DRAFT - DO NOT PUBLISH!

A Fingerprint Matching Algorithm For Programmable Smart Cards

Tommaso Cucinotta
Riccardo Brigo
Marco Di Natale

Scuola Superiore Sant'Anna

Abstract

This paper presents a hybrid fingerprint matching algorithm for user authentication based on the fusion of heterogeneous schemes, and designed to run on programmable smart cards. The approach is based on the well known texture vector and minutiae based techniques, where image processing and feature extraction occur on the host, while the card device performs the final match against the onboard template, which is never revealed to the outside world. This increases the security of the template itself and of the applications using it.

The matching algorithms have been tuned in order to achieve an acceptable performance despite computation and memory constraints. Experimental results, gathered from our implementation on a Java Card device, highlight the feasibility of the hybrid approach. Furthermore, they show to what extent it is possible to trade precision for speed in the verification process, using appropriate tuning of the on board matching parameters.

Introduction

Nowadays security is a fundamental requirement in software design, and authentication of users is one of the crucial issues to be addressed to meet this requirement. Most systems still rely on traditional password based authentication, where users authenticate to a system by proving knowledge of some secret information. Unfortunately, a password may be easily revealed to, obtained or guessed by unauthorized users, resulting in a possibly very low security level. Smart card technology, especially when used in conjunction with public key cryptography, allows a secret based authentication mechanism where it is technologically impossible, or at least unfeasible, to reveal the secret to third parties. A two step authentication system is usually deployed, where the user must prove possession of his own card and, optionally, knowledge of a *Personal Identification Number* (PIN), which is used to protect access to functionality of the card. In practice, the method grants authentication of the plastic card itself, not of the user and, despite the increase in the security level, a card may still be given to or stolen by unauthorized users.

Biometrics technology promises a final solution to this problem, letting a user authenticate himself to a system by showing some unique biological characteristics of his own body, such as fingerprint ridges, hand shape or retina. When using both biometrics and smart card technology, it is possible to realize a three factor authentication, where the user is required to prove, at the same time, knowledge of a secret information, possession of a physical device, and through the biometric, his presence.

This work focuses on solutions where the main authentication mechanism employs the cryptographic capability of a Java Card device, which can be accessed only after the user authenticates by means of a successful fingerprint verification process. PIN code verification may be implemented in addition to the biometric verification or used as an alternative. A possible application scenario is a secure application running entirely or in part on a smart card. An example is a smart card based digital signature, where the non repudiation property, usually established only at a legal level by dictating card owner liability, can be enforced by technology requiring biometrics authentication.

Specifically, our study is targeted at exploring the feasibility of fingerprint recognition on programmable smart card devices, through the use of a hybrid technique based on two well known algorithms: *fingercode* and *minutiae based matching*. The algorithm design aims at achieving an acceptable performance for this application context, while keeping complexity low enough to make an implementation onto a programmable card device feasible. We do not discuss issues more strictly related to the security of the fingerprint acquisition process itself, such as how to protect against the dead finger and residual print attacks, which, although important, are beyond the scope of this paper.

Before entering into the technical details of our approach the next section introduces the basic concepts about fingerprint verification and the related literature. The section *Matching Algorithm* features an overview of the proposed technique, with a detailed description of the feature extraction procedure and of the matching algorithm. Evaluation results for the proposed algorithm are reported in *Results* and specific notes about the algorithm implementation are reported in the section *Implementation Notes*. Finally, the section *Conclusions and Future Work* draws conclusions and presents possible areas of future investigation.

Background

Fingerprints are a traditional way of identifying people, because they are a permanent distinctive feature of each and every person. Things like cuts and bruises usually cause only minor changes in their structure over a subject's lifetime. A fingerprint is characterized by a series of almost parallel ridges, possessing both a global and a local structure. The global structure refers to one of five different kinds of possible ridge classes [4]: *whorl, right loop, left loop, arch* and *tented arch*. Local characteristics refer to ridge endings and bifurcations, also known as minutiae (see figure 1) that are unique to each individual, making minutiae based verification one of the most commonly used means of identification.

Figure 1 - **Example of minutiae detection. Some false minutiae have been detected on the image border.**

A minutiae based live scan fingerprint verification system consists of various components [11]:

- a *fingerprint scanner*, producing a two dimensional, grey level bitmap image representing the fingerprint;
- a *feature extraction module*, which runs a feature extraction algorithm producing a list of fingerprint minutiae; and
- a *matching algorithm*, which is run to compare the extracted minutiae with those corresponding to the person identity to be verified, also called the *template*.

The template is produced in a separate phase, the *enrollment*, during which the user is registered into the system, along with the representation of his fingerprint, after proper identity verification is performed by an official.

The feature extraction procedure works in two steps: first, it produces a set of claimed minutiae, which typically also includes false minutiae. For example, two ridge endings might be detected near a bruise on a ridge, causing two minutiae very close to each other to be detected. A minutiae selection algorithm is then needed to identify and discard false minutiae. The matching algorithm is usually based on point pattern matching techniques, in order to be tolerant to translations and rotations of the live scanned image with respect to the stored template, as well as to distortions caused by the adhesion of the finger surface to the sensor.

Correct use of biometrics and smart cards is not as straightforward as it could seem. Several works explore the possible attacks against a system integrating such technologies. For example, in [7] it is emphasized that biometrics do not provide the expected increase in the security level unless they are properly managed and embedded in a strong cryptographic protocol controlling interactions among the components of the authentication system. The authors also highlight the need to keep the biometrics data as secret as possible, especially in the context of the

''fake organ'' problem, i.e. when the sensor is not able to distinguish between a live organ and a properly designed synthetic one. In [11], after an estimation of the bit strength of a minutiae based matching algorithm, resulting in roughly 40 bits of information for 15 minutiae and in 82 bits for 25 minutiae, a minutiae search brute force attack is presented.

