Pattern Anal Applic (2006) 9:359–376
DOI 10.1007/s10044-006-0048-4

SHORT PAPER

An asymmetric fingerprint matching algorithm for Java Card ^{TM}

Stefano Bistarelli · Francesco Santini ·
Anna Vaccarelli

Received: 6 July 2005 / Accepted: 31 July 2006 / Published online: 3 October 2006

Abstract We propose a light-weight fingerprint matching algorithm that can be executed inside the devices with a limited computational power. The algorithm is based on the minutiae local structures (the ‘‘neighborhoods’’), that are invariant with respect to global transformations like translation and rotation. The match algorithm has been implemented inside a smartcard over the Java Card^{TM} platform, meeting the individual’s need for information privacy and overall authentication procedure security. The main characteristic of the algorithm is to have an asymmetric behavior, in respect to the execution time, between correct positive and negative matches. The performances in terms of authentication reliability and speed were tested on some databases from the Fingerprint Verification Competition 2002 and 2004 editions (FVC2002 and FVC2004). Moreover, our procedure showed better reliability when compared with a related algorithm on the same database. We can achieve a false acceptance rate (FAR) of 0.1%, a false rejection rate of about 6%, and from 0.3 to 8 s to match most of the finger pairs during the FAR tests.

S. Bistarelli
Dipartimento di Scienze, Università degli Studi ‘‘G. d’Annunzio’’, Pescara, Italy
e-mail: bista@sci.unich.it; stefano.bistarelli@iit.cnr.it

S. Bistarelli · F. Santini (✉) · A. Vaccarelli
Istituto di Informatica e Telematica, CNR, Pisa, Italy
e-mail: francesco.santini@iit.cnr.it

A. Vaccarelli
e-mail: anna.vaccarelli@iit.cnr.it

Keywords Biometrics · Fingerprint matching · Smartcard · Match on Card · Java Card^{TM}

1 Introduction

The term ‘‘biometrics’’ is commonly used today to refer to the authentication of a person by analyzing his/her physical characteristics (like fingerprints) or behavioral characteristics (like voice or gait).

Since these characteristics are unique to an individual, their measurement provides a more reliable system of authentication than ID cards, keys, passwords, or other traditional systems while accessing restricted areas in office buildings and factories, or controlling the security of computer networks, electronic commerce, and banking transactions. The reason is that all these secret keys can be easily stolen or cloned to steal the personal identity, or they can also be forgotten by the owner preventing the whole identification process. Biometric characteristics are, instead, generally more difficult to duplicate and they naturally always ‘‘follow’’ the owner. Moreover, an advantage of biometrics is that they cannot be lent (like a physical key), and thus, they grant the on-site presence.

The most common biometric techniques are signature verification, retinal analysis, facial analysis, fingerprint verification, hand geometry, and voice verification. These technologies are comparable by the aid of several indicators, such as permanence (measurement should be invariant with time), uniqueness (different values for different persons), universality (everyone should have this trait), acceptability (if people are willing to accept this technology), performance (the recognition accuracy and system requirements) and circumvention (how it is easy to fool the system).

Fingerprint matching is one of the most diffused biometric techniques used in automatic personal identification, because of its strong reliability and its low implementation cost; moreover, it is also the most mature and explored technology of all the others.

Performing a biometric verification inside a smartcard is notoriously difficult, since the processing capabilities of standard smartcard processors are limited for such a complex task. With *Match-on-Card* (MoC) technology, the fingerprint template is stored inside the card, unavailable to the external applications and the outside world. In addition, the matching decision is securely authenticated by the smartcard itself, in this way, the card has only to trust in itself for eventually unblocking stored sensitive information, such as digital certificates or private keys for digital signature. Our verification MoC algorithm was developed to work in this very strictly bounded environment.

The algorithm is based on some minutiae characteristics (ridge pattern micro-characteristics) and more precisely on their local structure information, so there is no need to pre-align the processing fingerprint templates, which would be a difficult task to implement inside a smartcard. Moreover, it shows an asymmetric execution time between correct positive matches (same fingerprint) and correct negative matches (two different fingers), and this is because the match procedure stops immediately when few minutiae pairs result in a positive match. If this check does not succeed, for example if the two fingers are different, or if the two acquisitions of the same finger are very disturbed, the procedure is fully executed (lasting longer) and the match decision is taken only at its end.

Experimental results show that the performances (speed and security) of the proposed algorithm are really good and allow the algorithm to be used in real verification applications.

1.1 Paper structure

This paper extends with new tests and outcomes the preliminary results presented in [1] and is organized as follows: in Sect. 2 we present some general background information about fingerprints, as their global appearance and the most important characteristics of the minutiae points (Sect. 2.1); from Sect. 2.2 to 2.4 we respectively explore the background about common fingerprint matching problems and the literature classification, the Java Card^{TM} platform characteristics used to develop the algorithm and, finally, some comments about MoC security and privacy implications and benefits. Moreover, in Sect. 2.5 we summarize some related works specific to fingerprint matching on resources-constrained devices.

Section 3 features, instead, an overview of our smartcard verification algorithm: in 3.1 we report the main characteristics of our scheme, in 3.2 we describe the information chosen to represent and match two fingerprints, in 3.3 we describe the matching procedure, and in 3.4 we show some problems and solutions concerning the Java Card^{TM} implementation.

Section 4 presents the fingerprint image databases used during the tests (Sect. 4.1), the typical match performance indicators (Sect. 4.2) and the performance evaluation of our algorithm in terms of speed and reliability (Sect. 4.3).

Section 5 draws the final conclusions and outlines intentions for future works.

2 Background

2.1 Fingerprint appearance and minutiae features

The most evident structural characteristic of a fingerprint is the pattern of interleaved ridges and valleys that often run in parallel. Ridges vary in width from 100 to 300 μm and the period of a ridge/valley cycle is typically about 500 μm. If analyzed at global level, almost all of the patterns exhibit one or more regions characterized by a distinctive shape and called *singular regions*. These regions can be classified into three typologies according to their shape: loop, delta, and whorl are characterized respectively by a $\frown$, Δ, and O shape. A particular presence of these singular regions defines the whole fingerprint class: the five classes in Henry's scheme [2] are *arch*, *tented arch*, *right loop*, *left loop*, and *whorl*.

At local level, other important features called *minutiae* refer to ridge discontinuities. Minutiae are sometimes called ''Galton details'', in honor of the first person who categorized them and observed that they remain unchanged over the individual's entire life [3]. Most frequently, the minutiae types can be identified by terminations, where a ridge line ends, and bifurcations, where a ridge bifurcates forming a ''Y'' (see Fig. 1), even if several types have been observed, described by their shape (dot, island, hook, lake, ridge crossing and multiple bifurcations).

