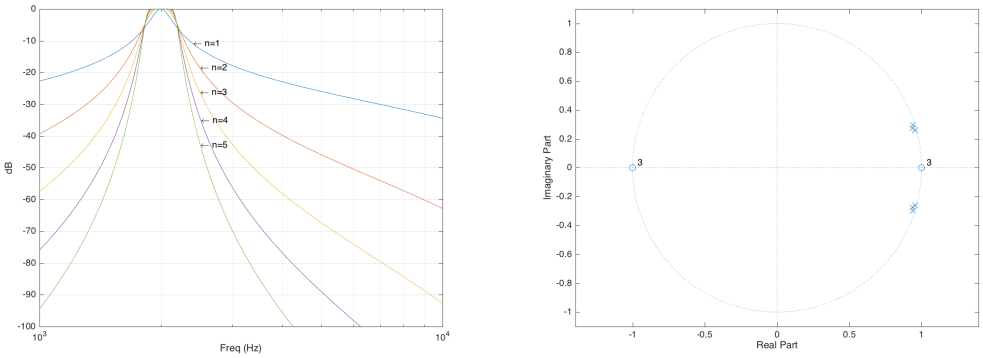
. Knowing that $j = e^{j\frac{\pi}{2}}$, the above result can be written as: $s^{2n} = e^{j\pi(2k-1+n)}$. Therefore the poles of $H(s)H(-s)$ lie in the unit circle at:

$$s_k = e^{\frac{j\pi}{2n}(2k-1+n)} \quad k = 1, 2, ..., 2n$$

$$s_k = cos\Big(\frac{\pi}{2n}(2k-1+n)\Big) + jsin\Big(\frac{\pi}{2n}(2k-1+n)\Big) \quad k = 1, 2, ..., 2n$$

Poles have to strictly be within the unit circle to have stable filters.
The transfer function of the Butterworth filter is:

$$H(s) = \frac{1}{(s - s_1)(s - s_2)\cdots(s - s_n)}$$



(a) Filters of order n=1 to n=5 (b) Pole locations for n=3

Figure 4.3: Bandpass Butterworth filters centered at 2000Hz

To design not normalized filters, w is replaced by w/w_c.

The implementation of the filter bank is done adapting the code from [11]. The changes needed are related to the frequencies used and more importantly the filter order. Filters of order three centered between 16Hz and 126.86Hz are not stable. Therefore we decrease their order to two. When the cutoff frequency is much smaller than the sampling frequency, specifically $f_c = 126.86$ and $f_s = 44100$, the poles move really close to the unit circle. Filter sharpness, which means an higher order filter, also causes the poles to move towards the unit circle. It is sufficient to reduced by one the order to assure filters stability.
After filtering, the energy content of each band is calculated as

$$\sum_{i=1}^{m} \frac{1}{m} |x[i]|^2$$

where m is the number of samples.
The fingerprint-block is calculated following equation 4.1. The result is a 32×256 matrix. 256 sub-fingerprints, one for each frame, 32 bits long. We compare the two fingerprint-blocks from the web and the phone samples by calculating their Hamming distance by subtracting the two matrices and looking at the number of non zero elements. Errors are made when in a determinate position in one block there is a 1 and in the other block there is a 0. The difference at that position will therefore be 1 or -1. On the contrary when both blocks have a 1 or a 0, the difference is 0.
Each fingerprint-block has $32 \times 256 = 8192$ elements. The threshold, i.e. the number of errors allowed, is set to $T = 3680$ yielding the best Equal Error Rate. The accepted Bit Error Rate is therefore increased to $\alpha = \frac{3680}{32 \times 256} = 0.45$. In the next section the results are calculated keeping the above discussed parameters.

Pseudo-code

```
Initialize:  frame_duration=0.37s, overlap=0.37-11.6e-3=0.3584s,
             number_of_frames=247, maxLag=11.6ms
for each pair of web and phone samples
   web=audioread('x_x_1web_x.wav')
   phone=audioread('x_x_2phone_x.wav')
   [web,phone]=crosscorr(web,phone,maxLag)
   web_in_frames=divide_in_frames(web,frame_duration,overlap)
   phone_in_frames=divide_in_frames(phone,frame_duration,overlap)
    for k=1:number_of_frames
      web(:,k)=hanning(web_in_frames(:,k))
      phone(:,k)=hanning(phone_in_frames(:,k))
      web(:,k)=one_third_octave_bands_filtering(web(:,k))
      phone(:,k)=one_third_octave_bands_filtering(phone(:,k))
      power_web(:,k)=sum(web^2(:,k))/length(web(:,k))
      power_phone(:,k)=sum(phone^2(:,k))/length(phone(:,k))
   end
   F1=fingerprint(power_web) % see Equation  4.1
   F2=fingerprint(power_phone) % see Equation  4.1
   hamming=F1-F2
   similarity=number_non_zero_elements(hamming)
```

4.1.3 Results

4047 pairs of matching samples were tested. 61 of those legitimate logins were rejected leading to a total False Rejection Rate of 0.01507 (Figure 4.4). 4240 pairs of non matching samples recorded in the same environment were tested. 64 of those attacks were classified as legitimate logins yielding a False Acceptance Rate of 0.01509. Figure 4.5 shows the Hamming distance between pairs of samples that should and should not match, blue and red crosses respectively, in the different environments. The blue crosses over the threshold line are the False Rejections while the red crosses under the threshold are the False Acceptances.

The FRR is evaluated in each setting: environment, user activity, and phone position. Table 4.5 and Figure 4.5 show the similarity score for each environment. The FRR is 0.006 (4 logins over 631) in the Office environment, 0.011 (10 logins over 884) in the HomeTV environment, 0.013 (12 logins over 902) in the LaptopMusic environment, 0.018 (12 logins over 656) in the Cafe environment, 0.021 (5 logins over 237) in the Lecture environment, and 0.024 (18 logins over 743) in the TrainStation environment. There is not a significant difference in the similarity score. The worst performance is recorded in the TrainStation where the

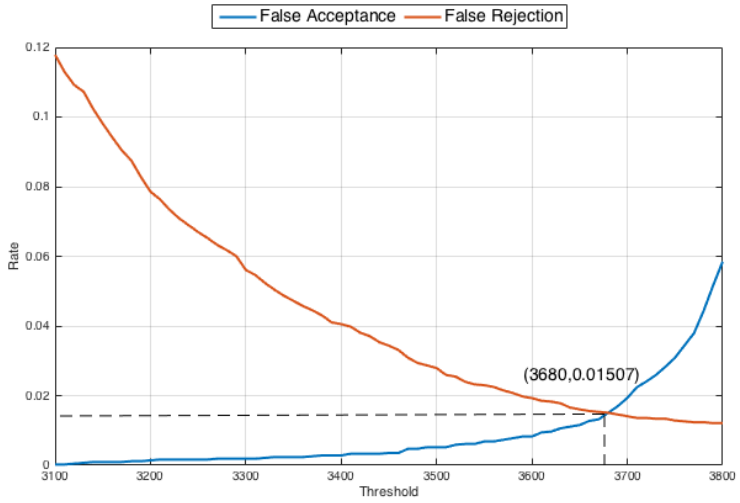


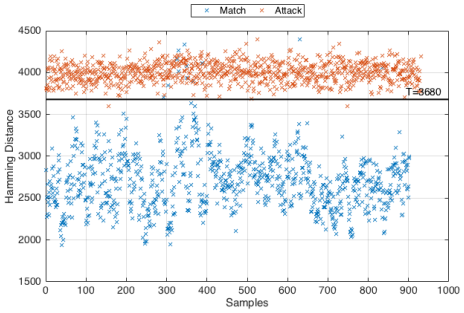
Figure 4.4: Robust Fingerprinting False Rejection Rate and False Acceptance Rate as a function of the threshold, T. The Equal Error Rate is 0.01507 at T=3680

background sound is the most different from music, the original application for which the method was designed.