Recently, the European Union has focussed attention on matching on card technologies, as reported in [15], where it is stressed that, in the context of electronic signatures, the possibility of identifying people based on biometric characteristics is of fundamental importance, due to the non repudiation security requirement.

Concerning feature extraction from fingerprint images, [6] presents a good overview of the general structure of automatic fingerprint identification systems (AFIS), emphasizing the main challenges. In [5] the authors demonstrate how an image enhancement algorithm based on Gabor filters can significantly improve the performances of an AFIS by enhancing the reliability and the precision of the minutiae extraction process. In [10] Prabhakar proposes an innovative approach to fingerprint analysis and matching, based on the use of a Gabor filter bank to extract statistical information from the fingerprint image. This approach has been proven to degrade more gently with image quality than classical minutiae based algorithms. The method has been further developed in [12] to achieve acceptable performance even when data is acquired from small sensors providing only a limited portion of the fingerprint image. In the same work, the authors introduce the idea of combining the output of different fingerprint matching algorithms to achieve a better performance despite the high correlation between the results of the individual methods.

The problem of combining two fingerprint matching algorithms to improve performance is finally addressed in [3], where the focus is on the experimental comparison of different methods to combine the scores from different matching algorithms.

With respect to previous investigations on hybrid fingerprint matching, our approach is specifically focussed on the feasibility of the implementation of these techniques onto programmable smart cards. For this purpose, a graph based encoding of the minutiae has been adopted to transfer minutiae data from the host to the card and a careful tuning of the on card matching algorithm has been made, in order to minimize the computation overhead onto the device.

We also present an on card architecture for implementing our matching algorithm, realized as an extension of the protocol and the JavaCard Applet introduced in [1], and we report experimental data gathered from the execution of the proposed algorithm onto a JavaCard device.

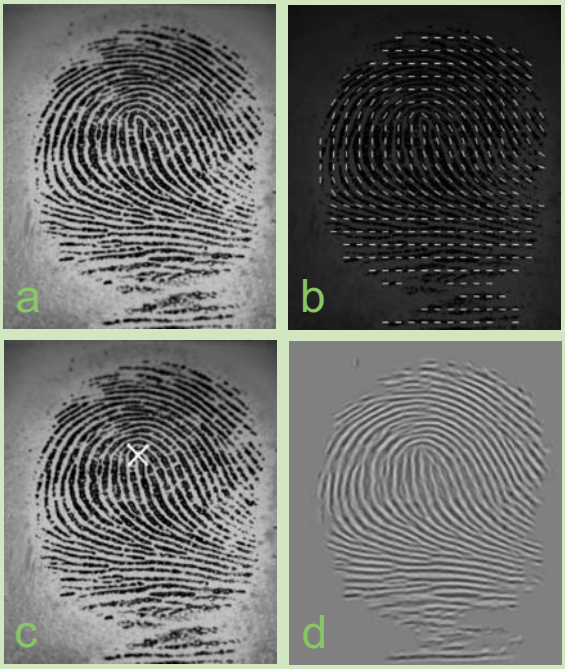

Figure 2 - **Preprocessing phases**
(a) Original fingerprint image
(b) Orientation field
(c) Core detection
(d) Enhancing

Matching Algorithm

This section contains an outline of the algorithms that have been developed to match a fingerprint image against its template definition inside the smartcard. The image undergoes a first preprocessing phase on the host, which enhances the relevant content and reduces the effect of noise, misplacement of the finger on the sensor device and other errors.

After preprocessing, the feature extraction phase takes place and, finally, the match is performed by a purposely defined search algorithm inside the card device. In the following description, some values are expressed in pixels. These are relative to an image scanned at a resolution of 500 dpi.

Preprocessing

The preprocessing phase consists of three sequential steps aimed at improving the quality of the image and detecting a reference core point inside it, which is later used by the subsequent feature extraction and matching phases. These steps are functional to both methods of fingercode and minutiae based matching.

Direction Field

The fingerprint image (figure 2(a)) is first divided into small square blocks. The blocks with a luminance contrast below a given threshold are ignored. Then, the direction of ridges is computed for each block as follows. First, the gradient at each point is computed by applying the Sobel

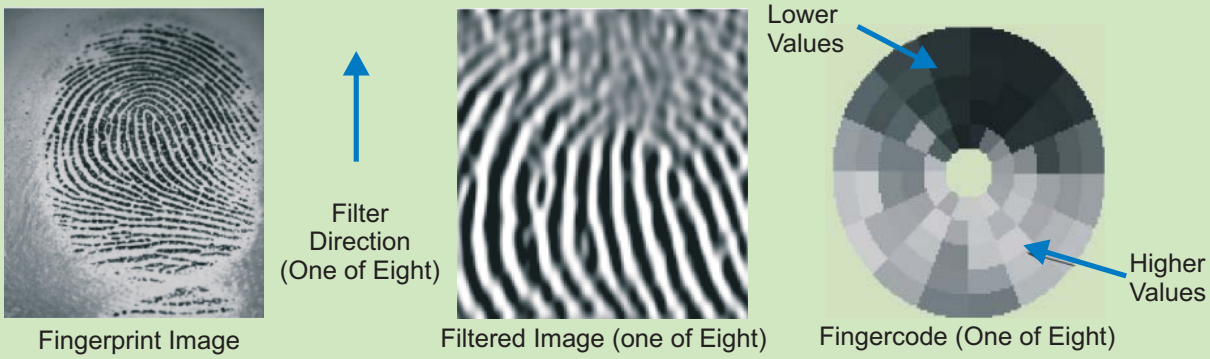

Figure 3 - **Directional Filtering of the Region of Interest**

operator. Then, the vector field obtained by doubling (modulo 360) the gradient angle at each point is averaged and the main direction of the ridges inside each block is computed by halving the angle of the average vector. Finally, in order to reduce the imprecision caused by poor quality blocks, a low pass filter is applied to the obtained ridge direction field (Figure 2(b)).

