BENCHMARK FOR MULTIMODAL AUTHENTICATION

Morgan Tirel [1], *Ekin Olcan Şahin* [2], *Guénolé C. M. Silvestre* [3], *Clíona Roche* [3], *Kıvanç Mıhçak* [2], *Sinan Kesici* [2], *Neil J. Hurley* [3], *Neslihan Gerek* [2], *Félix Balado* [3]

[1] University of Rennes, France
[2] Boğaziçi University, Turkey
[3] University College Dublin, Ireland

ABSTRACT

We report in this document on the development of a multimodal authentication benchmark during the eNTERFACE' 07 workshop. The objective of creating such a benchmark is to evaluate the performance of multimodal authentication methods built by combining monomodal authentication methods (i.e., multimodal fusion). The benchmark is based on a graphical user interface (GUI) that allows the testing conditions to be modified or extended. It accepts modular monomodal authentication algorithms (feature extraction, robust hashing, etc) and it allows them to be combined into multimodal methods. Attacks and benchmarking scripts are similarly configurable. An additional output of the project is a multimodal database of individuals, which has been collected in order to test the benchmark.

KEYWORDS

Benchmarking – Multimodal authentication – Feature extraction – Robust hashing

1. INTRODUCTION

Traditional authentication of individuals has usually been focused on methods relying on just one modality. Typically these modalities can be images of faces, hands (palms), irises or fingerprints, or speech samples. For instance, one may take a photo of the face of a person and obtain from it a nearly unique low-dimensional descriptor that identifies that person. Depending on the particular application targeted, this identifier can be obtained by means of different types of methods. Typical examples are feature extraction methods or, under some conditions, robust hashing methods, e.g. [1], [2]. The identifiers thus obtained can be compared to preexisting ones in a database for a match. Authentication systems based on multimodal strategies – that is, joint strategies– combine two or more monomodal methods into a multimodal one. For instance, it is possible to combine one method to hash an image using face images and another method to obtain a feature vector from a palm image. This is sometimes referred to as multimodal fusion. The aim is to increase the reliability of the identification procedure when combining different sources of information about the same individual (see [3], for example). As we will see, some other considerations are necessary in order to optimally undertake the merging of different multimodal methods.

Over the last number of years, many algorithms applicable to authentication have been proposed. Although some of these methods have been partially analyzed in a rigorous way, in many cases it is not feasible to undertake exhaustive analytical performance analyses for a large number of scenarios. This in part due to the sheer complexity of the task. Nevertheless, it is necessary to systematically evaluate the performance of new methods, especially when they are complex combinations of existing methods and used in a variety of scenarios. With such an evaluation it becomes possible to determine the best authentication strategies.

One way to tackle this problem is by means of benchmarking. Benchmarks have been proposed in the past for performance evaluation of many technologies, ranging from CPU units to watermarking technologies [4]. An advantage of benchmarks is that they see methods for testing as black boxes, which allows a high degree of generality. Despite this great advantage, one must be aware that benchmarks also entail issues such as how to choose fair (unbiased) conditions for benchmarking without an exponential increase in the associated computational burden.

The main goal of the eNTERFACE Workshop Project number 12 has been to create a GUI-driven benchmark in order to test multimodal identification strategies. This technical report contains information on the planning and development of this project. The remainder of this document is organized as follows. In Section 2 we describe the basic structure of the benchmark. In Section 3 we give the benchmark specifications which have been used as guidelines for implementing the benchmark, while Section 4 describes the methods and functions implemented to be tested within the benchmark. Finally, Sections 5 and 6 describe the database collection effort and the tests undertaken, while Section 7 draws the conclusions and future lines of this project.

2. DESCRIPTION OF THE BENCHMARK

Early in the project preparations, it was decided to implement the benchmark prototype in Matlab. This decision was taken in order to speed up the development time, as Matlab provides a rather straightforward procedure to build GUI applications, and it is faster to write Matlab code for the development of methods to be included in the benchmark. The downside is inevitably the execution speed, which can be critical for completing benchmark scripts within a reasonable timeframe. Nevertheless C code can also be easily interfaced to Matlab, using so called Mex files. The prototype is meant to be both usable and extendable, in order to facilitate the inclusion of new items and features. The interface has been designed so that extension or modification of the benchmark is almost completely automated. An exception is the addition of new benchmarking scripts (see Section 2.4), in order to keep the benchmark implementation simple. This means that it is possible to do most operations through the GUI, and manual adjustments of the source code are only necessary for the less frequent action of adding new types of benchmarking scripts. A scheme showing the relationships between the different parts of the benchmarking system is shown in Figure 1.

The benchmark relies on a database storing all relevant data. This is implemented in MySQL and interfaced to Matlab. The purpose of this database architecture is two-fold. Firstly, it is