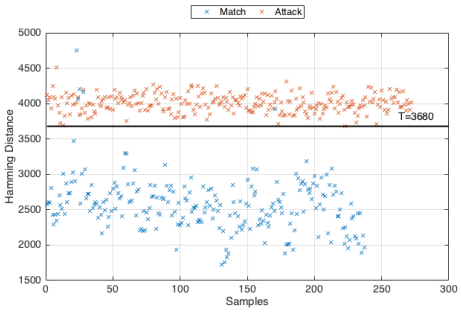
Environment	*rejection*	*login*	*FRR*
LaptopMusic	12	902	0.013
Lecture	5	237	0.021
HomeTV	10	884	0.011
Office	4	631	0.006
TrainStation	18	743	0.024
Cafe	12	656	0.018

Table 4.2: Robust Fingerprinting False Rejection Rate based on the environment

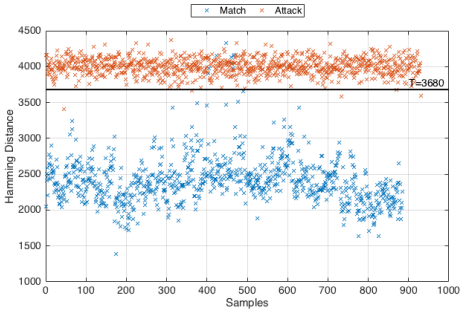
Figure 4.6 shows the similarity score based on the user activity. He could be silent, talking, coughing or whistling. The FRR is 0.032 (37 over 1147 logins) when the user is silent, 0.005 (6 over 1093 logins) when the user is coughing, 0.006 (6 over 1088 logins) when the user is speaking, and 0.017 (12 over 725 logins) when the user is whistling. In general, if the user makes some noise the accuracy improves. The error rate is particularly high when the user is silent in the enviroment TrainStation, 0.033, (15 logins over 243) and Cafe, 0.060, (13 logins over 216). These two environments remain the worst performing when looking at



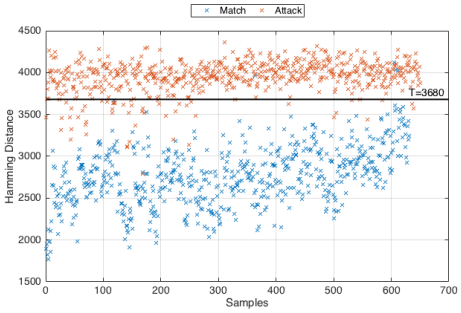
(a) LaptopMusic



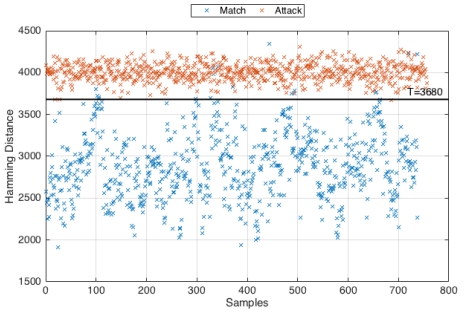
(b) Lecture



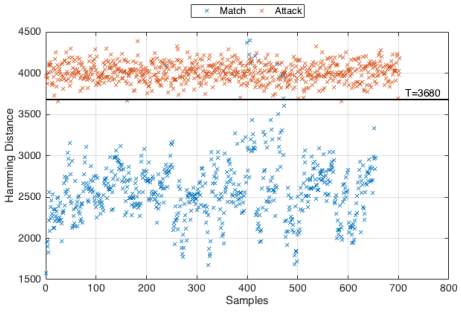
(c) Home



(d) Office

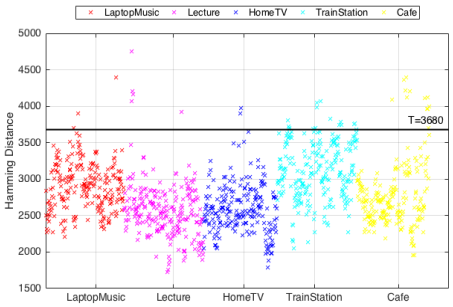


(e) Train Station

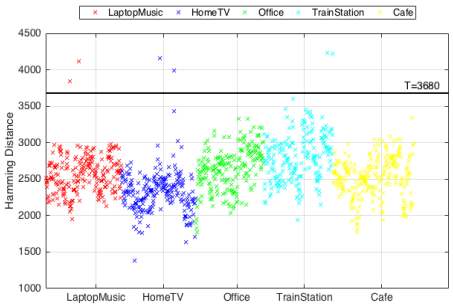


(f) Cafe

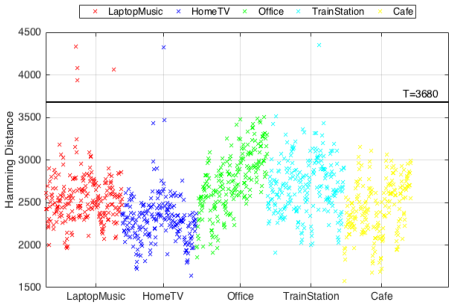
Figure 4.5: Error distribution based on the environment in Robust Fingerprinting



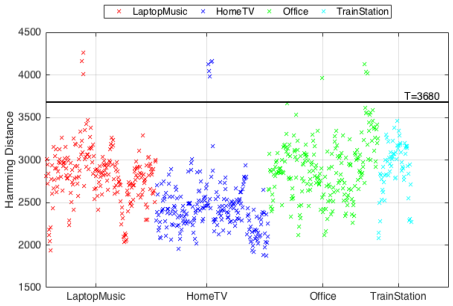
(a) Silence



(b) Coughing



(c) Talking



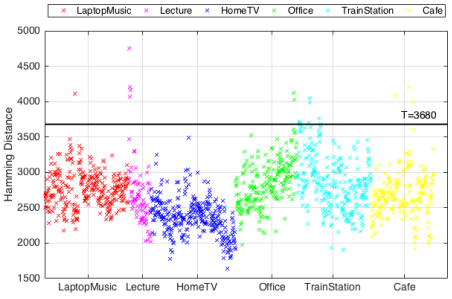
(d) Whistling

Figure 4.6: Impact of the user activity on the False Rejection Rate in Robust Fingerprinting

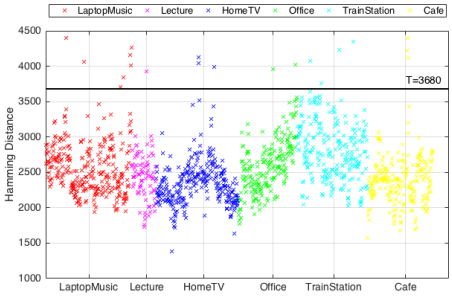
total FRR.

Figure 4.7 shows the similarity score based on the position of the phone: in the trousers pocket, outside on the table or on a bench or in a rucksack. The FRR is 0.013 (17 over 1356 logins) when the phone is in a pocket, 0.015 (20 over 1345 logins) when the phone is outside, and 0.018 (24 over 1352 logins) when the phone is in a rucksack. The position of the phone does not significantly influence the performance.

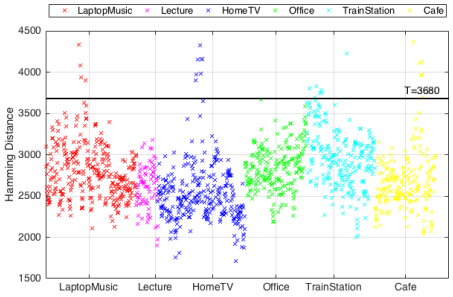
There are not malicious logins accepted in the Lecture environment. The FAR is 0.02 (2 logins over 930) in the LaptopMusic environment, 0.003 (2 logins over 756) in the TrainStation environment, 0.004 (3 logins over 702) in the Cafe environment, 0.005 (5 logins over 930) in the HomeTV environment, and 0.080 (52 logins over 650) in the Office environment. The FAR is higher for the Office environment compared to all the others. The same behavior is found in the current implementation of Sound-Proof, where the FAR is 0.025. This value is much higher compared to all the other environments as Table 2.5 illustrates. We can deduct that in the Office enviroment is particularly difficult to have robust fingerprints.