Reference Point

The next step is finding a reference point (or *core*) inside the fingerprint image (Figure 2(c)). This greatly simplifies the problem of matching the scanned image with the template (i.e. finding the correct rototranslation). The core is defined as the point of maximum curvature of the concave ridges, according to the definition provided in [10].

First, a set of ''candidate'' blocks, possibly containing the core, is determined using a method known as Poincaré index. Given a block B, its Poincaré index P_B is computed as:

$$P_B = \sum_{k=0..7} \left| d_k - d_{k+1 \,(\mathrm{mod}\ 8)} \right|_{-90..90}$$

where $d_0 \ldots d_7$ are the main ridge directions inside the eight blocks adjacent to B, taken in clockwise order, and $|\cdot|_{-90\ldots90}$ is the operator that applies a modulus 180 and represents the result as an angle in the range -90…90. P_B may only assume values, which are multiples of 180°, where a value of +180° denotes the presence of a core inside B or one of its eight adjacent blocks.

Finally, the orientation field is convolved with a filter designed to highlight image portions describing a downward arch: the centre of the block giving the maximum convolution among the candidates is chosen as the fingerprint core.

Image Enhancement

In this stage, each block of the image is approximated by a sinusoidal plane wave of specific direction and frequency. Directions are already known thanks to the orientation field, whilst the frequency is determined for each block by analysing its two dimensional Fourier spectrum and by extracting its dominant component. Given these two parameters, the block can be enhanced using a well tuned Gabor filter [8]. As a result the contrast between ridges and valleys is increased and the background noise (due to the sensing process and to cuts or bruises) is radically reduced (Figure 2(d)).

Features Extraction

The main feature extraction stages required by the two methods of fingercode and minutiae based matching will now be outlined.

The list of minutiae is arranged in a graph structure that is suitable for the on card matching phase, where the matching problem is transformed into a graph similarity problem, exploiting local relationships that are more robust to the finger skin stretching caused by the pressure on the sensor.

Fingercode

The fingercode is extracted from a circular region with a diameter of 120 pixels, centred on the fingerprint core, which Prabhakar defines as the *region of interest*. This region is first normalized to a given mean and variance. This operation removes the luminance variations due to finger pressure differences on the surface of the sensor. The image is then filtered by means of eight Gabor filters (Figure 3 shows the effect of one of them) and subsequently tuned over eight different directions. Each filter enhances the contrast of the ridges parallel to its direction whilst blurring the rest of the image.

The eight images obtained by applying the filters are then tessellated into rings and sectors, and for each tessel the average absolute deviation of luminance is computed: tessels corresponding to high contrast portions of the image feature high values of deviation, while blurred portions have near zero deviation. The deviation values of the 640 tessels (80 for each filtered image, for 8 images), normalized into the range 0…255 and encoded in binary format (one byte for each tessel), constitute the fingercode (Figure 3).

Minutiae Detection
The enhanced image is reduced from gray scale to black and white (Figure 4(a)) and the ridges are eroded down to a single pixel width (Figure 4(b)). For each black pixel its eight neighbours are considered. If only one of them is black the point is an ending, whilst if more than two of them are black the point is a bifurcation. Either way, a minutia m_i is detected and saved as the pair of its position and the angle δ_i of its ridge, obtained from the orientation field: $m_i = (p_i , \delta_i)$. Finally, a set of heuristics are used to remove false minutiae.

Graph Construction
Let $M = \{m_i\}_{0 \le i \le k}$ denote the set of all the detected minutiae, where the core point has been included as m_0. An oriented graph representation of M is built, where each node n_i represents a minutia m_i and an edge from n_i to n_j means that minutia m_j belongs to the neighbourhood of m_i, i.e. it is at a distance between a minimum value d_{min}, set to 30 pixels, and a maximum value d_{max}, set to 100 pixels. Neighbour minutiae of m_i are characterized by their polar coordinates relative to m_i. Minutiae that are too close are not considered because at small distances small errors in the position lead to large variations in the angle. Likewise, minutiae that are too far away are not considered because at large distances the elastic deformation of the skin potentially leads to inaccurate relative positions. Therefore, adoption of the two bounds d_{min} and d_{max} in the construction of a minutia neighbourhood makes it possible to build a representation that is robust with respect to inaccuracies in the detection phase. For each minutia, the algorithm allows up to a maximum number max_{out} of neighbours (*outdegree*) to be represented in the graph as edges. This limits the computation time needed on the card device during the matching phase. A different outdegree limit, max_{out_c}, is used for the core point due to its great influence on the precision and execution time of the on card matching phase, as shown later in the *Results* section. Furthermore, each minutia may be referenced in up to a maximum number max_{in} (*indegree*) of other minutiae after which it is ignored as neighbour of other minutiae. A careful selection of the max_{out} constraint avoids the construction of partial graphs disconnected from the outer minutiae. In our experiments we varied the indegree from 3 to 5 and the outdegree from 5 to 8.