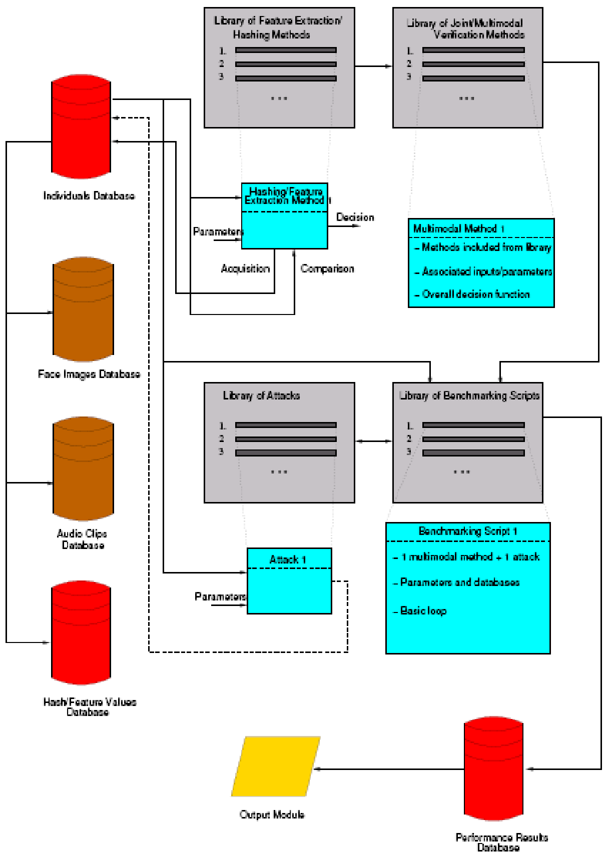

Library of Feature Extraction/ Hashing Methods
1.
2
3
...
Library of Joint/Multimodal Verification Methods
1.
2
3
...
Individuals Database
Hashing/Feature Extraction Method 1
Parameters
Decision
Acquisition
Comparison
Multimodal Method 1
– Methods included from library
– Associated inputs/parameters
– Overall decision function
Face Images Database
Library of Attacks
1.
2
3
...
Library of Benchmarking Scripts
1.
2
3
...
Audio Clips Database
Benchmarking Script 1
– 1 multimodal method + 1 attack
– Parameters and databases
– Basic loop
Attack 1
Parameters
Hash/Feature Values Database
Output Module
Performance Results Database

Figure 1: *Relationships between the main parts of the benchmark.*

an efficient way to store and access the information; secondly, it allows easy sharing of the data over a network in order to parallelize the benchmark in the future, thus distributing the unavoidable computational burden of the benchmark.

The project requires a database of individuals featuring signals such as face images, hand images and speech. The details on the database collection task are given in Section 5. All this information is stored in the MySQL database together with the identifiers (i.e., extracted features, hash values) obtained from the individuals, and all libraries of methods and functions. In order to minimize the effects of intra-individual variability, which especially affects some robust hashing algorithms (see for instance [5]), the database of individuals includes several instances of each identifier corresponding to a given individual.

The benchmark admits new modules through four libraries (see Figure 1) whose function we describe next.

2.1. Library of monomodal methods

This library contains standard monomodal methods which can be added, removed or edited through the GUI (see Section 3.6). For each method two functions are defined:

- An acquisition function, that takes as input a file containing a signal of the given modality (e.g., an audio clip or image) associated with a particular individual, as well as function-dependent parameters, such as thresholds and other. It outputs an identifier vector, binary or realvalued, depending on the method. The output identifier is stored in the database associated with the individual whose signal has been used.
- A comparison function, which takes as input two identifier vectors plus any necessary parameters, and outputs both a Boolean (hard) decision of similarity between them, and a (soft) reliability measure. This reliability shows the degree of confidence we put in the decision which is put forward by the function. As we will discuss in the next section, it is a key element in order to optimally combine two different modalities.

2.2. Library of multimodal methods

This library contains methods which, relying on the library in Section 2.1, specify ways to combine two (or more) monomodal methods in order to create multimodal identifiers. We may view this operation as an instance of multimodal fusion. For instance, the system allows the combination of a method to robustly hash face images with a method to extract features from a fingerprint; the newly created method is stored in the library as a multimodal method.

As already discussed, it is fundamental that each multimodal method implements an overall comparison function, able to break ties between possibly contradictory monomodal decisions when looking for matches in the database. Let us denote by e_1 the difference between the two input identifiers to the comparison function for modality type 1, and let us call d_1 the outcome of the monomodal binary decision, mapped without loss of generality to $+1$ and -1. If D_1 represents the random variable associated with that decision, with possible values $D_1 = +1$ (the two input identifiers correspond to the same individual) and $D_1 = -1$ (otherwise), the optimal monomodal decision is given by:

$$d_1 = \text{sign}\left(\log\frac{Pr\{D_1 = +1|e_1\}}{Pr\{D_1 = -1|e_1\}}\right). \qquad (1)$$

We may see the log-likelihood ratio as the reliability of the decision. We propose to obtain the overall decision d_F for the fusion of M modalities as

$$d_F = \text{sign}\left(\sum_{k=1}^{M} w_k \cdot \log\frac{Pr\{D_k = +1|e_k\}}{Pr\{D_k = -1|e_k\}}\right), \qquad (2)$$

where the subindex k refers to the modality k used in the fusion, and w_k is a set of positive weights such that $||\mathbf{w}||^2 = 1$. These weights reflect the importance that we wish to grant to each modality in the multimodal fusion. Note that in order to implement Eq. 1 accurate statistical modelling is required in order to obtain the conditioned probabilities, which may not always be feasible. In fact, many feature extraction and robust hashing methods implement this comparison function in a mostly heuristic way. If the reliability measures above are not available, it is always possible to implement a weaker version of Eq. 2 using the hard decisions:

$$\tilde{d}_F = \text{sign}\left(\sum_{k=1}^{M} w_k \cdot d_k\right). \qquad (3)$$

2.3. Library of attacks

It accepts attack functions on the signals stored in the individuals database. Attacked signals are used to assess how robust multimodal methods perform in two different situations:

1. The inputs are distorted versions of the authentic signals.
2. The inputs are non-authentic (malicious) signals, aiming at being wrongly verified as authentic.

2.4. Library of benchmarking scripts

It lists scripts which may be run in batch mode (i.e., autonomously), using signals from the database, a multimodal method, and attacks suitable to the modalities involved. Performance measures such as the rates of detection and false alarm (obtained by comparison with the authentic identifiers) will be computed during the execution of the script. In the scripts there may be loops where some attack parameters are generated pseudo-randomly.