(a) Pocket



(b) Outside



(c) Rucksack

Figure 4.7: Impact of the phone position on the False Rejection Rate in Robust Fingerprinting

Environment	*User Activity*	*rejection*	*login*	*FRR*
LaptopMusic	Silence	3	232	0.013
	Coughing	2	225	0.009
	Speaking	4	227	0.018
	Whistling	3	218	0.014
Lecture	Silence	5	237	0.021
	Coughing	-	-	-
	Speaking	-	-	-
	Whistling	-	-	-
HomeTV	Silence	2	219	0.009
	Coughing	2	221	0.009
	Speaking	1	221	0.005
	Whistling	5	223	0.022
Office	Silence	-	-	-
	Coughing	0	202	0
	Speaking	0	211	0
	Whistling	4	218	0.018
TrainStation	Silence	15	243	0.062
	Coughing	2	206	0.009
	Speaking	1	228	0.004
	Whistling	0	66	0
Cafe	Silence	13	216	0.060
	Coughing	0	239	0
	Speaking	0	201	0
	Whistling	-	-	-

Table 4.3: Robust Fingerprinting False Rejection Rate based on the user activity

It is worth noticing that in an office the only background noise is the one the user produces.
There are not any other significant differences in performance.

4.2 Waveprint

4.2.1 Algorithm

The two audio files are converted into a spectrogram using 371ms long slices, taken every 11.6ms, evaluated in 32 logarithmically spaced frequency bands between 318Hz and 2kHz. Then 1.48s long spectral images are extracted, each one 0.9s

Environment	*User Activity*	*rejection*	*login*	*FRR*
LaptopMusic	Pocket	1	290	0.003
	Outside	7	298	0.023
	Rucksack	4	314	0.013
Lecture	Pocket	4	77	0.052
	Outside	1	85	0.012
	Rucksack	0	75	0
HomeTV	Pocket	0	295	0
	Outside	3	287	0.010
	Rucksack	7	302	0.023
Office	Pocket	2	213	0.009
	Outside	2	204	0.010
	Rucksack	0	214	0
TrainStation	Pocket	7	262	0.027
	Outside	4	244	0.016
	Rucksack	7	237	0.030
Cafe	Pocket	3	219	0.014
	Outside	3	227	0.013
	Rucksack	6	210	0.029

Table 4.4: Robust Fingerprinting False Rejection Rate based on the phone position

Environment	*acceptance*	*login*	*FAR*
LaptopMusic	2	930	0.002
Lecture	0	272	0
HomeTV	5	930	0.005
Office	52	650	0.080
TrainStation	2	756	0.003
Cafe	3	702	0.004

Table 4.5: Robust Fingerprinting False Acceptance Rate in similar environments

apart.
After this processing spectral images are treated as an image-query system. Each image is further processed using wavelets, a mathematical tool to hierarchically decomposing functions. The wavelet transformation is not robust against noise or audio degradations. To achieve robustness only the top wavelet coefficients by magnitude are retained for each image. Jacobs's findings [12] revealed that the

values of the top coefficients are not needed. Each wavelet can be represented by two bits. Each top wavelet is labeled 10 for positive values or 01 for negatives. The remaining majority of not top wavelets are labeled as 00. The process yields a sparse vector, amenable for further dimensional reduction through the Min-Hash technique.
Min-hash is a locality sensitive hashing scheme used to estimate how similar two sets are as discussed in Chapter 2. The method works with binary vectors as follows: select a random permutation of all the vector positions and reorder the vectors based on this reordering. Determine the position of the first "1" occurrence. For computational reason if the first 1 appears after position 255 it is recorded as if it occur at position 255. This allows to represent the signature in 1 byte. This procedure measures the similarity of the vectors based on the matching positions of "1" because the zeros are uninformative based on the coding scheme of wavelets coefficients. The probability that the first occurrence of a 1 is in the same position for two vectors is the same as the Jaccard similarity of the two vectors.
The method described is repeated 50 times using a different permutation each time, resulting in a signature for the vector.

The probe sample is 60s long and spectral images are extracted every 46ms. The scheme was tested on 10,000 songs with different degradation sources on the probe signal. The signal degradations evaluated are: time-offset only, echo, equalizer, GSM adaptive multi-rate at 4.75 kbps CBR (AMR: GSM Codec), add Enya's "Watermark I" to the probe (Noise - "Enya"), add To Die For's "Veil of Tears Epilogue" (Noise - "Veil of Tears"), change the playback speed ±2% (LSM-98 and LSM-102), time-scale modification of ±10% of the tempo (TSM-90 and TSM-110). The results are reported in Table 4.6.

Degradation Source	*Accuracy*
Time-offset only	100.00
Echo-90%	99.90
Equalizer	99.80
MP3-32K	97.47
Noise-"Enya"	86.97
Noise-"Veil of Tears"	82.29
AMR: GSM Codec	95.86
LSM-102	80.30
LSM-98	93.74
LSM-90	99.40
LSM-110	99.09

Table 4.6: Accuracy tested on 10,000 songs using different sources of degradation

4.2.2 Implementation

Before applying the above described algorithm, we time align pairs of samples to be processed using cross-correlation with a maximum lag of 150ms. Then they are trimmed in order to have an equal amount of samples.
For each audio signal we calculate the spectrogram, a visual representation of the spectrum of frequencies in a sound. To calculate it the signal is divided into segments of length equals to 371ms ($frame\_length$). Each segment is windowed by an Hanning window of equal length. Moreover two subsequent segments are taken 11.6ms ($frame\_interval$) apart. The discrete Fourier Transform is evaluated in 32 bands ($nBands$) in the enlarged frequency spectrum between 19.1Hz and 4000Hz. In order to extract 1.48s long spectral image ($spectral\_image\_length$) every 0.9s ($spectral\_image\_sampling\_stride$) we have to extract

$$w = nBands \times \left\lfloor \frac{spectral\_image\_length - frame\_duration}{frame\_interval} \right\rfloor \tag{4.2}$$

samples every

$$s = nBands \times \left\lfloor \frac{spectral\_image\_sampling\_stride}{frame\_interval} \right\rfloor \tag{4.3}$$

samples.
We decompose each spectral image using the wavelet transformation described in Chapter 2. Of all the coefficient returned we keep only the top 200 by magnitude and we keep only their sign as explained before. The result is a binary fingerprint for each spectral image from the web browser sample and from the phone sample. The similarity of each pair of fingerprint is calculated using Min-Hash. We implement it using the Caltech Large Scale Image Search Toolbox [13]. The algorithms are implemented in C++ but it has a MEX interface for MATLAB. We create a Locality Sensitive Hashing index with the ccvLshCreate function. The type of hash function is 'min-hash' and the distance metric is Jaccard. The function returns an integer which is a pointer to a C++ structure. Into this structure we input the two vectors to be compared with the function ccvLshInsert. Then we search the vector coming from the phone in the structure using ccvLshKnn. CcvLshKnn returns the distance between the vector and the k-nearest neighbor. If we choose to return 2-nearest neighbor, an array with two elements is the output of the function. The first element is the distance of phone spectral image from itself and the other one it is the distance from the web spectral image. The distance is 0 when the two input vectors are the same, it is Inf when they do not match at all. Otherwise it is a number between 0 and 1.

Pseudo-code

```
Initialize:  frame_length=0.371s, frame_interval=0.0116,
             overlap=frame_length-frame_interval,
```

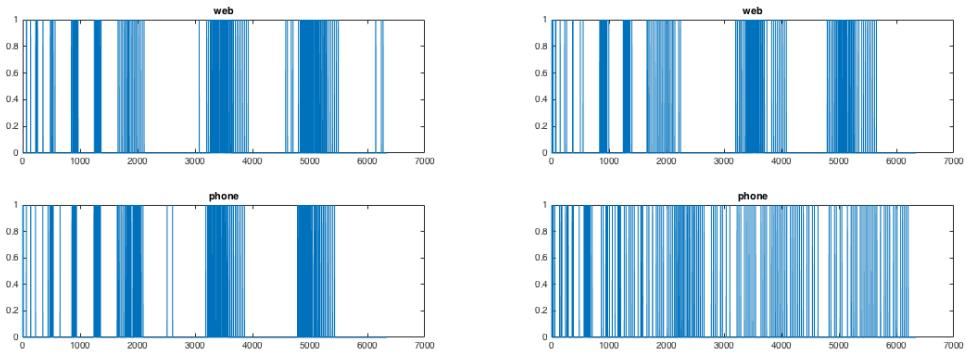
```
            spectral_image_length=1.48,
            spectral_image_sampling_stride0.9, fs=44100,
            Fc=see Table  4.1 , w=see Equation  4.2
            s=see Equation  4.3
for each pair of web and phone samples
   web=audioread('x_x_1web_x.wav')
   phone=audioread('x_x_2phone_x.wav')
   [web,phone]=crosscorr(web,phone,maxLag)
   spectrogram_web=spectrogram(web,hanning(frame_length),overlap,Fc,
                   fs)
   spectrogram_phone=spectrogram(phone,hanning(frame_length),overlap,
                   Fc,fs)
   spectral_images_web=(spectrogram_web,s,w)
   spectral_images_phone=(spectrogram_phone,s,w)
   for each spectral image
      wavelet_web=wavelet_decomposition(spectral_images_web)
      wavelet_phone=wavelet_decomposition(spectral_images_phone)
      top_web=select_top_coefficients(wavelet_web)
      top_phone=select_top_coefficients(wavelet_phone)
      lsh=lsh_create('min-hash','jaccard distance')
      lsh_insert(lsh,[top_web,top_phone])
      dist=lsh_knn(lsh,top_phone,2nd-knn)
   end
```