Figure 4 - **Minutiae Extraction**
(a) Binarization
(b) Skeletonization

```
M={m_i}_{i=0…k}            // set of input minutiae
ref[1…k] = {0,…,0}                          // ref[i]
number of incoming edges to m_i
pending = {m_0}            // set of minutiae to process
n = {}                     // set of neighbouring minutia
c = {}                     // set of ignored minutia
nodes = {}                 // set of processed minutia
edges = {}                 // set of edges in the graph

while (pending not empty){
   extract m_i from pending
   add m_i to c
   n={}
   out_count=0
   while (out_count < max_out){
        extract closest m_j from M\(c∪n) s.t. d_min≤d_i,j≤d_max
        if m_j not found then
            break
        add m_i→m_j to edges
        add m_j to n
        ref[j]++
        if (ref[j]==max_in then
            add m_j to c
        if m_j ∉ nodes then
            add m_j to pending
        out_count++
   }
   add m_i to nodes
```

Figure 5 - **Algorithm Building the Graph of Minutiae**

The graph of minutiae is constructed by the algorithm represented as pseudocode in Figure 5.

The representation of a fingerprint, as output by the entire process, consists of the list of minutiae $\{m_i\}$, including, for each minutia, its cartesian coordinates (relative to the core), ridge direction and relative position of its neighbours in polar coordinates. Finally, the upper left and lower right coordinates (relative to the core) of the *bounding box* delimiting the set of minutiae completes the representation.

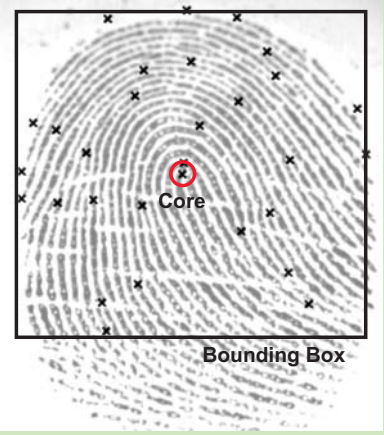

Figure 6 - **Minutiae Encoding. The Bounding Box Delimits the Set of Minutiae Included in the Graph.**

Feature Matching

Fingercode
Given the fingercodes of the two images, the matching score is computed as the first order metric distance between the two vectors of length 640 representing them, i.e. the sum of the absolute differences between the values associated with the corresponding tessels. Fingercode matching tolerates small relative rotations between the images being compared. Larger rotations can be accounted for by searching for the relative rotation (multiple of the tessellation sector angle) between the two fingercodes that achieves the best matching score $score_{fc}$.

In practice, it is quite safe to assume that the user applies the finger almost vertically, so that only three rotation angles need to be tried: no rotation, one sector clockwise and one sector counter clockwise.

Minutiae
The minutiae matching procedure has been inspired by the point pattern matching algorithm described in [9]. The original algorithm is too complex to be practically implemented onto a programmable card device, thus we provide a simplified implementation by exploiting the availability of a common reference point (the core) from which the matching process extends towards the periphery.

Given the graphs organizing the minutiae of the template and of the submitted fingerprint, the basic task of the algorithm is to build a *spanning, ordered tree* touching as many nodes as possible, starting from the two cores and visiting the two graphs by matching edges.

Let $\{m_i^S\}_{i\in\{0\ldots N\}}$ and $\{m_h^T\}_{h\in\{0\ldots K\}}$ denote, respectively, the minutiae of the submitted and template fingerprints. Given a pair of matching minutiae $\{m_i^S\}$ and $\{m_h^T\}$, the vectors $\mathbf{p}_{i,j}^S$ and $\mathbf{p}_{h,k}^T$ represent the position of m_j^S and m_k^T relative to m_i^S and m_h^T respectively. If δ_j^S and δ_k^T are the ridge angles of m_j^S and m_k^T, and α is the angle of mutual rotation between the two sets of submitted and template minutiae, then the two minutiae m_j^S and m_k^T match if:

$$\begin{aligned} &\left\|\mathbf{p}_{i,j}^S\right\| - \left\|\mathbf{p}_{k,h}^T\right\| < d_{th} \\ &\left|\theta(\mathbf{p}_{i,j}^S - \mathbf{p}_{\mathbf{h},\mathbf{k}}^T) + \alpha\right| \bmod 360 < \theta_{th} \quad (1) \\ &\left|\delta_j^S - \delta_k^T + \alpha\right| \bmod 180 < \delta_{th} \end{aligned}$$

where $\theta(\mathbf{p})$ and $\|\mathbf{p}\|$ denote the angle and length of the vector $\mathbf{p}$, d_{th} and θ_{th} are the maximum acceptable tolerance in the length and angle of the relative positions of m_j^S and m_k^T with respect to the matching pair (m_i^S, m_h^T), and δ_{th} is the maximum acceptable difference in the direction of the ridges of m_j^S and m_k^T. The three tolerances have been chosen by performing a statistical analysis over their respective quantities when matching pairs of corresponding fingerprints.

The rotation α is computed in the very first stage of the algorithm as the rotation resulting in the maximum number of matches among the neighbours of the two cores (assumed as the first matching pair). In order to reduce the amount of computation, we limit the search to the set of rotation angles which exactly align a pair of neighbours only, one for each set, and with an absolute value less than 20 degrees, since we assume that the user puts his finger on the sensor almost vertically, with minimal rotation.

The algorithm counts the number of neighbour pairs that match under each possible rotation α. The tolerances $d_{th}^{(core)}$, $\theta_{th}^{(core)}$ and $\delta_{th}^{(core)}$ used for matching the neighbours of the core are less restrictive than the ones used for matching the other pairs in order to account for the possible imprecisions in the localization of the core. If two rotations α_1 and α_2 give the same number of matches, the lower one (in absolute value) is preferred.

Let $\mathbf{C}$ denote the set of matching minutiae pairs, i.e. $(m_k^T, m_j^S) \in \mathbf{C}$ means that a match between m_k^T and m_j^S has been found relative to a pair of parent (in the graph) minutiae. Let *p* and *d* denote the sets, initially empty, of *pending* and *processed* minutiae, in the submitted graph. Then, the algorithm building the spanning tree proceeds as in the pseudocode of Figure 7.