3. BENCHMARK SPECIFICATIONS

We describe next the specifications that were used as technical guidelines to implement the benchmark. The most important structures and functions are described with some level of detail.

3.1. Individuals database

The basic structure of an entry in the individuals database is given by the following structure:

```
struct(
  'name',{},
  'authenticated',{},
  'file_list',struct(
    'name',{},
    'path',{},
    'type',{}
  ),
  'hash_list',struct(
    'method_name',{},
    'h_value',{}
  )
)
```

`h_value` may contain `double` or `char` values depending on the particular output of the method: some authentication methods methods output binary vectors, whereas others output real vectors.

Example: the 3rd individual `dbi(3)` in the database `dbi` with the structure above could be

```
dbi(3).name='joe'
dbi(3).authenticated=1
dbi(3).file_list(1).name='joe1.jpg'
dbi(3).file_list(1).path = '/tmp/'
dbi(3).file_list(1).type = 'face'
dbi(3).file_list(2).name='joe2.jpg'
dbi(3).file_list(2).path = '/tmp/'
dbi(3).file_list(2).type = 'face'
dbi(3).file_list(3).name='hand1.jpg'
dbi(3).file_list(3).path = '/tmp/'
dbi(3).file_list(3).type = 'hand'
dbi(3).file_list(4).name='joe1.jpg'
dbi(3).file_list(4).path = '/tmp/'
dbi(3).file_list(4).type = 'wav'
dbi(3).hash_list(1).method_name='philips_method'
dbi(3).hash_list(1).h_value='adsfdasbasdfsdsafsa'
dbi(3).hash_list(2).method_name='mihcak_method'
dbi(3).hash_list(2).h_value='qqvx&3242rew'
```

Notice that two hash string values are associated to this individual, corresponding to the output of the corresponding functions in the library of hashing/feature extraction methods (see next section). The `dbi` variable is duly stored in the MySQL database.

3.2. Library of monomodal methods

The basic structure of entries in this library is:

```
struct(
  'method_name',{},
  'media_type',{},
  'hash_function',struct(
    'name',{},
    'parameters_list',{}
  )
  'comp_function',struct(
    'name',{},
    'parameters_list',{}
  )
)
```

As discussed in Section 2.1, every monomodal method will have a hash function and a comparison function associated. The benchmark accepts functions whose prototype for the acquisition is

```
string h_value = function hash_f(string file,
     parameters)
```

and for the comparison

```
[boolean decision,double reliability] = function
     comp_f(string h_value1,string h_value2,parameters
     ).
```

If `decision=1` then the hash strings `h_value1` and `h_value2` match according to the comparison function, whereas `decision=0` means they do not. The `reliability` parameter ranges indicates how good the decision is.

Example: the 2nd method in a monomodal library `mml` with the structure above could be:

```
mml(2).method_name='philips_method'
mml(2).media_type='audio'
mml(2).hash_function.name='philips_hash'
mml(2).hash_function.parameters_list={0.37,0.95}
mml(2).comp_function.name='philips_comp'
mml(2).comp_function.parameters_list=.9
```

The files `philips_hash.m` and `philips_comp.m`, which must be in the path, implement the corresponding acquisition function

```
h_value = function philips_hash(file,frame_size,
     overlap),
```

and comparison function

```
[decision,reliability] = function philips_comp(
     h_value1,h_value2,threshold).
```

The `mml` array variable is stored in the MySQL database.

3.3. Library of multimodal methods

The basic structure of entries in this library will be:

```
struct(
  'method_name',{},
  'monomodal_methods_list',{},
  'comp_weights',{},
  'attack_list',{}
)
```

The generation of a multimodal hash entails the execution of all the monomodal methods whose names are listed in `monomodal_methods_list` on all corresponding file types of a given individual (image, audio). This generates a series of monomodal identifiers which are incorporated into the structure in Section 3.1.

As discussed in Section 2.2, the comparison of multimodal identifiers requires an overall function in order to break ties between two (or more) monomodal comparison functions (e.g. two monomodal methods that are fused into a multimodal one can give contradictory decisions when using the monomodal comparison functions). According to that discussion we implement this function using the `reliability` parameter furnished by monomodal comparison function, and using a set of weights `comp_weights`. This set is a list of values between 0 and 1 that adds up to 1; each value corresponds to a function in `monomodal_methods_list`, in order to weight the importance of the monomodal methods in the overall comparison. The multimodal decision will be 1 if the weighted sum of monomodal reliabilities is greater than 0.5, and 0 otherwise (note that we have mapped for convenience $\{+1, -1\}$ to $\{1, 0\}$ with respect to Section 2.2).

Example: the 1st entry in the multimodal library `MM1`, with the structure described above, could include two methods from the monomodal library. The first method was described above. Let us assume that the second method is of `media_type='image'`.

```
MM1(1).method_name='MM_first'
MM1(1).monomodal_methods_list={'philips_method','
    mihcak_method'}
MM1(1).comp_weights={.45,.55}
MM1(1).attack_list={'gaussian','random'}
```

The `MM1` array variable is stored in the database. The overall comparison for the multimodal function `MM_first` will be 1 if (cf. Eq. 2)

```
r_1*comp_weights(1)+r_2*comp_weights(2)>0.5
```

where `r_1`,`r_2` are the reliabilities given by the comparison functions of the two monomodal methods.

3.4. Library of attacks

The basic structure in this case is

```
struct(
  'media_type',{},
  'attack_function',struct(
    'name',{},
    'parameters_list',{}
  )
)
```

Each element `parameters_list(i)` is a triplet indicating a range {`starting_value`,`step`,`end_value`}. The prototype of an attack function is

```
string attacked_file = function attack_function_name(
     string file,parameters)
```

where `file` is the full patch of a file of type `media_type`.