4.2.3 Results

The wavelet transformation seems to capture the environmental noise main characteristics. Figure 4.8 shows the the top 200 wavelets coefficients for two matching audio samples and two non matching ones. The blue lines represents the occurrence of a 1 in the vector which means that in that position there is one of the top 200 coefficients by magnitude. In matching spectral images the positioning of the 1s is roughly the same. It is possible that some of them are shifted by some positions or that a couple of them are completely off alignment but overall a clear pattern is recognizable. For example the pattern is clearly detectable in Figure 4.8a where matching spectral images are displayed. On the contrary, in Figure 4.8a the position of the 1s is completely random.

The min-hash technique should extract this pattern from the matching vector. The Caltech Large Scale Image Search Toolbox [13] min-hash implementation does not suit the characteristics of the vectors produced by Waveprint.
To get an understating of how min-hash works we have done some tests with a sparse binary vector of 256 elements.

$$[0,0,0,0,1,1,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,1,1,$$



(a) Pair of matching audios (b) Pair of non matching audios

Figure 4.8: Top 200 wavelets coefficients by magnitude for a spectral image

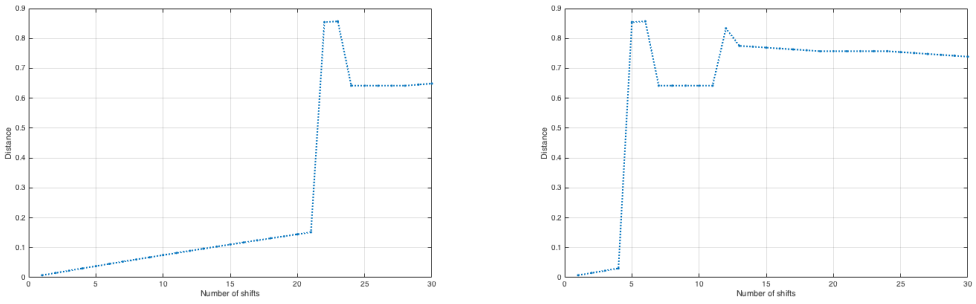
$$0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,1,1,1,0,$$
$$0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,$$
$$0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,1,1,1,1,0,1,1,$$
$$0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,$$
$$0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,1,1,0,1,1,1,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,$$
$$0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]$$

This vector is compared with a shifted copy of itself. The min-hash function outputs the distance between the two input vectors. If the two inputs match exactly the output distance is 0 and if they do not match at all is Inf. Otherwise it is a number between 0 and 1.
The first test we perform is a circular shift of 30 positions to the right. On a 10 elements vector shifting 2 positions circularly to the right means:

$$\begin{aligned} original &\quad [1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10] \\ shift1 &\quad [10 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9] \\ shift2 &\quad [9 \quad 10 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8] \end{aligned}$$

The distance between the two vectors linearly increases for the first 21 shift but remains small (≤ 0.15) as displayed in Figure 4.9a. This behavior is expected since the two inputs are the same shifted pattern. The abrupt distance increase to 0.85 happens when the first bit of the two vectors is different.
The same behavior is observed when circularly shifting 30 positions to the left (Figure 4.9b). For explanatory constrains, the 10 elements vector is used again:

$$original \quad [1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10]$$



(a) Shift rigth (b) Shift left

Figure 4.9: Distance output of the min-hash function as a function of the shift applied to the second input vector.

$$\begin{aligned} shift1 \quad & \begin{bmatrix} 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 1 \end{bmatrix} \\ shift2 \quad & \begin{bmatrix} 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 1 & 2 \end{bmatrix} \end{aligned}$$

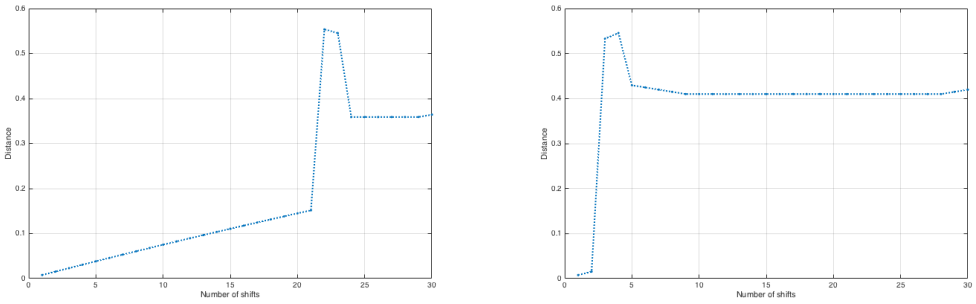
In this case, the jump is at the fifth shift but it happens for the same reason as before. It corresponds to the shift in which the first bit of the two vectors does not match.
To test how the algorithm works when the pattern of bits it is not only shifted but also broken we shift 30 positions circularly to the left and right in the middle of the vector. Again a two positions circularly right shift on the 10 elements vector is:

$$\begin{aligned} original \quad & \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{bmatrix} \\ shift1 \quad & \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 10 & 6 & 7 & 8 & 9 \end{bmatrix} \\ shift2 \quad & \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 9 & 10 & 6 & 7 & 8 \end{bmatrix} \end{aligned}$$

The above described behavior is repeated even when the array is broken in the middle (Figure 4.10).
Furthermore if the two vectors are basically the same and only the fourth bit in the second array is flipped, the output distance is 0.8468. On the other and, when only the 251-th bit of the second array is flipped the distance is 0.0232. When both the 4-th and 251-th positions are flipped the distance is again 0.8468.

This leads to the conclusion that the first position of bit flipping is highly significant. Therefore the algorithm is not suited to Waveprint where the two inputs vector have roughly the same pattern with some misalignment. In our case if two vectors are the same a part from a flipped first bit, the distance should be very low.

Given the result on this test vector we decide not to test this algorithm. We expect the current implementation to give poor results, no better than chance in classifying matching and non matching samples.



(a) Shift rigth (b) Shift left

Figure 4.10: Distance output of the min-hash function as a function of the shift applied to the second input vector in the middle.

4.3 Moving Average

4.3.1 Algorithm

Each audio sample, from the pair constituting a login attempt, is filtered using one third octave band filters into 30 frequency bands between 16 and 12500Hz. We have implemented the filter bank using Butterworth filters as described in the implementation section of the algorithm a Highly Robust Fingerprinting System. For each band the moving average is calculated on a window of w samples. The length of the window logarithmically decreases as the central frequency of the band increases to take into account the varying amount of information contained in each band. The used values are reported in Table 4.7. The overlap of two consecutive averages is 95% of the length of the window. Such a high overlap is necessary because the audio samples have pitches. By taking them into account in multiple windows the moving average smooths out the time representation of the signal.

In order to compare two samples two methods are tested. It is possible to compare the evolution of the moving average. For each window, w_i, the evolution is generated as follows:

$$Signature(i) = \begin{cases} -1 & \text{if} \quad movingAverage(w_i) < movingAverage(w_{i-1}) \\ 1 & \text{if} \quad movingAverage(w_i) > movingAverage(w_{i-1}) \\ 0 & \text{if} \quad movingAverage(w_i) = movingAverage(w_{i-1}) \end{cases}$$

The value of the first window, $movingAverage(0)$, is always set to 0. To compare two samples, the percentage of matching positions in the signature is computed.