Finally, the number nM_{bbS}^T of minutiae of *T* lying in the bounding box of *S*, and the number nM_{bbT}^S of minutiae of S lying in the bounding box of T are computed, and the minutiae score $score_{min}$ is evaluated as:

$$score_{\min} = 100\frac{numMatches^2}{nM_{bbS}^T \times nM_{bbT}^S}$$

Fusion of Scores
Given the two scores $score_{fc}$ and $score_{min}$, the overall score is computed as a linear combination with weights α and β:

$$score = \alpha score_{fc} + \beta score_{min}.$$

If score exceeds a given threshold th_{score} the system considers the proposed fingerprint to be similar to the enrolled one and the match succeeds, otherwise the match fails. The α and β coeffcients have been determined with an a posteriori analysis as the ones which minimize the overall EER.

```
C = set of matching nodes (m_k^T, m_j^S)
p = set of matching parents m_j^S
d = set of visited nodes
numMatches = size(C)
while (p not empty){
    extract m_i^S from p
    find m_h^T s.t. (m_h^T, m_i^S) ∈ C
    for each pair of matching edges m_i^S → m_j^S, m_h^T → m_k^T {
        add (m_k^T, m_j^S) to C
        if m_j^S ∉ d then
                add m_j^S to p
        numMatches++
    }
    add m_i^S to d
}
output C, numMatches
```

Figure 7 - **Algorithm Building the Matching Spanning Tree**

Results

Given the limited resources available within a smartcard, both in terms of memory and computational power, we are particularly interested in the tradeoff between the overall algorithm discriminatory efficiency and its execution time.

As fingerprint recognition systems are based on an approximate matching of the stored template against live scanned data, they are characterized by a set of tunable parameters that allow the matching phase to be either more restrictive or more permissive. In the first case, the ability to reject impostors is enhanced, but the probability to reject the legitimate user is increased as well. By making the algorithm more permissive, the ability to accept legitimate users is enhanced, but the probability to accept an impostor is increased as well. Therefore, evaluation of these systems is based on the achieved *False Acceptance* and *False Rejection Rates* (FAR and FRR), usually evaluated on a database of biometric data. By varying the acceptance threshold, the set of FAR and FRR pairs obtained may be plotted into a curve called a *Receiver Operating Curve* (ROC). A point of particular interest on this curve is the one corresponding to equal FAR and FRR values, the *Equal Error Rate* (EER) point. These metrics are used for the evaluation of various configurations of the algorithm parameters.

The implementation of the fingercode matching is straightforward and leaves very little space for further simplification or optimization.

The complexity of the minutiae matching algorithm depends on the value of the three parameters defining the structure of the oriented graph produced by feature extraction: the maximum outdegree $\max_{out_c}$ of the core node, and the maximum indegree max_{in} and outdegree max_{out} of all the other nodes.

We conducted experiments in order to quantify the influence of these three parameters (table 1 summarizes the adopted configurations), analysing a database of 480 images (48 fingerprints, each scanned ten times).

The fingerprint scanner used in the experiments is the FX2000 USB by Biometrika s.r.l. [17], providing a portable development kit and API for access to the acquired biometrics data. The reference development platform is a RedHat 7.3 Linux system. The Java Card is a Cyberflex Access 32K from Schlumberger.

Figure 8 represents the effect of the variation of the $\max_{out_c}$ parameter over the ROC curve, while table 2 details, for each configuration, the EERs obtained (both by the minutiae only match and by the hybrid one) and the average execution times on the smartcard, fractioned into their three main components: the load time spent to send the fingerprint representation to the card

Configuration	$\max_{out_c}$	$\max_{out}$	$\max_{in}$
base	5	5	3
core6	6	5	3
core7	7	5	3
core8	8	5	3
min6	5	6	3
min7	5	7	3
min8	5	8	3
ref4	5	8	4
ref5	5	8	5
ref6	5	8	6
ref7	5	8	7
ref8	5	8	8
max	8	8	6

Table 1 - **Configurations of Parameters Compared, for the Minutiae Based Algorithm.**
The Configurations Named *base* and *max* are Given as Lower and Upper References

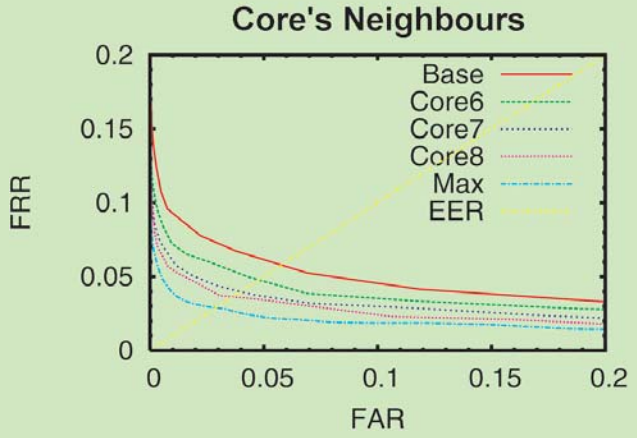

Figure 8 - **ROC Curves for Various Values of the max_{out_c} Parameter**

and to initialize the algorithm, and the time required respectively by the fingercode and minutiae matching phases. Note that the fingercode time is not affected by the parameters configuration, and its variation in the table is due to inaccuracy in experimental measurements.

The setup time doesn't appear to be significantly influenced by the parameters chosen and, consequently, by the size of the fingerprint representation (which ranges from about 1650 bytes for `base` configuration to about 2300 for `Ref8`). This leads to the conclusion that about 2 seconds are required by the card's operating system just to initialize itself. We assume this delay is card dependent and outside of our control.