Another important point in the image, which can be used also to align the fingerprint images, is the ''core point'', corresponding to the center of the north most

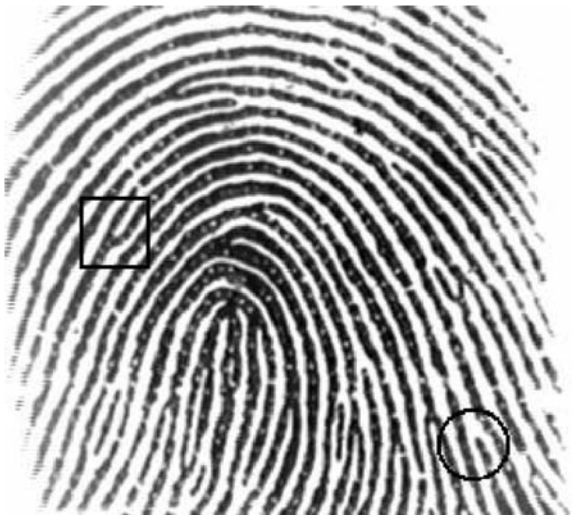


Fig. 1 Minutiae details. The *circle* highlights a ridge termination, while the *square* shows a bifurcation

loop type singular region. In fingers without loop or whorl regions, the core is associated with the point of maximum ridge line curvature.

The most important minutiae characteristics are the location coordinates inside the image, their form type (e.g. termination, bifurcation, island, etc.) and the orientation of the ridge (in degree) on which the minutia is found.

While singular regions can aid in classification, the minutiae can be used for fingerprint matching, since they represent some of the unique details of the ridge flow and are considered as singularity evidence; minutiae matching is certainly the most well-known and adopted method for fingerprint automated matching.

The two fundamental premises on which fingerprint identification is based are that the fingerprint details are permanent and unique. The validity of the first premise is established based on the anatomy and morphogenesis of friction ridge skin (except due to accidents such as bruises and cuts). The notion of the fingerprint individuality is widely accepted, based on manual inspection of millions of fingerprints, and there are several mathematical models describing the probability of a same particular fingerprint configuration [4].

2.2 Fingerprint matching problems and solutions

The response of a fingerprint match procedure is typically a matching score s that quantifies the similarity (or dissimilarity) between two fingerprint templates; the template, in its generic definition, is a mathematical representation of the fingerprint "uniqueness" to be used during the matching phase. The match decision is then regulated by applying a threshold t on this score: for example, if $s < t$ the fingerprints are considered as matching. Threshold t can be also defined as the "operating point".

Matching two templates is extremely difficult because of the variability in the different impressions of the same fingers (Fig. 2). The most important disturbing factors introduced during the image acquisition or the fingerprint feature extraction are:

- *Displacement* depending on the different positioning of the finger on the acquisition sensor.
- *Rotation* depending on the different rotation in positioning the finger on the sensor between different acquisitions.

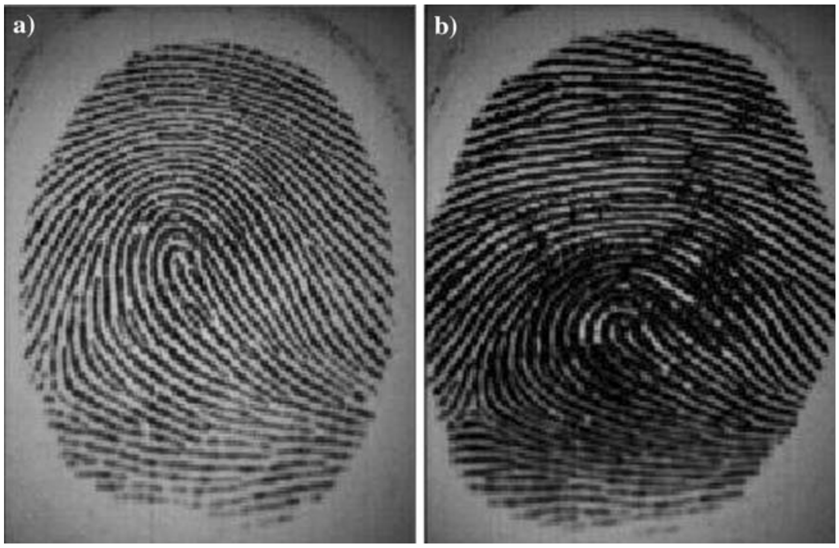


Fig. 2 Two impressions of the same fingerprint, **a** good quality, **b** distorted

- *Partial overlap*. Still because of imperfect positioning, a part of the fingerprint can fall outside of the acquisition area and, therefore, different samples of the same finger could correspond only on a small area.
- *Non-linear distortions* due to the skin plasticity. Forces non-orthogonal to the sensor surface can cause the finger image to be distorted in some zones.
- *Pressure* on the sensor surface can result in a different thinning of the ridge pattern.
- *External factors*. Other external factors, not depending on the finger positioning on the sensor surface, could be skin conditions such as dryness, sweat, dirt, grease, or possible skin disease and consumption in manual workers or the elderly.
- *Feature extractor errors*. During the extraction stage, the processing module can introduce some errors by adding some "false" minutiae (not really present in the fingerprint) or by not detecting true existing features.

Regarding the algorithms used to solve the fingerprint matching problem, all the possible approaches can be classified into three main branches [2]:

Correlation-based matching In this class, the match is obtained by superimposing two fingerprint images and then computing the correlation between the corresponding pixels [5, 6]; in this case the template is directly the finger image. Some problems could derive from non-linear distortions that make the impressions of the same finger significantly different in terms of global structures. Also the skin condition and pressure cause the image brightness and contrast to vary across different impressions.

Minutiae-based matching The theory behind this algorithm class is fundamentally the same as for the manual fingerprint examiners. Matching essentially consists in finding the maximum number of corresponding minutiae between the two templates.

Ridge feature-based matching The approaches belonging to this family compare the fingerprints in terms of ridge pattern features other than the minutiae or pixel intensity (the previous two classes): examples are the shape features [7], spatial relationship and geometrical attributes of the ridge lines [8] (like frequency and shape), texture information [9] and sweat pores [10]. In principle, the first two methods could be seen as subsets of this one, since also the minutiae and pixel intensity can be considered as ridge features.

The minutiae matching problem can be addressed also as a more general *point pattern matching* problem, even if the presence of the ridge direction associated with each minutia, or the minutia type, add some more important information. The *Hough transform-based* approach is quite popular for resolving the fingerprint match: the transformation parameters space, which can relate two sets of points, is firstly discretized, then the correct parameters can be derived by accumulating evidences in this space, and, finally the most "voted" transformation will be chosen [11]. The use of an algebraic technique for point pattern matching is, instead, described in [12].