As shown in Chapter 1, cross-correlation is another option to measure signal similarity. We calculate the normalized cross-correlation on the moving average

Band	f_{centr}	*window*	*Band*	f_{centr}	*window*
1	16	10,000	16	212.89	6500
2	19.1	9720	17	252.98	6320
3	22.59	9440	18	300.62	6140
4	26.84	9180	19	357.24	5970
5	31.9	8920	20	424.52	5800
6	37.91	8670	21	504.46	5640
7	45.05	8420	22	599.46	5480
8	53.54	8180	23	712.36	5330
9	63.62	7950	24	846.52	5180
10	75.6	7730	25	1005.94	5030
11	89.84	7510	26	1195.39	4890
12	106.76	7300	27	1420.51	4750
13	126.86	7090	28	1688.03	4610
14	150.75	6890	29	2005.93	4480
15	179.15	6690	30	2388.7	4350

Table 4.7: Window length, w, for each band

with a maximum lag, $\tau = 150ms$. It is important to calculate the new sampling frequency of each band when transforming the lag from seconds to a number of samples. If the audio file has N samples after trimming, its length in seconds is $length\_in\_seconds = \frac{N}{fs}$, where $fs = 44100$ and N is the number of samples. After the moving average is calculated the audio has n samples with $n < N$, therefore the new sampling frequency is $\frac{n}{length\_in\_seconds}$.
For each band we store the peak of the cross-correlation. The similarity score is calculated as the average of the normalized cross-correlation in the considered bands and it is compared to a predefined threshold to determine if the two audios come from the same environment.

Pseudo-code

```
Initialize:  Fc=see Table  2.1, w=see Table  4.7,
             o=0.95×w
for each pair of web and phone samples
   web=audioread('x_x_1web_x.wav')
   phone=audioread('x_x_2phone_x.wav')
   for each frequency band
      web_band=one_third_octave_band_filter(web,Fc of the band)
      phone_band=one_third_octave_band_filter(phone,Fc of the band)
      moving_average_web=moving_average(web_band, w of the band,
                  o of the band)
```

```
        moving_average_phone=moving_average(phone_band, w of the band,
                    o of the band)
        fs_web=length(moving_average_web)/(length(web)/fs)
        fs_phone=length(moving_average_phone)/(length(phone)/fs)
        max_corr=crosscorr(moving_average_web,moving_average_phone,
                    maxLag)
    end
similarity_score=mean(max_corr)
```

4.3.2 Results

Comparing the percentage of matching positions in signatures leads to unreliable results. It randomly labels pairs of samples as matching or non matching. The normalized cross-correlation is the similarity score chosen for this approach.

Figure 4.11 shows the average value of the normalized cross-correlation peak in each band. The black curve represents valid logins while the red one non valid logins. The shaded regions respectively represent the standard deviation of valid and non valid logins.

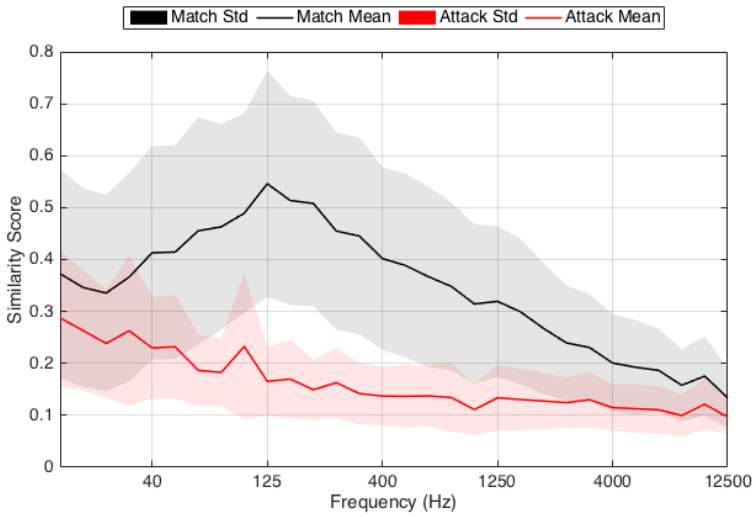


Figure 4.11: Mean and Standard Deviation of the normalized cross-correlation in each band for matching and non matching samples

There is a significant overlap in the lowest and highest frequencies which is further analyzed dividing the matching and non matching samples per environment. In the environment LaptopMusic there is a peak in the similarity score at 100Hz (Figure 4.12a). The peak is due to the fact that in that band there is only noise with an amplitude of the order of $10^{-3} - 10^{-4}$ in a consistent fraction

of samples. Bands carrying signal have an amplitude of the order of 10^{-1}. Non matching samples containing only noise have normalized cross correlation values up to 0.8 while the value drops down when some signal is present. This explains the above average standard deviation. The same consideration can be extended to the environment Cafe at 31.5 Hz (Figure 4.12f).

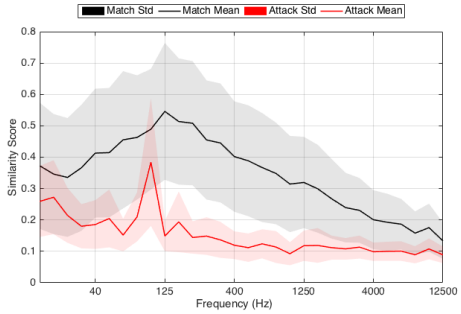
The smallest Equal Error Rate is achieved when considering bands between 50Hz and 1600Hz. As shown in Figure 4.13, the ERR is 0.01702 at the threshold 0.2357. The False Rejection Rate is 0.01702 meaning that 69 legitimate logins where rejected over 4053 attempts. The False Acceptance Rate is 0.01722, 73 fraudulent logins where classified as legitimate over 4240 attempts. The band centered at 100Hz which is non informative for the environment LaptopMusic is considered because it carries relevant information for the environments TrainStation and Cafe as seen in Figure 4.14

Environment	*rejection*	*login*	*FRR*
LaptopMusic	22	902	0.024
Lecture	5	237	0.021
HomeTV	7	884	0.008
Office	12	631	0.019
TrainStation	10	743	0.013
Cafe	13	656	0.020

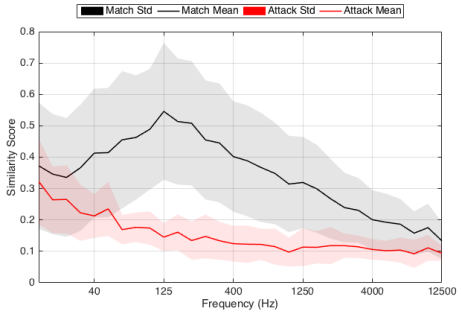
Table 4.8: Moving Average False Rejection Rate based on the environment

The algorithm performs equally well in all environments, indoors and outdoors. Figure 4.15 shows the similarity score of matching (blue crosses) and non matching pair of samples (red crosses). All blue samples below the threshold line are False Rejections while red samples above the threshold are False Acceptances.
The FRR in 0.008 (7 logins over 884) for the HomeTV environment, 0.013 (10 logins over 734) for the TrainStation environment, 0.019 (12 logins over 631) for the Office environment, 0.020 (13 logins over 656) for the Cafe environment, 0.021 (5 logins over 237) for the Lecture environment , and 0.024 (22 logins over 902) for the LaptopMusic environment. LaptopMusic and Lecture and the worst performing environments. Lecture is also challenging for the a High Robust Fingerprinting System, having one of the highest FRR.