The experiments indicate max_{out_c} as the most critical parameter of the minutiae matching algorithm, determining the greatest excursion both in time (from 4 to 6.38 seconds) and precision (from 5.5% to 3.7% of EER). The parameter plays a significant role at the very beginning of the algorithm, where the cores' neighbours are used to determine the best mutual rotation between fingerprints. An error in its estimation drastically increases the probability of a failed match between corresponding fingerprints and, consequently, raises the algorithm's FRR.

Larger values of max_{out} provide a better estimate of the mutual rotation; on the other hand, limiting the search domain to rotations lower than a given threshold only does not speed up the search.

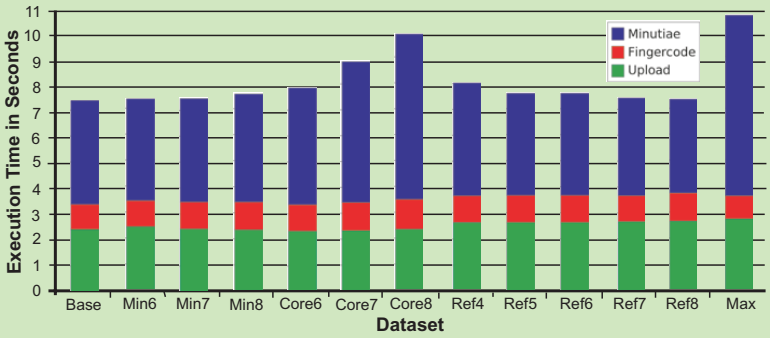

Figure 9 - **Composition of the Execution Times**

Config Name	Min. EER %	Hyb. EER %	Load time s	FC time s	Min. time s	Total time s
base	5.5	2.2	2.41	0.96	4.00	7.37
core6	4.6	1.7	2.34	1.02	4.49	7.85
core7	3.8	1.5	2.33	1.09	5.44	8.86
core8	3.7	1.4	2.35	1.15	6.38	9.88
min6	5.2	2.0	2.44	1.00	3.92	7.36
min7	5.0	2.0	2.43	1.04	3.98	7.45
min8	4.6	1.9	2.35	1.06	4.15	7.56
ref4	4.5	2.0	2.64	1.03	4.35	8.02
ref5	4.2	2.0	2.63	1.04	3.92	7.59
ref6	4.0	2.0	2.66	1.01	3.97	7.64
ref7	3.9	2.0	2.67	1.00	3.77	7.44
ref8	4.2	2.0	2.65	1.08	3.62	7.35
max	3.0	1.4	2.74	0.89	6.95	10.58

Table 2 - **Performance and Execution Time Obtained With the Configurations of Parameters in Table 1**

This entire phase could be avoided if we were able to extract a coherent reference axis from the fingerprint which, combined with the core, would make fingerprint representations totally translation and rotation invariant. In reality, the region around the core exhibits local symmetry, and some authors have successfully used this symmetry axis to improve fingerprint classification. Unfortunately it cannot be detected with the precision needed by a fingerprint matcher, due to the skin's elastic deformations produced by the friction between the finger and the sensor.

Incrementing max_{out} does not appear to affect the execution time much. The algorithm has linear complexity over the number of edges of the graph, which in turn is approximately proportional to max_{out}; on the other hand its implementation has been specifically studied to quickly skip redundant edges, so that the *effective* complexity is almost linear with the number of minutiae, independently from their outdegree.

Increasing the maximum indegree (max_{in}) we introduce more redundancy into the graph. This improves the algorithm precision by reducing the probability of missing a pair of matching mi-

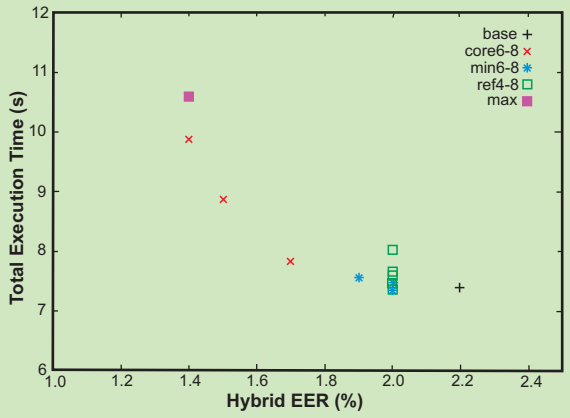

Figure 11 - **Precision Versus Speed at Various Parameter Configurations**

nutiae, and it has also the side effect of lowering the execution time because a greater fraction of the edges is skipped during the matching process.

When considering the time and EER performance of the hybrid algorithm, obtained by merging the fingercode and minutiae scores, it becomes clear (Table 2) that the most significant parameter is the number of core neighbours.

For all the other cases, the overall EER remains almost constant between 1.9% and 2.0%. The total execution time also has a maximum difference of less than 0.7 seconds. However, increasing the number of core neighbours from 5 to 8 allows reducing the EER from 2.2% to 1.4%. with a corresponding increase of the computation time of approximately 2.5 seconds (from 7.37 to 9.88).

By choosing the EER point as representative of the achievable precision with each set of parameters, figure 11 plots each EER value against the execution time needed to achieve it, thus summarizing the choices that are available in the trade off between precision and speed.

From Figure 8 it is evident that the minutiae algorithm allows, by itself, to achieve an EER of approximately 3%.