Example: a simple unintentional attack can be Gaussian noise addition on audio (or image) files. For instance, assume that the first element `atl(1)` in the array of attacks `atl` with the structure above implements Gaussian noise addition for audio files:

```
atl(1).media_type='audio'
atl(1).attack_function.name='g_noise'
atl(1).attack_function.parameters_list={{.5,.1,2}}
```

The function `g_noise.m` which must be in the execution path will have a header

```
attacked_file = function g_noise(file,power)
```

More complex attack functions can be defined after this type of simple attacks is properly implemented.

The `atl` array variable will be stored in a MySQL database and interfaced to the Matlab code.

3.5. Library of scripts

Benchmark scripts undertake simulations of the effect of attacks on the performance of multimodal methods, relying on the database of individuals and on the multimodal and attacks libraries. Scripts are implemented as loops sweeping the parameter range of a given attack, while computing the rates (i.e., empirical probabilities) of miss/false alarm when using a given multimodal method and attack:

- The rate of miss is computed as the percentage of authenticated individuals not correctly matched.
- The rate of false alarm is computed as the percentage of non-authenticated individuals (incorrectly) matched to authenticated individuals.

In order to simplify the GUI implementation, the structure of benchmark scripts is defined by templates. For the creation of a new script, a list of predefined templates is offered to the user. Upon choosing a multimodal method and suitable attacks from the corresponding lists, a script is created based on the template chosen. The newly created script is stored in the library of scripts. The basic structure to add a script to the library is

```
struct(
  'script_name',{},
  'template_name',{},
  'script_path',{},
  'run_status',{},
  'multimodal',{}
)
```

A resettable Boolean variable indicates whether the script has been run by the benchmark already.

`script_path` gives the full name of the `.m` benchmark script file and `run_status` indicates whether the script hasn't been run yet, it is currently running, or it has been run. The output of the script will be found by default in a file with extension `.output.mat`, with the same name without extension as `script_path`. The output file containing the results from running the benchmarking script is timestamped and included in the database.

Example: the pseudocode of a script template may be:

```
- acquire 'multimodal hash'
for all individuals for all authenticated individuals
     in database
  for all 'ranges' of 'attack'
    - 'attack' individual
    - compute 'multimodal hash' of attacked individual
      for all hashes in the library
      - 'compare hash' with attacked hash
      - compute rate of miss
      end
  end
end
```

Using this particular template, the creation of a benchmark script would require to fill in the terms in inverted commas, that is, basically the multimodal method and the attack from the corresponding libraries. Templates will be Matlab files with dummy strings placed where the functions or parameters must be filled in.

For instance, the first method in the variable `scl`, containing the scripts library with the structure defined above, could be

```
scl(1).script_name='gaussian'
scl(1).template_name='template_1'
scl(1).script_path='/home/scripts/gaussian_script.m'
scl(1).run_status=2
scl(1).multimodal='newhand_face'
```

The output of this script will be found by default in the file `gaussian_script.output.mat`. The `scl` array variable is stored in the MySQL database.

3.5.1. Output module

Completed tasks will allow the user to plot the output resulting from running the benchmark script. The output file will store a fixed structure that will allow the output module to produce plots. It is the responsibility of the template to produce the right output file. This output file will contain a structure variable called `output` with the following form:

```
struct(
  'plot_list',struct(
    'xlabel',{},
    'ylabel',{},
    'title',{},
    'x',{}
    'y',{}
  )
)
```

Note that the vectors `plot_list.x` and `plot_list.y` must have the same size. An output plot will typically show ROC plots (probability of false alarm versus probability of detection), or these probabilities for different thresholds or noise levels.

Text reports about the benchmarking results are also produced. A text report may include details such as functions and parameters used, number of iterations, database signals used, and quality measures obtained.

Example: In `gaussian_script.output.mat` we may find the structure

```
output.plot_list(1).xlabel='Probability of Miss'
output.plot_list(1).ylabel='Noise Variance'
output.plot_list(1).title='Gaussian Additive Noise'
output.plot_list(1).x=[0.1 0.2 0.3 0.4 0.5]
output.plot_list(1).y=[0 0.01 0.05 0.075 .1]
```

More than one plots may be found in `plot_list`, and the user should be able to browse all of them.

3.6. Workflow

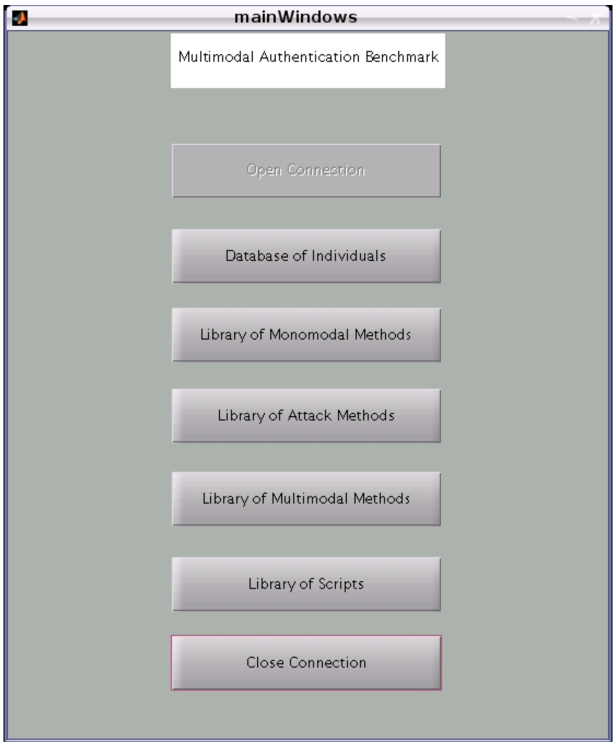

mainWindows
Multimodal Authentication Benchmark
Open Connection
Database of Individuals
Library of Monomodal Methods
Library of Attack Methods
Library of Multimodal Methods
Library of Scripts
Close Connection