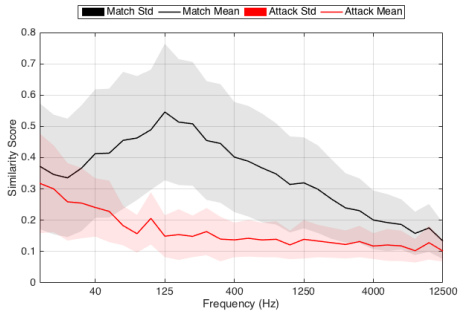
Table 4.9 shows the FRR based on the user activity. It is 0.036 (41 logins over 1147) when the user is silent, 0.032 (23 over 725) when the user is whistling. The performance increases when the user coughs or speaks. The FRR is 0.001 (2 logins over 1093) and 0.002 (3 logins over 1088) respectively. The worst performing environments when the user is silent are LaptopMusic (14 logins over 323) and Cafe (13 logins over 216) with a FRR of 0.060. Overall if the user makes some noise, the accuracy improves. Whistling is the activity that helps the least while coughing



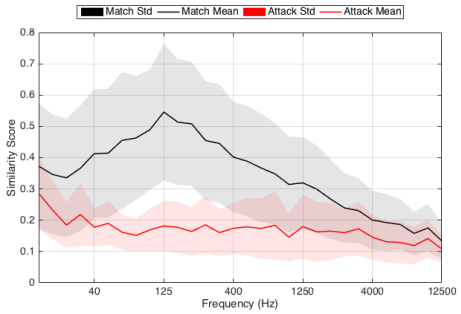
(a) LaptopMusic



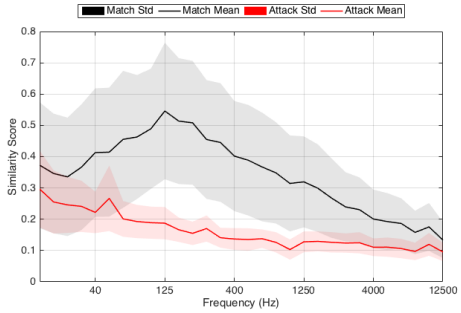
(b) Lecture



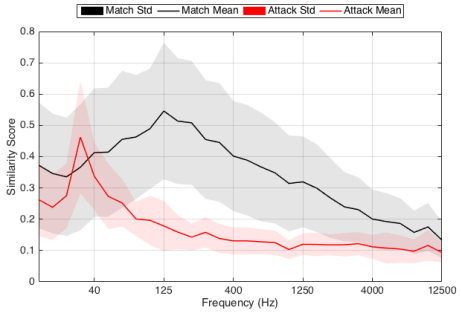
(c) Home



(d) Office



(e) Train Station



(f) Cafe

Figure 4.12: Mean and Standard Deviation of the normalized cross-correlation in each band for matching and non matching samples in a specific environment

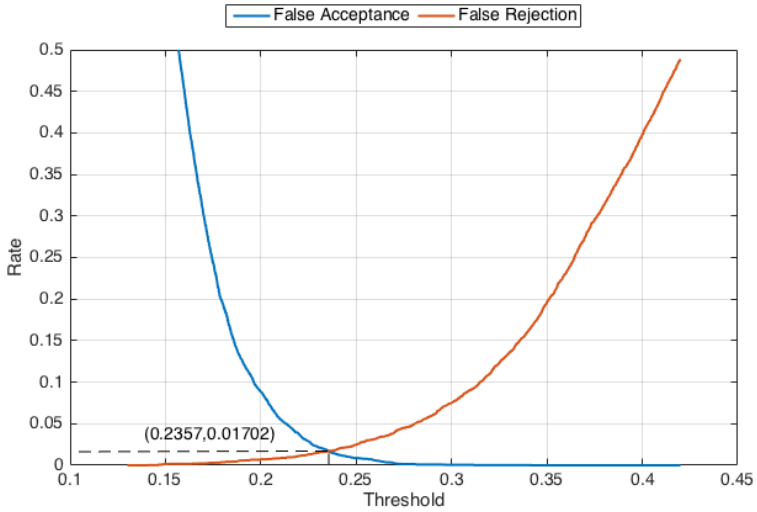
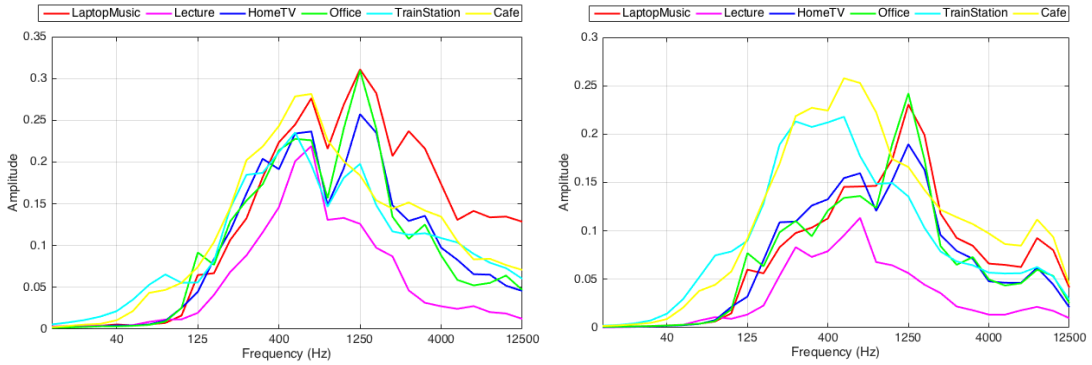


Figure 4.13: Moving Average False Rejection Rate and False Acceptance Rate as a function of the threshold, T. The Equal Error Rate is 0.01702 at T=0.2357

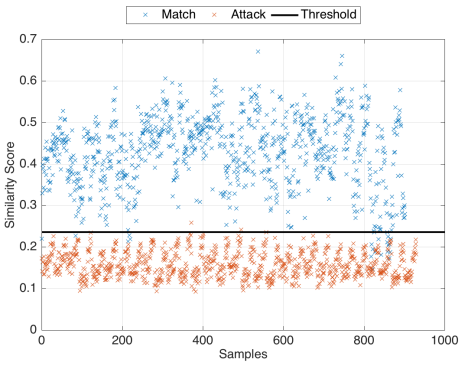


(a) Computer (b) Phone

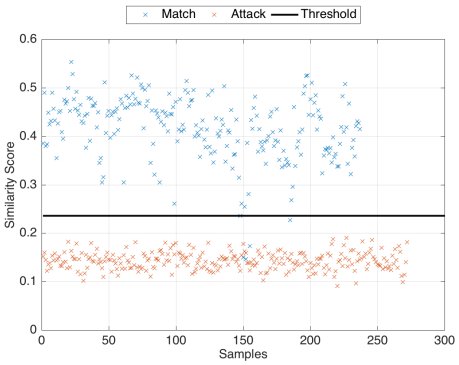
Figure 4.14: Average maximum amplitude per band in audio files recorded with a computer or a phone

helps the most. Coughing is significantly improving the accuracy also in a Highly Robust Fingerprinting System.

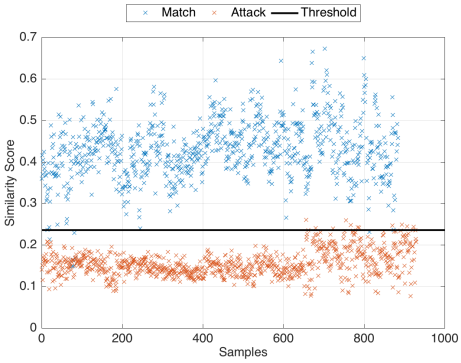
Table 4.10 shows the FRR based on the phone position. It is 0.010 when the phone is outside, in the user hand, (14 logins over 1345), 0.018 (25 logins over 1356) when the phone is in a pocket, and 0.022 (30 logins over 1352) when the phone is in a rucksack. Therefore the method performs slightly better when the phone is outside, on a bench or on a table.