The ROC curve, evaluated for 8 neighbour points at the core, allows for an FRR of only 5% for a FAR of 1%. Figure 10 shows the performance data obtained by combining the two methods. The two top graphs show the scores for, respectively, genuine matches and impos-

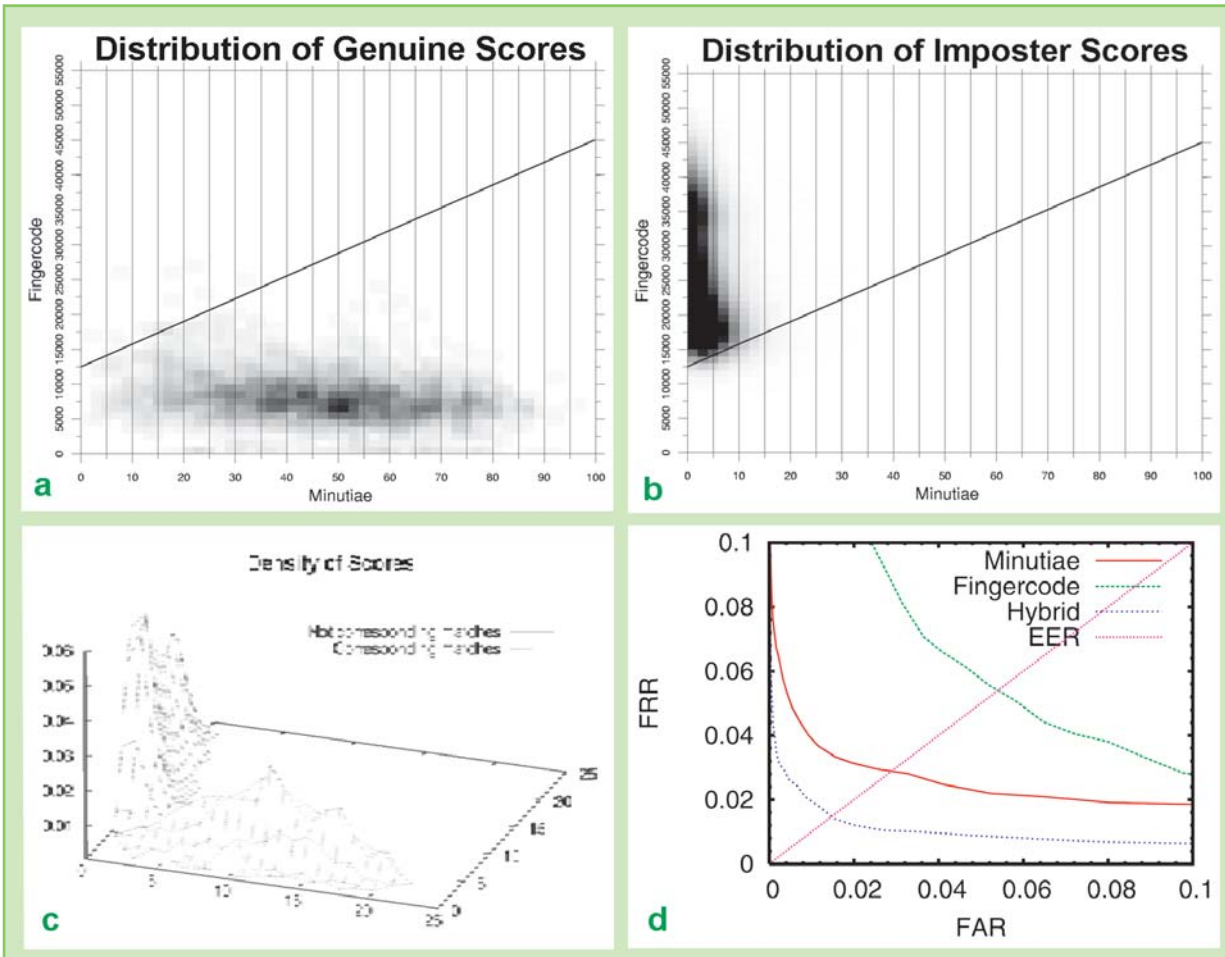

Figure 10 - **(a) (c) Joint Score Distributions for Genuine and Impostor Matches. (d) ROC Curves.**

tors, as measured by the minutiae based algorithm (on the X axis higher scores are better) and by the fingercode algorithm (on the Y axis lower scores are better). The linear combination that statistically performs best for our fingerprint database is represented by the separation line plotted in the two graphs.

It is very difficult to understand what is the density of the scores with the two dimensional diagrams on the top of the figure. This is why, in the left bottom side, we have also provided a 3D view, which hopefully allows a much clearer understanding of the density of the points in the two dimensional score spaces.

Finally, the bottom right hand graph shows the ROC curve for each matcher taken separately, and for their combination. The hybrid combination results in a considerable increase of performance, when compared with the individual algorithms. In the hybrid ROC curve, and for each possible FAR, the value of FRR is consistently lower than the ones obtainable with the fingercode and minutiae methods alone. The EER that can be obtained by the hybrid method is approximately 1.4%, sensibly lower than the 3% and the 4% values that can be obtained by the minutiae and the fingercode matchers.

Implementation Notes

The described biometrics authentication system has been developed as an extension to the MUSCLE Card open architecture [1], which defines a host side smartcard API that applications can use to access storage, cryptographic and PIN management on card services in a unified, card independent way. Translation of the API functionality into low level smart card commands is delegated to a set of plugins that support a wide range of devices. One of these plugins supports the MUSCLE Card Protocol [2], implemented on JavaCard compliant devices as an open applet.

Briefly, this protocol allows applications to manage on board data (arranged into separate containers, called objects), cryptographic keys and PIN codes. An access control model allows to protect objects and keys by associating *Access Control Lists* (ACLs) to them. Each ACL states, for each operation allowed on the key or object, which authentication mechanisms must have been successfully run in the smartcard session in order to allow the operation on the object.

Biometrics authentication has been embedded in this context by allowing the access to on card resources only after a successful on board fingerprint matching. For example, it is possible to allow the reading of an object contents, or the use of a cryptographic key (such as for digital signature applications), only after the user has successfully run a biometrics authentication mechanism, possibly in addition to a PIN based or challenge response cryptographic authentication.