Focusing only on the minutiae based algorithms, we can subdivide them into two more classes. *Global minutiae matching* requires a first fingerprint alignment phase that subsequently allows template matching. The alignment can be absolute if each fingerprint is pre-aligned independently from the others, and relative if the input fingerprint is pre-aligned with respect to the database template with which will be compared. Relative pre-alignment is generally more effective because the features in the stored template can properly help the process. An interesting approach based on minutiae ridge registering, which exploits ridge features for a relative pre-alignment, is proposed in [13]: the minutiae representations are converted into a symbolic string and then the two strings (one for each fingerprint) are matched with a dynamic programming technique finding their *edit distance* and, finally, this distance is compared with a threshold. A variant of this method is proposed in [14].

In *local minutiae matching*, two fingerprints are compared according to their local minutiae structures, which are characterized by attributes invariant with respect to global transformations such as translation or rotation; it is consequently appropriate for matching without any a priori alignment. Local matching supplies simplicity, low computational complexity and higher distortion tolerance, while global matching grants a higher distinctiveness. This algorithm class is mainly based on the neighborhood concept, for which the local structure is represented by the features of the nearest minutiae: some examples are in [15–17]. Other solutions are based on grouping the minutiae inside a bounding box [18, 19], or by the triangular matching proposed in [20]. Local structures can be used also for fingerprint indexing [21].

Modern high security applications typically adopt multimodal biometric systems. These systems use distinct types of features concerning the same biometric technique (i.e. fingerprint minutiae and texture), or even other completely different biometric characteristics combining, for example, fingerprint and facial analysis. The match result is derived from distinct submatcher modules working on the different biometric measurements, like the algorithm in [22] which uses minutiae plus texture information: all the results are then merged into one single final decision (fusion of the matchers), improving the global reliability of the verification system.

2.3 The Java Card^{TM} Platform

A smartcard [23] resembles a classic credit card in size and shape, but inside it may contain an embedded 8–16 bit microprocessor (microprocessor card) under a gold contact pad on one side of the card. This grants computational power and enforces access to the on-card data. Smartcards with memory only (memory card) offer simply a protection for data storing. Common smartcards may have up to 4–5 kb of RAM, about 160 kb of ROM, 32–72 kb of programmable EEPROM and an internal clock up to 30 MHz. The card operating system and other permanent data are usually masked in the ROM memory, while the EEPROM can be used to store the applications or personal information. The smartcard receives its power from external sources like a card reader, more generally defined as "Card Acceptance Device" (CAD).

Java Card^{TM} technology [24] adapts the Java^{TM} platform for use on smartcards or devices like USB tokens, whose environments are highly specialized, and whose memory and processing constraints are typically more severe than those of a common PC. This adaptation produces a global reduction of the platform functionalities and results in a substantial decrease of the programming expressive capacity. Java Card^{TM} technology is described in Fig. 3, which shows the different architectural layers on the smartcard.

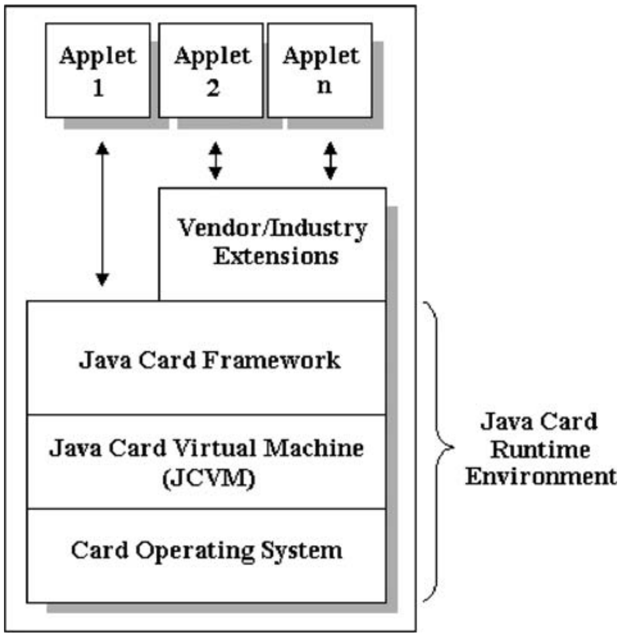

Applet 1
Applet 2
Applet n
Vendor/Industry Extensions
Java Card Framework
Java Card Virtual Machine (JCVM)
Card Operating System
Java Card Runtime Environment

Fig. 3 Java Card^{TM} architecture

The Java Card^{TM} virtual machine (JCVM) specification defines a subset of the Java programming language and additional constraints on many program attributes, like the maximum number of classes in a single package [24]. The Java Card^{TM} API specification defines a small subset of the traditional Java programming language API, while the Java Card^{TM} Framework defines its own set of core classes specifically created to support the applications.

The applet objects are created in the EEPROM persistent memory and so their state is saved between different CAD sessions, but it is a 1,000 times slower to write in the EEPROM than in the RAM, due to their different technology. Since a garbage collector is not always available in a Java Card^{TM} implementation (it is not mandatory in the specification), an application may never reclaim the storage allocated to the objects that are no longer referenced and, consequently, it is essential to reuse the same objects during several CAD sessions, otherwise the memory space will sooner or later be saturated. Inside a smartcard, creating and deleting the objects are time-consuming tasks.

One more important drawback is also the additional bytecode interpretation layer that could slow down the computation, with respect to compiled applications.

A more exhaustive description of the Java Card^{TM} platform is given in [25].

2.4 MoC introduction

For a reliable biometric verification, it is important to maintain the confidentiality of the fingerprint data and it is essential to perform the match in the same environment where the data is stored. One possible solution is to store the private biometric template on a smartcard and to perform the verification algorithm directly on it. This implements the description of the MoC technology, in which security is significantly improved in respect to the *Template-on-Card* (ToC), where only the template is saved on the card and has to be extracted every time during the verification process executed outside, for example in a PC.