Figure 2: *Main window of the benchmark GUI.*

The main benchmark window in Figure 2 features several buttons which allow access to the subwindows which are described next as well as providing the interface for connecting and disconnecting the GUI from the database. The windows were designed keeping simplicity in mind and using the `guide` tool of Matlab. This tool generates a standard `.m` file associated to each window (`.fig` file). This file can be edited in order to implement the callback functions required by the buttons and other window objects.

3.6.1. Database of individuals window

The interface allows the user to:

- Browse, add and remove audio clips associated with each face image (these face images and audio clips must come in pairs).
- Generate hashes for images and audio clips as they are added to the database.

3.6.2. Library windows

These windows allow the user to browse the corresponding libraries and to add and remove functions. The libraries of monomodal methods and attacks accept names of external Matlab functions, whose headers we have defined above. It is also possible to enter the desired parameters for these functions. The library of multimodal methods accepts combinations of functions in the library of monomodal methods and an associated weight and attack function for each of these monomodal methods.

The libraries follow the structures defined in Section 3.

The library of scripts also allows the user to

- Generate a new script using an existing template and multimodal function.
- Run one script or all of the scripts, preferably as background processes. The window displays the `run_status` of each script - 0 if not run, 1 if currently running and 2 if run.
- Plot the outputs of scripts with `run_status=2`. Plots are generated from the files `*.output.mat` as described in Section 3.5.1.
- Generate a report detailing the inputs and outputs of the script e.g. the multimodal, monomodal and attack functions used.

4. METHODS AND FUNCTIONS IMPLEMENTED

In this section we briefly review the features of the methods and attacks that were implemented in order to test the benchmark capabilities.

4.1. Monomodal methods

4.1.1. Image Hashing

- Iterative Geometric Hashing [6]. Two algorithms are proposed. The first one (algorithm A) initially shrinks the input while keeping its essential characteristics (low frequency components). It is recommended in [6] to use to this end the discrete wavelet transform (DWT). However, a three-level DWT takes quite a long time in Matlab. Instead, we shrink the image linearly. Next, geometrically significant regions are chosen by means of simple iterative filtering. The reason for keeping geometrically strong components while minimizing geometrically weak ones is that a region which has massive clusters of significant components is more resilient to modifications. The second algorithm proposed in [6] (algorithm B) simply applies algorithm A on pseudorandomly chosen regions of the input.
- NMF-NMF-SQ. This algorithm is based on a dimensionality reduction technique called *nonnegative matrix factorization* (NMF) [7]. The NMF method uses nonnegative constraints, which leads to a parts-based representation of the input. The algorithm implements a two-stage cascade NMF, because it is experimentally shown in [7] that this serves to significantly enhance robustness. After

obtaining the NMF-NMF hash vector, a statistics quantization (SQ) step is undertaken in order to reduce the length of the hash vector.

- PRSQ (Pseudo-Random Statistics Quantization). This algorithm is based on the assumption that "the statistics of an image region in a suitable transform domain are approximately invariant under perceptually insignificant modifications on the image" [7]. After shrinking the input (i.e., obtaining its low-frequency representation), a statistic is calculated for each pseudo-randomly selected and preferably overlapping subregions of the gist of the input. Scalar uniform quantization on the statistics vector yields the final hash vector.

4.1.2. *Audio Hashing*

If we assume that the conditions are such that a speaker is able to approximately repeat the same utterance (as when a fixed text is read aloud), then audio hashing algorithms can be used for identifying voice clips.

- Microsoft Method [8] (also known as Perceptual Audio Hashing Algorithm). It computes the hash value from robust and informative features of an audio file, relying on a secret key K (seed to pseudorandom generators). An algorithmic description is given below:

  1. The input signal X is put in canonical form using the MCLT (Modulated Complex Lapped Transform) [9]. The result is a time-frequency representation of X, denoted by T_X.
  2. A randomized interval transformation is applied to T_X in order to estimate statistics, μ_X, of the signal.
  3. Randomized adaptive quantization is applied to μ_X yielding $\hat{\mu}_X$.
  4. The decoding stage of an error correcting code is used on $\hat{\mu}_X$ to map similar values to the same point. The result is the intermediate hash, h_X.

  The estimation of the signal statistics is carried out using Method III (see [8]), which relies on correlations of randomized rectangles in the timefrequency plane. For perceptually similar audio clips, estimated statistics are likely to have close values, whereas for different audio clips they are expected be different. The method applies frequency cropping to reduce the computational load, exploiting the fact that the Human Auditory System cannot perceive frequencies beyond a threshold.

- Boğaziçi Method [5]. This algorithm exploits the time-frequency landscape given by the frame-by-frame MFCCs (mel-frequency cepstral coefficients) [10]. The sequence of matrices thus obtained are further summarized by choosing the first few values of their singular value decomposition (SVD) [5]. The actual cepstral method implemented is an improvement on [11].

- Philips Fingerprinting [12]. This method is an audio fingerprinting scheme which has found application in the indexing of digital audio databases. It has proved to be robust to many signal processing operations. The method is based on quantizing differences of energy measures from overlapped short-term power spectra. This staggered and overlapped arrangement allows for excellent robustness and synchronization properties, apart from allowing identification from subfingerprints computed from short segments of the original signal.