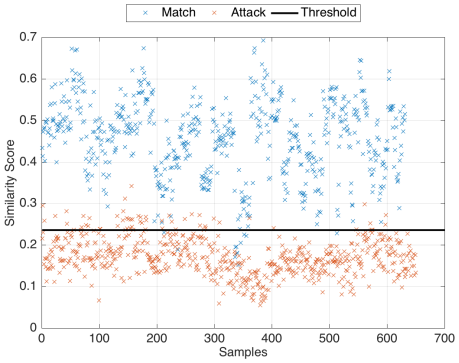
(a) LaptopMusic



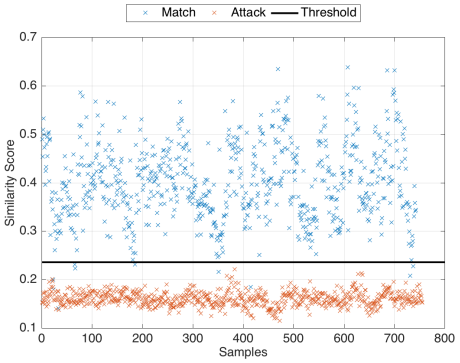
(b) Lecture



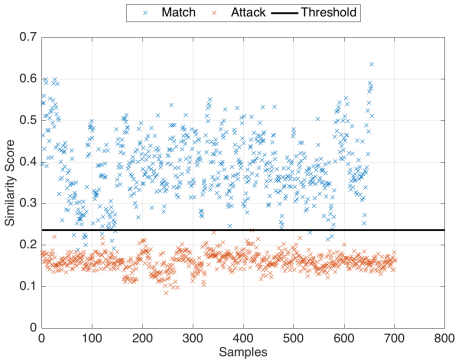
(c) Home



(d) Office

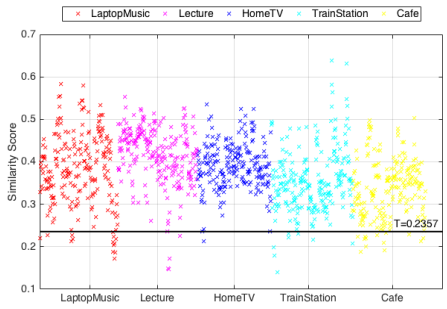


(e) Train Station

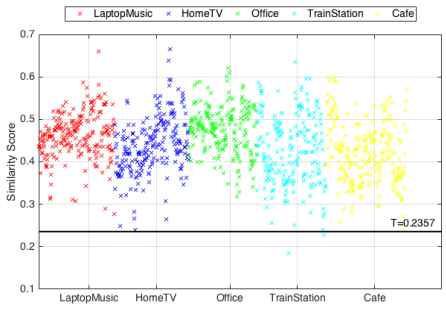


(f) Cafe

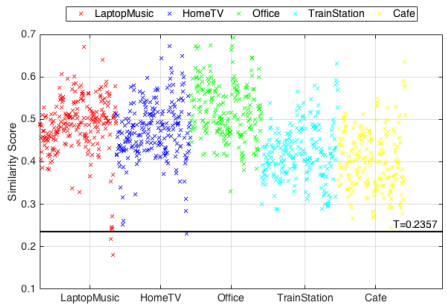
Figure 4.15: Moving Average Error distribution based on the environment



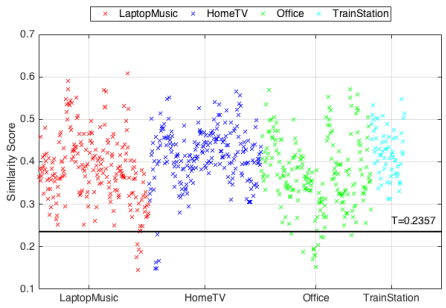
(a) Silence



(b) Coughing



(c) Talking



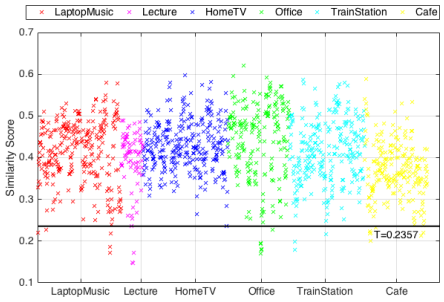
(d) Whistling

Figure 4.16: Impact of the user activity on the False Rejection Rate in Moving Average

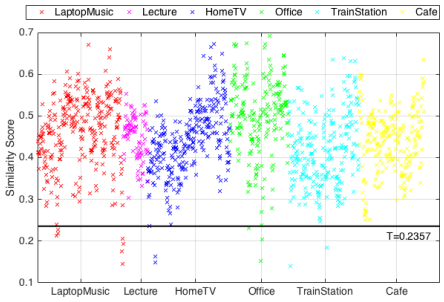
The False Acceptance Rate based on the environment is shown in Table 4.11. There are not any malicious logins accepted in the Lecture, TrainStation and Cafe environments. The FAR is 0.002 (2 logins over 930) for LaptopMusic, 0.013 (12 logins over 930) for HomeTV, and 0.091 (59 logins over 650) for Office. Office is by far the worst enviroment to classify matching and non matching samples. A Highly Robust Fingerprinting System and the current implementation of Sound-Proof, also have lower performances in this enviroment compared to the other settings.

Environment	*User Activity*	*rejection*	*login*	*FRR*
LaptopMusic	Silence	14	232	0.060
	Coughing	0	225	0
	Speaking	2	227	0.008
	Whistling	6	218	0.027
Lecture	Silence	5	237	0.021
	Coughing	-	-	-
	Speaking	-	-	-
	Whistling	-	-	-
HomeTV	Silence	1	219	0.004
	Coughing	0	221	0
	Speaking	1	221	0.005
	Whistling	5	223	0.022
Office	Silence	-	-	-
	Coughing	0	202	0
	Speaking	0	211	0
	Whistling	12	218	0.055
TrainStation	Silence	8	243	0.033
	Coughing	2	206	0.009
	Speaking	0	228	0
	Whistling	0	66	0
Cafe	Silence	13	216	0.060
	Coughing	0	239	0
	Speaking	0	201	0
	Whistling	-	-	-

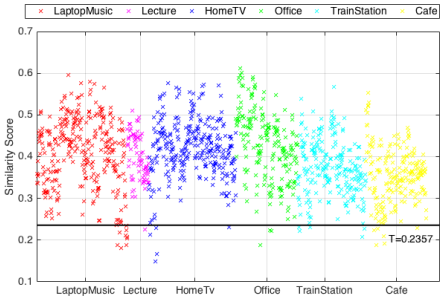
Table 4.9: Moving Average False Rejection Rate based on the user activity



(a) Pocket



(b) Outside



(c) Rucksack

Figure 4.17: Impact of the phone position on the False Rejection Rate in Moving Average

Environment	*User Activity*	*rejection*	*login*	*FRR*
LaptopMusic	Pocket	5	290	0.017
	Outside	7	298	0.023
	Rucksack	10	314	0.032
Lecture	Pocket	1	77	0.013
	Outside	0	85	0
	Rucksack	1	75	0.013
HomeTV	Pocket	0	295	0
	Outside	2	287	0.007
	Rucksack	5	302	0.016
Office	Pocket	6	213	0.028
	Outside	3	204	0.015
	Rucksack	3	214	0.014
TrainStation	Pocket	3	262	0.011
	Outside	2	244	0.008
	Rucksack	5	237	0.021
Cafe	Pocket	7	219	0.032
	Outside	0	227	0
	Rucksack	6	210	0.029

Table 4.10: Moving Average False Rejection Rate based on the phone position

Environment	*acceptance*	*login*	*FAR*
LaptopMusic	2	930	0.002
Lecture	0	272	0
HomeTV	12	930	0.013
Office	59	650	0.091
TrainStation	0	756	0
Cafe	0	702	0

Table 4.11: Moving Average False Acceptance Rate based on the environment

Chapter 5

Conclusions

The best performing algorithm is a Highly Robust Audio Fingerprinting System [2]. The achieved Equal Error Rate is 0.01507 which is slightly worst than the 0.002 of the current implementation of Sound-Proof. To further improve the performance an extensive grid search on the algorithm parameters such as frame length, frame overlap, frequency band, should be performed. On the other hand, the classification of matching and non matching samples can be improved on the current implementation using some machine learning classification techniques to find the best separating hyperplane between the two classes, matching and non matching audio files.
For example SVM, Support Vector Machines, finds the hyperplane that is at the largest distance to the nearest training data point of any class. This margin to the surfaces separating the classes lowers the generalization error when classifying new samples. A graphical interpretation of the SVM algorithm is given in Figure 5.1.

Another approach is to use ensemble methods such as the Random Forest classifier. Hundreds of decisions trees are trained on a random subset of the training set. A decision tree (Figure 5.2) partitions the input space into regions whose edges are axis aligned. Each region is assigned to a class. To determine the variable split and the threshold all possible choices are tested and the one that gives the smallest residual sum of squares is retained. The classification of a samples is the majority vote from all the individual regression trees. The advantage of this method is that the variance is reduced by a factor B, the number of different classifiers used.

Furthermore we have tested the algorithm only on a subset of the non matching audios, i.e. the malicious login attempts. In order to have statistically more relevant data we suggest to extend testing to the all database available.