In the MoC architecture (see Fig. 4), the "candidate template" (acquired for the verification phase) has always to be sent to the smartcard where the "reference template", associated with the card-user identity, has been already stored during the card personalization. In this way, it is impossible to intercept the reference

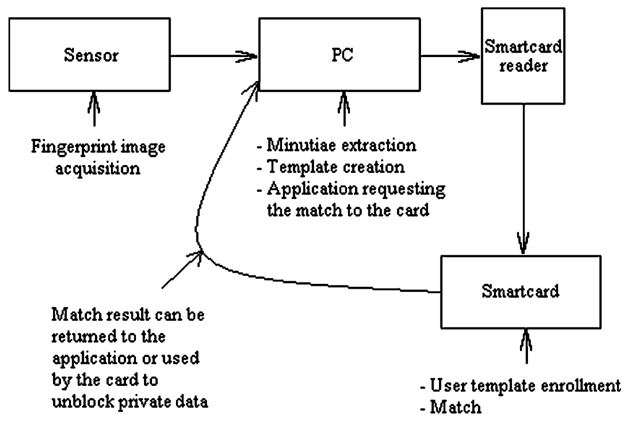

Sensor
PC
Smartcard reader
Fingerprint image acquisition
- Minutiae extraction
- Template creation
- Application requesting the match to the card
Smartcard
Match result can be returned to the application or used by the card to unblock private data
- User template enrollment
- Match

Fig. 4 Typical distribution of the tasks in a Match-on-Card (*MoC*) system

template with the aim to eventually propose a ‘‘replay’’ attack, but, for the same purpose, it is still possible to intercept a candidate template that leads to positive match (cryptography could help, in this case). Moreover, the embedded smartcard environment is clearly more secure with respect to a common PC, and it is impossible to install additional software that could influence the match procedure. Finally, the private biometric data is not stored in central databases, whose security violation could be a serious menace: a successful attack could steal a lot of critical information. Considering the attacks described in [26], MoC technology can avoid the attacks based on substituting or modifying the matcher, tampering with the reference template and by controlling the channel between the reference template and the matcher. In addition, if the MoC verification decision is internally used by the card to un block private data and it is not sent outside, the final ‘‘yes/no’’ result of the matcher cannot be intercepted and replaced.

Attacks against smartcard executed to analyze the content without the authorization, like *differential power analysis attacks* (DPA) [27], specifically need the target smartcard, and, for this reason, are not large-scale attacks and cannot be easily repeated on a large collection of templates. With MoC, the personal biometric information (the template) never leaves the smartcard and is owned directly by the user, granting in this way more privacy and increasing the user acceptability of fingerprint verification.

In the *System-on-Card* (SoC) the card is further enhanced with the fingerprint scanner and the feature extraction function, thus achieving the best security: in this way, it is possible to prevent the attacks on the feature extraction software and on the scanner-extractor and extractor-matcher channels. This technology is still far from being easily implemented with low cost cards, for example because processors and connections need to withstand bending tests, and because the increased power consumption must be compatible with power supply.

While the ToC technology is today fully implementable, since the card memory is wider than a typical template space occupation (a few kilobytes), implementing a matching algorithm inside the card (MoC) is yet a difficult task due to the still limited smartcard computational power, and since this type of computation usually demands some relatively complicated arithmetic. Moreover, the verifying procedure has to be kept as fast as possible even in this environment, to avoid to frustrating the card-owner with a long wait. Thus, keeping a high security while reducing the match complexity and time, is the most important problem concerning MoC.

2.5 Related work

Regarding scientific literature on MoC, in [28] the minutiae spatial coordinates and the associated ridge angle (x, y, θ) are used to represent and match the minutiae in the two templates. The authors employ an accumulator array to compute the approximate transformation, and this process is repeated to find a fine-grain resolution, discretizing every time the array cell corresponding to the best transformation. However, the resources needed by this algorithm are above the current availability of today’s smartcard, since it has been tested on a 32 bit ARM-7 processor. The test image set is composed of 400 images taken from 100 individuals and the reported *equal-error rate* (EER) is about 6% (see Sect. 4.2 for performance indicators definition).

Even the authors of [29] adopt the same representation (x, y and θ). The preliminary registration between the two templates is accomplished outside the smartcard using the average horizontal and vertical coordinate values and the average direction of all the minutiae. The match is then performed by transforming the coordinates in a polar form with respect to the previous average values; for this reason, problems could arise in case of partial overlapping between the two fingerprints (see Sect. 2.2). Here the database have been generated using only 10 different fingerprints, and about 20 minutiae in each template. The total time for an authentication is 3–4 s, considering also the data transfer.

One more paper using a first step concerning external registration is [30]. Here, the match is then accomplished by applying Gabor filters to the fingerprint image, as described in [31]. The reliability tests were executed on the Siemens fingerprint database, containing 100 images each of 36 distinct users. Performances are reported for several different system configurations and, in the best reliability case, the procedure achieves a *false rejection rate* (FRR) of about 4% for a FAR of 0.1%, using four different reference templates (storage occupation is 9–10 kb) and only one "query template". The match time is not reported, and it is linearly dependent on the number of templates (as the storage cost).

The algorithm in [32] use both the binary fingerprint image and the minutiae position. First, the host PC transmits the core location of the new acquired image, which is used by the card to compute registration parameters; these parameters are sent back and used by the host to register its image. Then the smartcard chooses some coordinates (typically, near to the minutiae locations) and the host PC use them to cut rectangular pieces (called "chips") from the fingerprint image. Again, these chips are transmitted to the card and used for the final match step. Performances are obtained on a database with 576 fingers, with a FAR of 0.1% and FRR of 2%.

In [33] is presented a verification method implemented in a personal digital assistant (PDA) with a 206 MHz StrongArm processor, so for a less resources-constrained system in respect to smartcards. The minutiae information recorded during the extraction is the triplet x, y and θ. The main aim of the authors is to show that replacing floating-point with fixed-point computation does not affect the verification reliability performance: actually, most of the embedded processors in embedded devices do not support floating-point arithmetic. Only a small subset of all the minutiae points (the ones near to the core point) is used during the match procedure, which simply applies a bounding box technique. The test databases is collected from 383 different fingers for a total of 1,149 images and the average computation time is 0.9 s. The achieved EER is nearly 7%.

Even the authors in [34] introduce an algorithm developed for embedded devices. This algorithm is based on the minutia neighbor features, like the neighbor distance and orientation with respect to the central minutia. The minutiae neighborhood similarity is computed by finding the feature distances and successively controlling them with the aid of a delimiting bounding box; if the checks are positive, the corresponding neighbors are then matched. The compared minutiae are considered as matched if the total number of their matching neighbors is above a certain predefined value. The final decision regarding the two entire templates is taken from an estimation based on the total number of the minutiae matched in this way. The test database has 100 images and performance are claimed to be 0.01% FAR and 1% FRR.

In [35] a description is given of a matching algorithm expressly developed for the Java Card$^{\text{TM}}$ platform and using the same feature extractor software as the one adopted in our solution. It uses two distinct algorithms for different feature types (a hybrid matcher), and, at the end, the overall score is computed as a linear combination of the two independent sub-scores. The first algorithm is based on the minutiae features and a graph structure is built starting from the core point position, then visiting the neighbor minutiae. The matching procedure is inspired from the point-pattern matching algorithm in [36] and its purpose is to find a *spanning ordered tree* touching as many nodes as possible in the two graphs. The second algorithm is ridge feature-based (texture) and is implemented exactly as described in [9]. The on-card computation needs about 11–12 s, and security is reported and compared with our results in Sect. 4.3.