4.1.3. *Hand Recognition*

The benchmark includes one algorithm for recognition of hands, based on [13]. The algorithm takes as input images of hands captured by a flatbed scanner, which can be in any pose. In a pre-processing stage, the images are registered to a fixed pose. To compare two hand images, two feature extraction methods are provided. The first is based on measuring the distance between the contours representing the hands being compared, using a modified Hausdorff distance. The second applies independent Component Analysis (ICA) to the binary image of hand and background.

4.2. Attack functions

4.2.1. *Image Attack Functions*

- Random Bending Attack. This attack distorts the image by modifying the coordinates of each pixel. A smooth random vector field is created and the pixels are moved in this field. The vector field must be smooth enough so that the attacked image is not distorted too much. An iterative algorithm is applied to create the horizontal and vertical components of the vector field separately. In each iteration, a Discrete Cosine Transform (DCT) is applied and high frequency components removed. The attack function is designed for grayscale images; color images are tackled using the luminance. The parameters of the attack are the strength of the vector field, the cutoff frequency for the DCT filtering, the maximum number of iterations, and a smoothness threshold.

- Print Scan Attack. Floyd and Steinberg's [14] error diffusion algorithm is applied to transform each of the components of a color image to bilevel values (0 or 1). The algorithm processes the pixels in raster order. For each pixel, the error between the bilevel pixel value and the image pixel value is diffused to the surrounding unprocessed pixel neighbours, using the diffusion algorithm. After processing all pixels, the image is filtered by an averaging filter.

- Contrast Enhancement. This function increases the contrast of the input image using the `histeq` histogram equalization function of Matlab. An input parameter specifies a number of discrete levels N, and the pixel values are mapped to these levels to produce a roughly flat histogram. Histogram equalization is applied separately to the three components of a color image.

- Rotation and Crop Attack. This function rotates the input image by a specified angle, relying on a specified interpolation method. Because we include crop in `imrotate` function we just have the central portion of the rotated image in the output. The input parameters are the rotation angle and the interpolation type (bilinear, nearest neighbor or bicubic interpolation).

- Noise Attack. This function adds noise of a specified variance to the input image using the `imnoise` function of Matlab. Four different types of noise are supported, namely Gaussian noise, Poisson noise, salt & pepper noise, and speckle noise.

- Simple Chimeric Attack. An image is pseudo-randomly selected from the database and a weighted average of the image with the input image is created, using weights given as input to the attack function. The two images are not registered before the averaging, and hence the resulting image does not correspond to a true morphing of the images. Nevertheless, if the weight of the randomly selected image is sufficiently strong in comparison to that of the input image, it can be expected that this attack may be useful to benchmark recognition algorithms.

4.2.2. Audio Attack Functions

Three audio attacks have been included in the benchmark:

- Noise Attack. This attack adds noise to the audio signal. The strength of the added noise is determined by an input parameter that represents the signal to noise ratio. Another parameter specifies the distribution of the noise, which can be Gaussian, uniform, Gamma or Rayleigh.
- Delay Attack. The audio signal $x(t)$ is summed with a delayed version of itself, $x'(t) = x(t)+\alpha x(t-t_0)$ which produces an echo effect. The delay t_0 and the weight α of the delayed signal are input parameters.
- Pitch Bending Attack. This attacks modifies the pitch of the audio signal without changing its duration. Firstly, the length of the audio signal is changed while retaining the original pitch, using the Matlab toolbox "phase vocoder" [15]. Then the signal is decimated/interpolated in order to recover the original length, what changes the pitch. An input parameter determines whether the pitch is compressed or stretched.

4.3. Templates

Two basic templates are defined in the prototype of the benchmark:

- The one described in the example in Section 3.5
- A template able to generate and test attacks that generate modified/chimeric characters. It is essentially the same as the previous one, but, as it generates non-authenticated individuals, it focuses on computing the probability of false alarm instead of the probability of miss.

```
- acquire 'multimodal hash' for all individuals
for all authenticated individuals in database
  for all 'ranges' of 'attack'
    - 'attack' individual (i.e. non-
        authenticated individual)
    - compute 'multimodal hash' of attacked
        individual
    for all hashes in the library
      - 'compare hash' with attacked hash
      - compute rate of false alarm
    end
  end
end
```

5. MULTIMODAL DATABASE

Data was collected from 92 subjects. For each individual, six face images were collected, two scans of the palm of the left hand and three videos. Subjects stood in front of a blue background canvas and the room was illuminated by artificial light. The face images were taken with the subject's head in the following positions:

1. Directly facing the camera;
2. Directly facing the camera, with glasses removed, if subject was wearing glasses in the first picture;
3. Head turned $-90°$;
4. Head turned $-45°$;
5. Head turned $+45°$;
6. Head turned $+90°$.

The three videos recorded the subject carrying out the following activities:

1. Reading a fixed text;
2. Performing four gestures;
3. Moving head while speaking freely.

The only part of the video data used in the multimodal benchmark is the audio portion of the first video. In this audio signal, the subject is recorded reading a fixed English text of around 100 words. The participants consisted of an international group of mainly non-native English speakers. Subjects were asked to ignore any mistakes made in pronunciation and to continue reading to the end of the text. The final part of the text is the numbers one to ten, which were read twice. The rest of the video data was collected for use in another eNTERFACE project incorporating gesture recognition.

6. TESTS

6.1. Database of Libraries

The database consists of both the data collected and libraries of the various monomodal, multimodal and attack functions. It has been written as a simple relation database. The database has been integrated into the Matlab code by means of a Mex file in which a number of standard SQL commands have been defined e.g. `insert`, `select`. The benchmark is capable of running, and has been tested, under Linux, Mac OS and Windows.