Various secure applications have been modified in order to integrate smart cards through this framework, and are scheduled to be modified in order to support the new biometrics authentication mechanism. Examples are OpenSSH, an open source implementation of the Secure Shell [16] protocol for secure remote terminal, and a command line application for digital signatures. Higher level middleware components have also been developed using this framework, like a PKCS#11 [13] module, allowing integration of all available applications supporting this standard on open platforms (such as Netscape or Mozilla), and a *Pluggable Authentication Module* (PAM) [14], allowing smart card based secure login, and smart card based access to all applications using this mechanism. All software components are available for free download either from the Muscle Card web site [18], or from the SmartSign web site [19].

Conclusions and Future Work

In this paper, a hybrid fingerprint matching mechanism was introduced, designed with the aim of running onto a programmable smart card. The experimental tests show that taking advantage of the simplifications inherent to our scenario and using ad hoc designed data representations it is possible to implement high performance, multi modal matching algorithms even into such low resource devices as Java cards, with reasonable execution times.

The work can be extended in many directions. Currently, the critical factor in our procedure is the computation of the core. We are now evaluating different filtering options and different methods for computing a better estimate of the core position. Furthermore, local features of the fingerprint (or of the graph encoding) can be identified and exploited for improving the performance of the matcher. Finally, the Java code implementation of the algorithm is now under revision, in order to bring down the complexity of the program in selected points and to reduce execution times to an order of magnitude compatible with interactive usage.

References

[1] Tommaso Cucinotta, Marco di Natale, and David Corcoran, *An open middleware for smart cards*. In CRL Publishing Ltd, editor, To appear in a special issue of the Computer Systems Science and Engineering (CSSE) Journal.

[2] Tommaso Cucinotta, Marco Di Natale, and David Corcoran, *Breaking down architectural gaps in smart card middleware design*. In Proc. of the 1 st International Conference on Trust and Privacy in Digital Business (TrustBus'04), volume LNCS 3184, pages 279—288, Zaragoza, Spain, 8 2004. Springer.

[3] P. Loddo G.L. Marcialis, F. Roli, *Fusion of multiple matchers for fingerprint verification*.

[4] E.R. Henry, *Classification and uses of finger prints*. Routledge, London, 1900.

[5] L. Hong, A. Jain, S. Pankanti, and R. Bolle, *Fingerprint enhancement*. In FL Sarasota, editor, Proc. 1st IEEE WACV, pages 202—207, 1996.

[6] A. Jain and S. Pankanti, *Automated fingerprint identification and imaging*

[7] Kluwer, editor, *Biometrics, Access Control, Smart Cards: A not So Simple Combination*, IFIP Conference Proceedings, 2000.

[8] G. Loy, *Fast computation of the gabor wavelet transform*. In DICTA2002: Digital Image Computing Techniques and Applications, 1 2002.

[9] S. S. Iyengar P. B. Wamelen, Z. Li, *A fast algorithm for the point pattern matching problem*.

[10] S. Prabhakar, *Fingerprint classification and matching using a filterbank*. PhD thesis, Michigan State University, 2001.

[11] N.K. Ratha, J.H. Connell, and R.M. Bolle, *Enhancing security and privacy in biometricsbased authentication systems*. IBM Systems Journal, 40(3), 2001.

[12] Arun Ross, Salil Prabhakar, and Anil Jain, *Fingerprint matching using minutiae and texture features*. In Proceeding of the International Conference on Image Processing (ICIP), pages 282—285, 10 2001.

[13] RSA Laboratories, *PKCS 11 version 2.1.1 Final Draft: Cryptographic Token Interface Standard*, June 2001.

[14] V. Samar and R. Schemers, *Request for comments 86.0: Unified login with pluggable authentication modules (PAM)*, October 1995.

[15] Dirk Scheuermann, Scarlet SchwiderskiGrosche, and Bruno Struif, *Usability of biometrics in relation to electronic signatures*. Technical Report GMD Report 118, GMD Forschungszentrum Informationstechnik GmbH, 11 2000.

[16] T. Ylonen, T. Kivinen, M. Saarinen, and S. Lehtinen, *Internet Draft: SSH Protocol Architecture*. Network Working Group, January 2002.

[17] http://www.biometrika.it

[18] http://www.musclecard.com

[19] http://smartsign.sourceforge.net

About the Authors

Tommaso Cucinotta coordinates research activities at Scuola Superiore Sant'Anna, Pisa (Italy) in the area of computer security, with reference to protocols and architectures for smart card interoperability, digital signatures, on-card biometrics based authentication, and workflow and document management systems security. He also cooperates at research activities in the area of QoS control for soft real-time applications. He holds a course on Computer Security and Cryptography in the International Master on Information Technology at Scuola Superiore Sant'Anna.

Riccardo Brigo is a PhD student at Scuola Superiore Sant'Anna. He works on applications of smart-card technology in the context of public administration, focusing on the implementation of biometric protection system onto cryptographic, programmable devices.

Marco Di Natale is associate professor of computer engineering at the Scuola Superiore Sant' Anna in Pisa. He is currently director of the RETIS Lab.(REal TIme Systems Laboratory) of the Scuola Superiore S. Anna and national representative in the mirror group of the EU ARTEMIS Technology Platform on Embedded Systems. He started his esearch work during his PhD at the University of Massachusetts and his research interests include real time systems, algorithms, embedded systems, operating systems and design methodologies for embedded systems; object oriented and component-based design. Professor Di Natale is currently teaching the Operating Systems course in the Electronic Engineering program of the University of Pisa. He is an IEEE member and has served in the program or technical committees of some of the most important international conferences in the embedded systems and real-time systems field.