3 Our MoC algorithm

3.1 Solution characteristics

As introduced in Sect. 2.4, matching two fingerprints is a difficult problem due to several elastic and rigid transformations and other disturbing factors. Trying to develop a solution in a Java Card environment$^{\text{TM}}$ adds some more problems depending on the platform and, principally, on the smartcard computational constraints (Sect. 2.3). However, bringing the verification service on this kind of very small devices may resolve many security and privacy problems (Sect. 2.4).

In Sect. 2.5 we present the background directly related to the MoC problem. Some works have however been proposed for processor and devices with higher computational capabilities [28, 33, 34], other algorithms are analyzed on databases with few images [29] and in other papers, good security performances are achieved only repeating the match on multiple templates [30], thus decreasing the average match time. Most of times, given solutions seem to need too much processor resources to be developed inside present-day smartcards.

In FVC2004 competition [37], the EER average performance drop between the open and the light (lightweight algorithms) categories, and considering the top ten systems, is about 44% [38]; thus, the computational limitations heavily impact the security performance. Besides, smartcard requirements are more stringent than FVC2004 Light category: the 0.3 s comparison time limit in the competition refers to a 3,000 MIPS CPU, while a typical smartcard CPU performs at about 10 MIPS [38].

During our work, the intention was to propose a scheme to be really implemented in a current smartcard environment; at the same time, the algorithm had to have satisfying characteristics of speed and security. For this reason, in our solution we use the local minutiae matching technique based on minutiae neighborhood (see Sect. 2.4), with which we avoid to register the two fingerprints outside the card, and which generally supplies simplicity and low computational complexity. With local structures we are able to deal with displacement, rotation and partial overlapping problems, while we employ bounding (or tolerance) boxes on the feature differences to compensate for the small plastic deformations [2, 16]. Moreover, we adopt a cascade approach to accumulate similarity evidences between the two templates, trying to stop the match procedure as soon as it is possible to state a positive result.

Some parameters of our solution (e.g. minutiae and neighbors' numbers) can be easily adapted to the increasing performance chips of the next future (surely for the next 2–3 years), while we reckon that more complicated algorithms need more advanced technological improvements. In terms of hardware, our platform could also correspond to one of the lowest-cost MoC solution of the next years.

3.2 Features extraction

In our algorithm implementation, the image processing stage used to extract the minutiae set from a fingerprint is based on the NIST Fingerprint Image Software [39], an open source toolkit which includes the MINDTCT package (''minutiae detection''). This detection step extracts information about the minutiae, such as the x and y coordinates, local ridge direction θ, type (found between the ridge ending or the bifurcation classes) and the reliability, as directly derived from the corresponding image zone quality. Moreover, for every such singularity point, MINDTCT finds a list of the neighboring minutiae, which are the nearest minutiae to the considered one in the euclidean space (from now, simply called ''neighbors''). Also an estimation of the *ridge count* between central and neighbor minutia is found. Given two points a and b, the ridge count between them is the number of ridges intersected by the segment$\overline{ab}$: forensic experts and latent fingerprint examiners often use the ridge count to increase the reliability of their analysis [2].

We use this information extracted by MINDTCT to derive the ''features'' to build the templates directly used by our matching algorithm. These features are calculated for each neighbor with respect to its central minutia, and our template is made of only these neighbor features: every minutia in our template is described using only the features of all its neighbors (its neighborhood).

Each neighbor is described by the four features graphically described in Fig. 5, where D is the segment linking the central minutia A and its neighbor B, θ_1 and θ_2 are the two minutiae orientation angles and α is the angle between D and the orientation of A (θ_1):

- The euclidean distance between the central minutia and its neighbor (segment D in Fig. 5); referred to as *Ed* in the rest of the paper (*euclidean distance*).
- The angle between segment D and the central minutia ridge direction (angle α in Fig. 5); latterly referred to as *Dra* (i.e. *distance relative angle*).
- The difference angle between central minutia and neighbor ridge orientation angle ($\theta_1 - \theta_2$ in Fig. 5); latterly referred to as *Oda* (i.e. *orientation difference angle*).
- The ridge count between the central minutia and its neighbor: in Fig. 5, the ridge count value is 1; latterly it will be referred to as *Rc* (*ridge count*).

Choosing the maximum number of neighbors is very important for system reliability performances (but in contrast with the matching speed), and so we have decided to increase this number from the default MINDTCT value (5) to the new value of 8. Thus, to represent a minutia in the template we use a maximum of 32 feature values, 4 for each of its neighbors.

We have also modified the MINDTCT C source code to find, for every minutia, only the neighbors

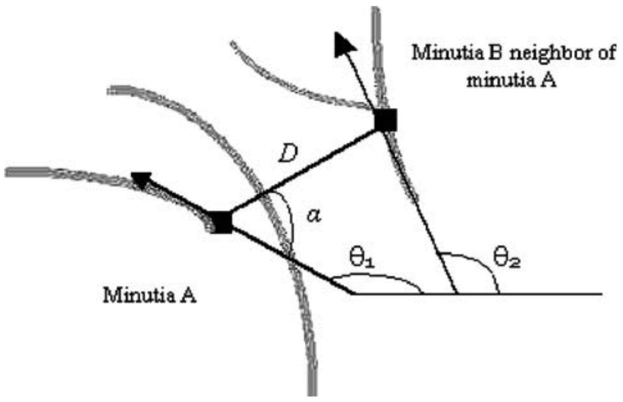


Fig. 5 Graphical description of the features

above a minimum reliability threshold (as previously stated, the reliability evaluation is found by MINDTCT). If the number of neighbors found in this way is too low (i.e. less than 5), then the neighbors are searched again with a lower reliability threshold.

We decided to discard the minutiae type information (bifurcation or ridge ending) since sometimes MINDTCT fails to correctly identify the type, principally in the disturbed ridge zones or in the loop zones.

We have introduced all these changes to build a "good" neighborhood, with enough trustworthy information to face the possible lack of some minutiae in the template. Given a minutia, the collection of all the features of its neighbors describes its neighborhood view, which is a sort of "panorama" of the surrounding minutiae. Inside the template, these neighborhoods are stored in the reliability decreasing order of their corresponding central minutia (the reason is in Sect. 3.3).

3.3 Algorithm description

Our proposed matching algorithm computes how much the neighborhood of a minutia in the candidate template is similar to the neighborhood of each minutia in the reference template. At the end of this scan step, the two most similar minutiae (those whose "similarity value" is the lowest) are matched and then discarded from subsequent scan phases concerning other different minutiae of the candidate template. All these similarity measures are summed together during the process and, at the end, the algorithm can decide if the two templates match by applying a threshold on this global score.