6.2. Testing & Debugging

To illustrate the use of the benchmark, a multimodal identifier was created through the benchmark consisting of a face identifier and a hand identifier, both constructed using the iterative geometric image hashing algorithm A. Each monomodal method was set to report a match when the reliability of the comparison function was above a threshold of 0.8. The face identifier was weighted 0.6 in the multimodal combination. In a benchmark script, a print scan attack was applied to both the face and hand images, varying the window size of the averaging filter and tested on a small database of 5 individuals. The script produced a report containing the probability of a miss i.e. the probability that the multimodal identifier failed to correctly identify the individual, for different window sizes. Note that since there are 6 face images associated with each individual and two palm scans, 12 combinations of face and hand images could be tested per person. The script output the probability of mis-identification which is plotted in Figure 3.

The output plot in Figure 4 shows the results of simulating the effect of rotation and crop on a multimodal fusion using algorithms A and B in Section 4.1.1, applied to face images and hand images respectively. The database of 92 individuals was used. We see here that the benefit of the multimodal method over either of the modalities on its own.

7. CONCLUSIONS AND FUTURE WORK

Although the basic structure of the benchmark is fully functional, some issues have inevitably arisen during the short period of time allowed for the development of this project. The main issue faced at the end of the workshop were the computational problems posed by the Matlab implementation of the methods. The computational burden associated to the amount of iterations within a benchmarking script may pose difficulties to complete

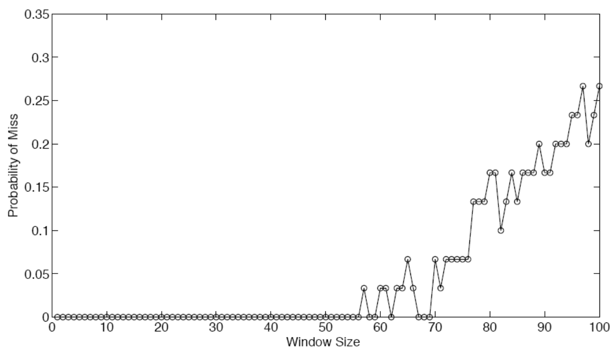


Figure 3: *Probability of a miss for a multimodal face and hand identifier under the print & scan image attack.*

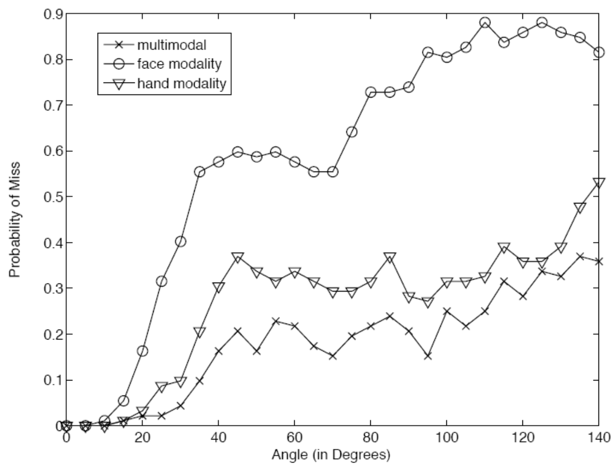


Figure 4: *Probability of a miss for a multimodal face and hand identifier under the rotation and crop attack.*

the simulations within a reasonable amount of time unless the methods used are optimized.

In a similar fashion as other benchmarks [16], it would be interesting to endow this benchmark with a certification procedure. This procedure would entail defining a fixed set of attacks – obviously dependent on the modalities – and benchmarking scripts. The report obtained from this certification procedure would be used to rank methods in a more systematic way. However establishing what type of attacks and scripts will be include can be controversial, as the rankings obtained may be biased if they are not carefully chosen.

8. ACKNOWLEDGEMENTS

The members of project #12 thank Prof. B. Sankur for contributing the source code of the hand feature extraction method developed by himself and his group.

This report, as well as the source code for the software developed during the project, is available online from the eNTERFACE'07 web site: `http://www.enterface.net`

9. REFERENCES

[1] F. Ahmed and M. Siyal, "A secure biometric authentication scheme based on robust hashing", in *Proceedings of the 5th International Conference on Information, Communications and Signal Processing*, (New York, USA), pp. 705–709, Dec 2005. 147

[2] Y. Sutcu, H. T. Sencar, and N. Memon, "A secure biometric authentication scheme based on robust hashing", in *Procs. of the 7th workshop on Multimedia and security*, (New York, USA), pp. 111–116, October 2005. 147

[3] A. Ross and A. Jain, "Information fusion in biometrics", *Pattern Recognition Letters, Special issue: audio- and video-based biometric person authenticaion (AVBPA 2001)*, vol. 34, pp. 2115–2125, September 2003. 147

[4] F. A. Petitcolas, "Watermarking schemes evaluation", *IEEE Signal Processing*, vol. 17, pp. 58–64, September 2000. 147

[5] H. Özer, B. Sankur, N. Memon, and E. Anarım, "Perceptual audio hashing functions", *EURASIP Journal on Applied Signal Processing*, no. 12, pp. 1780–1793, 2005. 148, 152

[6] M. K. Mıhçak and R. Venkatesan, "New iterative geometric methods for robust perceptual image hashing", in *Procs. of ACM Workshop on Security and Privacy in Digital Rights Management*, (Philadelphia, USA), 2001. 151

[7] V. Monga and K. Mıhçak, "Robust image hashing via non-negative matrix factorizations", in *IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 2, pp. 14–19, May 2006. 151, 152