Waveprint: Efficient Wavelet-Based Audio Fingerprinting [3] seems a promising approach when printing the output of the wavelet transform. Unfortunately the key point to test the algorithm is missing. The min-hash implementation from the Caltech Large Scale Image Search Toolbox is not suitable to find a pattern in similar arrays. Due to lack of time we have not tested other methods but further

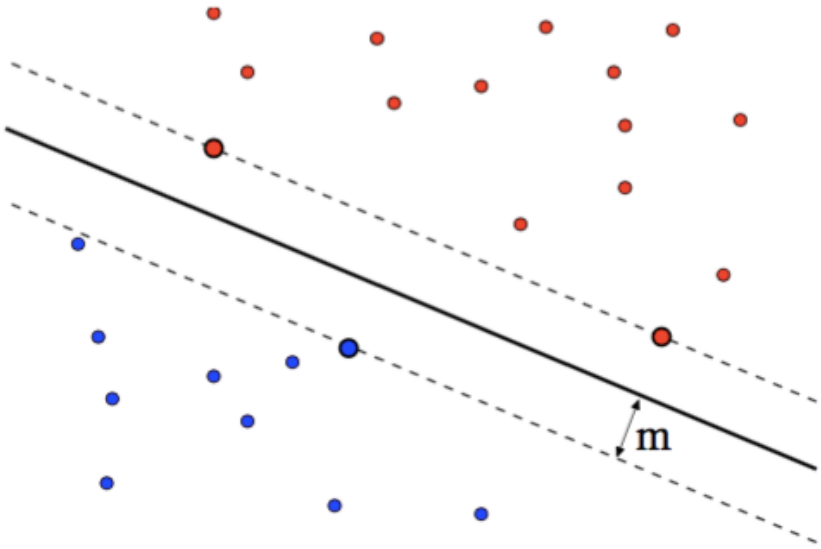


Figure 5.1: SVM algorithm on a two classes dataset

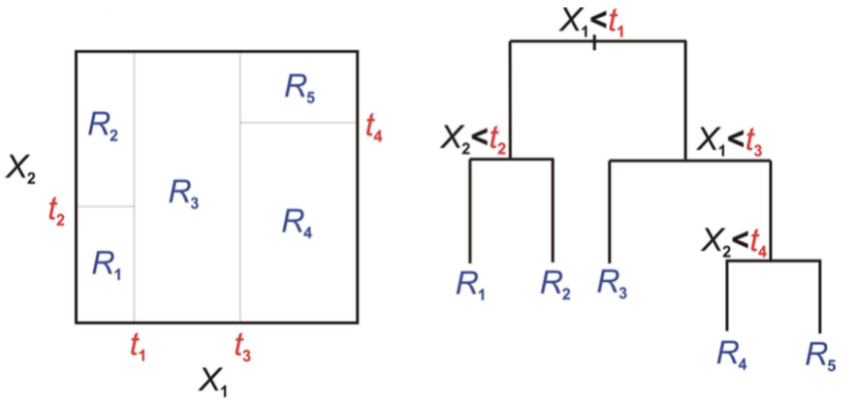


Figure 5.2: Decision Tree

research in the matter could lead to relevant results. The paper a Survey of Binary Similarity and Distance Measures [14] sums up different distance metrics that are suited for binary vectors. Some of them are tested in [15] on a 10 dimensional binary space and on handwriting recognition with 512 dimensional binary features vectors. The best performing ones are the Rogers-Tanmoto and Sokal-Michener similarity measures that are defined as follows:

$$S_{ROGERS-TANMOTO} = \frac{a+d}{a+2(b+c)+d}$$

$$S_{SOKAL-MICHENER} = \frac{a+d}{a+b+c+d}$$

where a counts the positions where there is a 1 in both vectors to be compared, b and c where there is a 1 in one vector and a 0 in the other, d where there is a 0 in both vectors. Another option would be to implement ourselves a min-hash function. We expect that an adequate similarity measure will give excellent results given the appreciable difference to the human eye of matching and non matching samples out of the wavelet transformation.

The moving average method is very similar to what Sound Proof already implements. The worse performance, Equal Error Rate of 0.01702, is due to averaging the output of each band filter. Averaging the signal representation looses resolution yielding a slightly worse accuracy when performing the normalized cross-correlation. It is highly unlikely that this approach can improve the accuracy. Nonetheless the machine learning algorithms describes before can be tried out to improve the current implementation Equal Error Rate.

To further improve Sound-Proof accuracy we propose to keep investigate a Highly Robust Audio Fingerprinting System. As published in a follow up paper [16], the authors suggest to replace the 33 energy band comparison by a local normalization and Fourier-Mellin transform. Local normalization emphasize the tonal part of the power spectrum by normalizing it with its local mean and it is robust against all kinds of audio processing steps such as local modification of audio spectrum. The Fourier-Mellin transform enables speed changes resilience. In our dataset speed changes are not present due to the nature of the recordings. Nonetheless since these changes improved the accuracy of the algorithm in the music recognition application, also for example for compression, we can reasonably expect improvements also in our implementation.

We would also suggest to further investigate the Pairwise Boosted Audio Fingerprint algorithm. This algorithm is used in the open content music database, MusicBrainz, on top of Chromaprint. Chromaprint works on the audio spectrogram and further transforms frequencies into notes. Since the audio files in Sound-Proof do not only contain music, we are not interested in the Chromaprint part of the algorithm but only in the boosted classification on matching and non matching samples. We have discarded this approach during our research because we do not have enough audio samples to implement it. Since it is a machine learning based algorithm two independent dataset are needed: one for training the classifier and one to test it. For example the authors of the paper used 22,000 matching and 22,000 non matching audios for training and 11,000 matching and 220,000,000 non matching audios for testing. Our dataset is composed of 4101 matching files which are too few to train and test the classifier.

Bibliography

[1] Nikolaos Karapanos, Claudio Marforio, Claudio Soriente, and Srdjan Capkun. Sound-proof: Usable two-factor authentication based on ambient sound. In *24rd USENIX Security Symposium (USENIX Security 15)*, pages 483–498, Washington, D.C., 2015. USENIX Association.

[2] Jaap Haitsma and Ton Kalker. A highly robust audio fingerprinting system. In *ISMIR*, volume 2002, pages 107–115, 2002.

[3] Shumeet Baluja and Michele Covell. Waveprint: Efficient wavelet-based audio fingerprinting. *Pattern Recognition*, 41(11):3467 – 3480, 2008.

[4] Specification for octave-band and fractional-octave-band analog and digital filters. *Americal National Standard ANSI S1.11-1986 (ASA 65-1986)*, 1993.

[5] Jeffrey D. Ullman Jure Leskovec, Anand Rajaraman. *Mining of Massive Datasets*. Cambridge University Press.

[6] Avery Wang et al. An industrial strength audio search algorithm. In *ISMIR*, pages 7–13, 2003.

[7] Yan Ke, Derek Hoiem, and Rahul Sukthankar. Computer vision for music identification. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 597–604. IEEE, 2005.

[8] Dalwon Jang, Chang D Yoo, Sunil Lee, Sungwoong Kim, and Ton Kalker. Pairwise boosted audio fingerprint. *Information Forensics and Security, IEEE Transactions on*, 4(4):995–1004, 2009.

[9] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[10] Sungwoong Kim and C.D. Yoo. Boosted binary audio fingerprint based on spectral subband moments. In *Acoustics, Speech and Signal Processing, 2007.*

ICASSP 2007. IEEE International Conference on, volume 1, pages I–241–I–244, April 2007.

[11] Christophe Couvreur. Implementation of one-third octave filter bank in matlab. `http://www.mathworks.com/matlabcentral/fileexchange/69-octave/content/octave/oct3bank.m`.

[12] Charles E. Jacobs, Adam Finkelstein, and David H. Salesin. Fast multiresolution image querying. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 277–286, New York, NY, USA, 1995. ACM.

[13] Caltech large scale image search toolbox. `http://vision.caltech.edu/malaa/software/research/image-search/`.

[14] Seung-Seok Choi, Sung-Hyuk Cha, and Charles C Tappert. A survey of binary similarity and distance measures. *Journal of Systemics, Cybernetics and Informatics*, 8(1):43–48, 2010.

[15] Bin Zhang and Sargur N Srihari. Properties of binary vector dissimilarity measures. In *Proc. JCIS Intl Conf. Computer Vision, Pattern Recognition, and Image Processing*, volume 1, 2003.

[16] Jin Soo Seo, Jaap Haitsma, and Ton Kalker. Linear speed-change resilient audio fingerprinting. In *Proc. IEEE Workshop on Model based Processing and Coding of Audio*, 2002.

List of Figures

List of Tables