As previously stated, matching on smartcard environment is bounded by the hardware simplicity (CPU limitations first of all), and thus waiting for a complete minutiae match could lead to a waiting time which is too long for the user. In our algorithm we solve this problem by stopping the computation as soon as it is possible to assert, with satisfactory confidence, that the considered templates belong to the same fingerprint. To achieve this improvement, our algorithm stops as soon as it finds some minutiae pairs (i.e. a number between 2 and 5) matching with a very good average similarity value, or even immediately when only the last examined minutiae pair has a matching value lower than a very rigourous threshold. Otherwise, if these two conditions are not true, the algorithm explores all the minutiae pairings space. This relaxation showed a very good security performance in our tests and provided an evident speed improvement in the matching decisions regarding positive matches (Sect. 4.3). The delay for unsuccessful matches scanning all the minutiae list of different fingerprints is not of much interest, since it is clearly more important to gain a high execution speed while verifying the true card-owner identity, than quickly rejecting an impostor!

As input, one iteration of the match procedure receives the candidate minutia (*C* in Fig. 6) to be matched, and the entire reference template with all its minutiae (each of them is called *R* in Fig. 6). As shown in Sect. 3.2, the minutia information exactly corresponds to its neighborhood features: the terms "minutia" and "neighborhood" can be used as synonyms, since to match a minutia we need to match its neighborhood. The procedure in Fig. 6 is repeated for

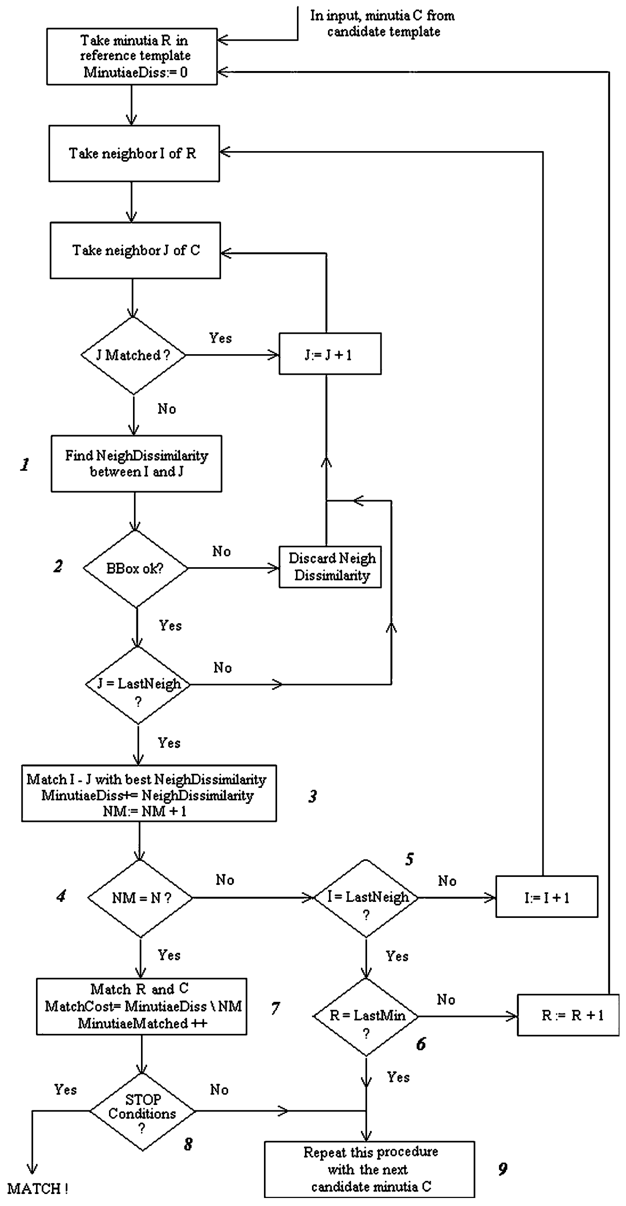

In input, minutia C from candidate template
Take minutia R in reference template MinutiaeDiss:= 0
Take neighbor I of R
Take neighbor J of C
J Matched ?
Yes
J:= J + 1
No
1
Find NeighDissimilarity between I and J
2
BBox ok?
No
Discard Neigh Dissimilarity
Yes
J = LastNeigh ?
No
Yes
Match I - J with best NeighDissimilarity MinutiaeDiss+= NeighDissimilarity NM:= NM + 1
3
4
NM = N ?
No
5
I = LastNeigh ?
No
I:= I + 1
Yes
Yes
Match R and C MatchCost= MinutiaeDiss \ NM MinutiaeMatched ++
7
R = LastMin ?
No
R := R + 1
6
Yes
Yes
STOP Conditions ?
No
8
MATCH !
Repeat this procedure with the next candidate minutia C
9

Fig. 6 Flowchart of the match algorithm

each *C* minutia in the candidate template, or until one stopping condition is true.

The algorithm scans sequentially the minutiae of the reference template until a good match for the input minutia is found. Both candidate and reference minutiae lists are stored according to the increasing minutia reliability value: in this way we try to stop the procedure more quickly by scanning a reduced portion of the template minutiae lists. In fact, a minutia with a high reliability in a given template, when not cut away by partial overlapping, will probably have a high reliability also in other templates obtained from the same finger. Thus, the stopping conditions can be met earlier than in a casual disposition of the minutiae in the list. Moreover, it is obviously better to prematurely stop the procedure with few but "good" minutiae than with low quality ones. The minutia of the reference template matched in this way, is then marked as "already matched" and is not considered in the successive iterations.

To compute the dissimilarity between two minutiae (or neighborhoods) in different templates, the algorithm uses the neighbor features explained in Sect. 3.2 and executes the following four steps in sequence (labels 1 and 2 in Fig. 6):

1. To find the difference in absolute value between corresponding features: $EdDiff = |Ed_1 - Ed_2|$, $rcDiff = |Rc_1 - Rc_2|$, $draDiff = |Dra_1 - Dra_2|$ and $odDiff = |Oda_1 - Oda_2|$.
2. To check that every feature difference value is below the corresponding acceptance threshold; if only one difference value exceeds the relative threshold, the two neighbors cannot correspond in the two respective neighborhoods (*edDiff* must not be greater than the limit set by *edDiffThr*, *rcDiff* than *rcThr*, *edDiff* than *draThr* and *odDiff* than *odThr*). The set of the four feature difference thresholds can be globally defined as the features *bounding box*, which makes the algorithm tolerant to small non-linear distortions.
3. To multiply each feature difference for the corresponding weight value: thus, $edWghtDiff = edDiff \times edWght$, $rcWghDiff = rcDiff \times rcWght$, $odWghtDiff = odDiff \times odWght$ and $draWghtDiff = draDiff \times draWght$. The different weight values are necessary to attribute more importance to the features that match better such as, in our test experience, the euclidean distance. Before multiplying for the weight value, we have normalized the feature differences with respect to the bounding box thresholds (to have homogenous values).
4. To sum together all the four weighted differences to represent the global dissimilarity between the two neighbors: $NeighDissimilarity = edWghtDiff + rcWghtDiff + draWghtDiff + odWghtDiff.$

Following these four steps, the algorithm finds for the first neighbor of the reference minutia, the most similar neighbor in the input minutia among those satisfying the bounding box checks; the most similar is the one for which the algorithm finds the lowest *NeighDissimilarity* value (label 3 in Fig. 6). The chosen most similar neighbor in the reference minutia is then marked and not considered while matching other neighbors. The obtained *NeighDissimilarity* value is then added to the global similarity score between the minutiae, *MinDissimilarity*, and the number of neighbors matched *NM* is incremented by one (label 3 in Fig. 6).