[8] M. K. Mıhçak and R. Venkatesan, "A perceptual audio hashing algorithm: A tool for robust audio identification and information hiding", in *Procs. of the 4th Information Hiding Workshop*, vol. 2137 of *Lecture Notes in Computer Science*, (Pittsburgh, USA), pp. 51–65, Springer, April 2001. 152

[9] H. S. Malvar, "A modulated complex lapped transform and applications to audio processing", in *IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 3, pp. 1421–1424, March 1999. 152

[10] L. R. Rabiner and R. W. Schafer, *Digital Processing of Speech Signals*. Prentice Hall, 1978. 152

[11] B. Logan, "Mel frequency cepstral coefficients for music modelling", in *Procs. of International Symposium on Music Information Retrieval*, (Indiana, USA), October 2000. 152

[12] J. Haitsma, T. Kalker, and J. Oostveen, "Robust audio hashing for content identification", in *Procs. of the International Workshop on Content-Based Multimedia Indexing*, (Brescia, Italy), pp. 117–125, September 2001. 152

[13] E. Yoruk, E. Konukoglu, B. Sankur, and J. Darbon, "Shape-based hand recognition", *IEEE Trans. Image Processing*, vol. 15, pp. 1803–1815, July 2006. 152

[14] R. Ulichney, *Digital Halftoning*. The MIT Press, 1987. 152

[15] D. P. W. Ellis, "A phase vocoder in Matlab", 2002. `http://www.ee.columbia.edu/~dpwe/resources/matlab/pvoc`. 153

[16] J. Vorbruggen and F. Cayre, "The Certimark benchmark: architecture and future perspectives", in *IEEE Intnl. Conf. on Multimedia and Expo*, vol. 2, pp. 485–488, 2002. 154

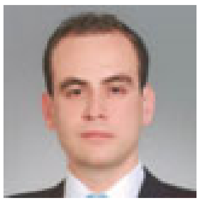
10. BIOGRAPHIES

Morgan Tirel is a M.Sc. student at the University of Rennes, France.
Email: morgan.tirel@etudiant.univ-rennes1.fr

Ekin Olcan Şahin received in 2006 his B.Sc. in Electrical Electronics Engineering at Bilkent University, Ankara, Turkey. Currently he is a M.Sc. student at the Electrical & Electronics Engineering Department of Boğaziçi University, Turkey.
Email: ekin.sahin@boun.edu.tr

Guénolé C. M. Silvestre received the M.Sc. degree in electronic and electrical engineering in 1993. In 1996, he received the Ph.D. degree from the University of Dublin, Trinity College, Ireland, for his work in silicon-on-insulator materials. As a post-doctoral fellow with Trinity College, Dublin, he pursued research on digital image processing and watermarking. In 1997, he was appointed Research Scientist at the Philips Research Laboratories, Eindhoven, The Netherlands, and his research focus switched toward the study of polymer light-emitting materials. In 1999, he joined the National University of Ireland (University College Dublin), where his present research activities lie in the area of digital communications, data-hiding, and signal processing. Dr. Silvestre was the 1995 recipient of the Materials Research Society Graduate Student Award.
Email: guenole@ihl.ucd.ie

Clíona Roche received her Bachelor of Arts (Hons) in Computer Science (major) and History of Art, at University College Dublin (UCD), Dublin, Ireland. Currently she is a Ph.D. student in Computer Science at the same university on the topic of High throughput comparative modelling of protein structure by machine learning.
Email: cliona.roche@ucd.ie



Kıvanç Mıhçak was born in Turkey in 1974. He received his B.S. degree from Electrical and Electronics Engineering Department, Bilkent University, Ankara, Turkey in 1996 and the M.S. and Ph.D. degrees from Electrical and Computer Engineering Department, University of Illinois, Urbana-Champaign (UIUC), in 1999 and 2002 respectively. At UIUC, he was in the Image Formation and Processing Group; his thesis advisors were Pierre Moulin and Kannan Ramchandran. Between 2002 and 2005, he was a researcher, with the Cryptography & Anti-Piracy Group at Microsoft Research, Redmond, WA. Currently, he is assistant professor with the Electrical and Electronic Engineering Department of Boğaziçi University.
Email: kivanc.mihcak@boun.edu.tr

Sinan Kesici was born in 1984, in Erzincan, Turkey. He is currently an undergraduate student in Boğaziçi University, Turkey. He is a senior student in Electrical & Electronics Engineering Department. His specialization option is Telecommunication Engineering.
Email: sinan940@yahoo.com

Neil J. Hurley received the M.Sc. degree in mathematical science from University College Dublin (UCD), Dublin, Ireland, in 1988. In 1989, he joined Hitachi Dublin Laboratory, a computer science research laboratory at the University of Dublin, Trinity College, from which he received the Ph.D. degree in 1995, for his work in knowledge-based engineering and high-performance computing. He joined the National University of Ireland, University College Dublin, in 1999 where his present research activities lie in the areas of data-hiding, signal processing, secure and robust information retrieval and distributed computing.
Email: neil.hurley@ucd.ie

Neslihan Gerek is a M.Sc. student at Boğaziçi University, İstanbul, Turkey.
Email: neslihan.gerek@gmail.com

Félix Balado graduated with an M.Sc. in Telecommunications Engineering from the University of Vigo (Spain) in 1996, and received a Ph.D. from the same institution in 2003, for his work in data hiding. He then joined the National University of Ireland (University College Dublin) as a post-doctoral fellow at the Information Hiding Laboratory. Previously he worked as a research and project engineer at the University of Vigo, in different research projects funded by the Galician and Spanish Governments, and by the European Union. His research interests lie in the areas of multimedia signal processing, data hiding, and digital communications.
Email: fiz@ihl.ucd.ie