This procedure is repeated exactly for all the other neighbors in the minutia of the reference template ("No" case in the label 5, Fig. 6) and excluding the already marked ones, or until the required minimum number *N* (i.e. 4) of neighbors has been matched ("Yes" case in label 4, Fig. 6): we have seen that stopping before the whole neighborhood scan is sufficient to grant good reliability and, meanwhile, the match time is considerably speeded up.

Otherwise, at the end of the two neighborhoods scanning and if the procedure has found less than *N* matching neighbor pairs between the two minutiae ("Yes" case in label 5, Fig. 6), these two minutiae can not be considered as matching because their neighborhoods agree on too few points of evidence to be a reliable pair, even if their *MinDissimilarity* value is very low. Thus, the following minutia *R* in reference template has to be checked ("No" case in label 6, Fig. 6), but if there are no more minutiae *R* to be examined, the entire procedure in Fig. 6 is repeated for the next minutia *C* in the decreasing reliability order of the candidate template.

Instead, if the neighborhoods of the two *R* and *C* minutiae have been matched ("Yes" case in label 4, Fig. 6), the *MinDissimilarity* score between *M* and *N* is finally divided by the number of matched neighbor pairs and then added to the global dissimilarity value between the candidate and reference templates: the *MatchCost*. The number of matched minutiae *MinutiaeMatched* is then incremented (label 7 in Fig. 6).

Just after we state that two minutiae/neighborhoods can be matched, we can check also if the two stopping conditions explained before can be applied (label 8 in Fig. 6): first, the algorithm checks if the temporary average *MatchCost* is "very good", that is below a tightening threshold *OptValue*; second, the algorithm checks if the last matched minutiae pair has an "exceptionally good" *MinDissimilarity* value, below a threshold which is intended to be much stricter than the previous one.

If even one of these two conditions is true, the match can be immediately stopped with a positive result (the two templates belong to the same finger), otherwise one more minutia *C* from the candidate template must be checked (label 9 in Fig. 6).

In case these conditions are never met during the templates scanning, and when all of the input minutiae have been processed, the final *MatchCost* value is divided by the number of matched minutiae *MinutiaeMatched*, finding in this way the average cost to match one single minutia. A final comparison between a match threshold and this mean value can consequently be used to decide if the two templates belong to the same fingerprint.

The described algorithm complexity is $O(n^2)$, where n is the number of the minutiae in a single template, even if, in practice, the approach of stopping the computation with few minutiae shows a significant speed improvement.

3.4 Algorithm implementation

The fingerprint matching algorithm described in Sect. 3.3 was fully developed, debugged and tested on a PC with a Pentium4 CPU (2.66 GHz), a RAM of 512 Mb and using the JDK 1.4.3. Then the same algorithm was ported on the Java Card^{TM} platform using the Java Card^{TM} 2.1.2 API, and was finally deployed on a JCOP41v2.2 Java Card^{TM} with the IBM JCOP Tools 3.1.1. The chosen smartcard has 72 kb of EEPROM, more than 4 kb of RAM memory, an internal clock up to 30 MHz (external clock 1–10 MHz) and supports cryptographic operations.

The algorithm was developed by implementing the methods of the Java Card^{TM} Biometric API [40], designed by Java Card forum (JCF) [41]. This application programming interface (approved by the *Java Card Forum Biometric Task* and the *NIST Biometric Consortium Working Group*) ensures the interoperability of many biometric technologies with Java Card^{TM} and allows multiple independent applications on a card to access the identity verification service. Clearly, our application manages even the enrollment and match requests coming from the external PC applications through several CAD sessions. The feature extraction stage is executed on the PC.

We chose to adopt the Java Card^{TM} technology since this platform offers good security and a high-level programming language, close to classic Java^{TM}. Benefits and drawbacks of this platform are those explained in Sect. 2.3. The difficulty consists in adapting the code to this restricted environment. Furthermore, the card processor supports only fixed point arithmetic and the Java^{TM} math API are totally absent, so the operations have to be very simple.

Despite this global simplification, the verification reliability performance has to be kept as high as possible. Due to these environment constraints, we limited the maximum number of the minutiae (forming the card-owner reference template) to be stored in the EEPROM: only the 20 most reliable minutiae are stored inside the card. Similarly, the match decision is taken anyway after the 20th minutia in candidate template is received (a number between 20 and 25 minutiae has shown the best performances). In the interests of speed improvement and memory occupation contraction, the maximum number of neighbors per minutia was limited to 8. Moreover, the *N* threshold in Sect. 3.3 was set to 4 to speed-up the procedure (we decide if two neighborhoods can be matched just after *N* neighbor pairs have been matched).

The neighbor feature values in Sect. 3.2 must be sampled to be stored in the low capacity Java Card^{TM} primitive data types like the *byte*, preferring its use in respect to other types such as the *short* (which has a size of 16 bits instead of 8). Features *euclideanDistance*, *distanceRelativeAngle* and *orientationDifferenceAngle* were normalized in order to fit in the *byte* interval [0, 127], while the *ridgeCount* with a neighbor difficultly exceeds the *byte* maximum value and so, was not sampled. This sampling also prevents the sum of the feature differences (instead stored inside a *short*) from exceeding the capacity of the Java Card^{TM} data types.

To test our algorithm (Sect. 4.3) we did not impose a minimum number for the minutiae in the templates to be matched, since we decided to have no rejected fingerprints during the enrollment or the match phases, but in real world applications, it is surely fundamental to require mandatory minimum information.

The EEPROM allocation space to be reserved on the card for our MoC applet (after its instantiation) is about 10 Kb using 20 minutiae, about one-seventh of the whole EEPROM memory in our test smartcard. Every minutia is stored as a Java Card^{TM} object, so this storage occupation can be reduced by using a simpler data structure: with 20 minutiae and 8 neighbors for each minutia, a 640 bytes array is enough, since each of the four features of a neighbor can be represented with a *byte*. The use of a plain data structure will also speed-up the memorization process of the template inside the smartcard.

4 Performances

4.1 Fingerprint databases

Usually, the performances of a fingerprint matching algorithm are tested with a proprietary fingerprint database and so, it is often impossible to compare their results with uniform operating conditions. An algorithm could achieve good performances for its chosen image acquisition system, so certifying the overall (acquisition plus matching) system performances, but it can also be useful to test the functioning over a common reference. For this reason, we decided to use the fingerprint databases from the *Finger Verification Competition* 2002 [42] (FVC2002) and 2004 [37] (FVC2004) editions, since, as we know, this is the only public benchmark (together with FVC2000 [43] edition) allowing industrial, academic and independent developers to unambiguously compare their algorithms. In this way, we can have also a comparison and show the performance drop in respect to the algorithms in the competitions, which are indeed not suitable for the smartcard constraints.

Each of the FVC databases is 110 fingers wide and 8 impressions per finger deep (880 fingerprints in all); the real benchmark is constituted by fingers numbered from 1 to 100 (set A), since the fingers from 101 to 110 (set B) have been made available to the competition participants to allow a parameter tuning before the submission of their algorithms.

In our tests, we used the two databases from FVC2002 collected respectively with the optical sensor ‘‘TouchView II’’ by Identix, and the optical sensor ‘‘FX2000’’ by Biometrika; moreover, we show the preliminary results achieved with one of the FVC2004 databases, synthetically generated with the *SFINGE* software [2].

Other large public domain databases are the *National Institute of Standards and Technology* (NIST) collections, even if they are not completely suitable for systems working on live-scan images: for example, the NIST DB 4 [44] contains images scanned from enrolled inked impressions. One more public fingerprint database is described in [45] together with a signature database, since the purpose of this project promoted by the *Biometric Research Laboratory—ATVS* (Universidad Politecnica de Madrid) was to collect a bimodal database. It is made of 79,200 fingerprint samples acquired with two different sensors.

Moreover, we personally collected a small database using the FX2000 optical scanner model by Biometrika [46], directly acquiring some images from people poorly trained in biometric devices and systems; our collection has 8 repetition for each of the 40 different fingers, for a total of 320 images. Subsequently, we will refer to this as the ‘‘Internal Database’’.

In addition to the image databases already discussed, we analyzed our algorithm in respect to that proposed in [35], using also the same proprietary database provided to us by the authors. This very good quality image collection is made up of 10 different repetitions of 55 fingers, for a total of 550 samples; even these images have been collected using the same FX2000 scanner. From now on, we will refer to this as the ‘‘Hybrid Database’’.

We present all these results to show also how much different databases can affect the performances of a match algorithm.

4.2 Performance indicators

Commonly, a typical biometric verification system commits two types of errors: mistaking the biometric measurements from two different fingers to be from the same one (false acceptance), and mistaking two biometric measurements from the same finger to be from two different fingers (false rejection). The probability of these events are respectively defined as the FAR and FRR.

A further performance description can be given by the EER, which denotes the error rate for which FAR and FRR are identical at some operating point t (the match threshold). The trade-off curve associated with a fingerprint authentication system is known as the *receiver operating characteristic* (ROC) curve. The ROC curve plots the FAR against the FRR for the same match threshold value, and can be generated by obtaining FAR and FRR values under many different operating points.

Another important factor to be considered for MoC algorithms is the average matching time, clearly because of the hardware limitations. The average enrollment time is, instead, less important for our purposes because the enrollment is executed only once at the smartcard personalization phase.

Also, the maximum template size is of significant importance, because the smartcard EEPROM memory is usually limited to 32–72 kb and is also used to store the match procedure code and other applications. Usually, the algorithms not using correlation based methods have an average template size of few kilobytes [38], since they do not directly need the fingerprint images. The template size for our algorithm is discussed in Sect. 3.4.

Other interesting performance indicators can be derived by increasing or decreasing the matching

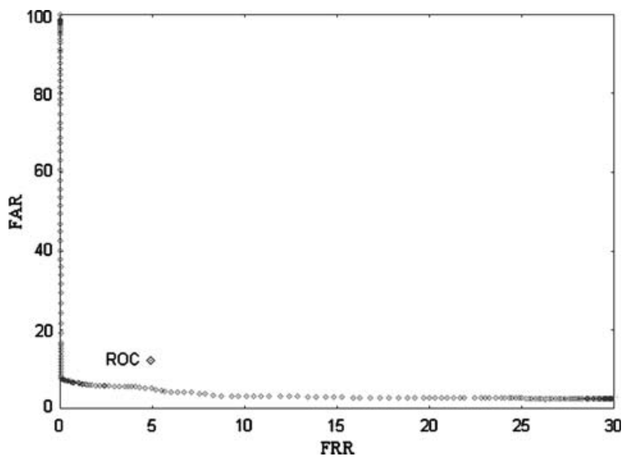


Fig. 7 The receiver operating characteristic (*ROC*) curve for the fingerprint verification competition 2002 (*FVC2002*) FX2000 database (using the set A images)

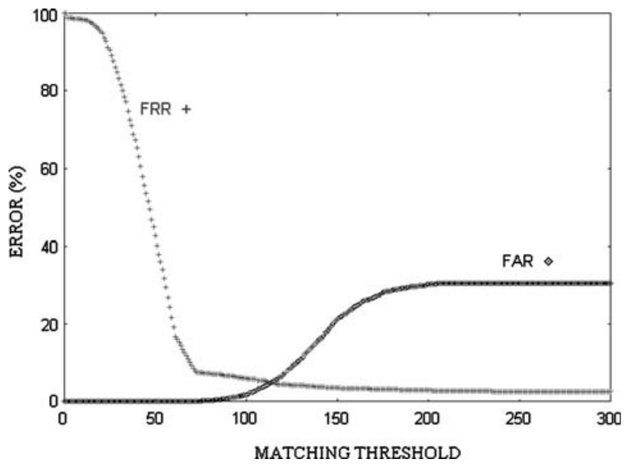


Fig. 8 The false acceptance rate (*FAR*) and false rejection rate (*FRR*) curves for the FVC2002 FX2000 database. Reported equal-error rate (*EER*) (4.6%) is at their intersection point (using the set A images)

threshold and then observing the algorithm behaviour: for example the FAR100 (the lowest achievable FRR for a FAR ≤1%), FAR1000 (the lowest FRR for FAR ≤0.1%), ZeroFAR (the lowest FRR for FAR = 0%) and finally ZeroFRR (the lowest FAR for FRR = 0%).

All these performance indicators can be substantially altered in our algorithm by modifying some parameters.

4.3 Performance results

The distribution between FAR and FRR tests can greatly influence the declared performances. For this reason we decided to run the same tests as executed during the FVC competition (see [37] for more information): 2,800 iterations to find the FRR and 4,950 to find the FAR, for a total number of 7,750 match tests executed between exactly the same fingerprint images as in the international competition. For our main parameter configuration and our purposes, we were essentially interested in giving the best FAR1000 performance.

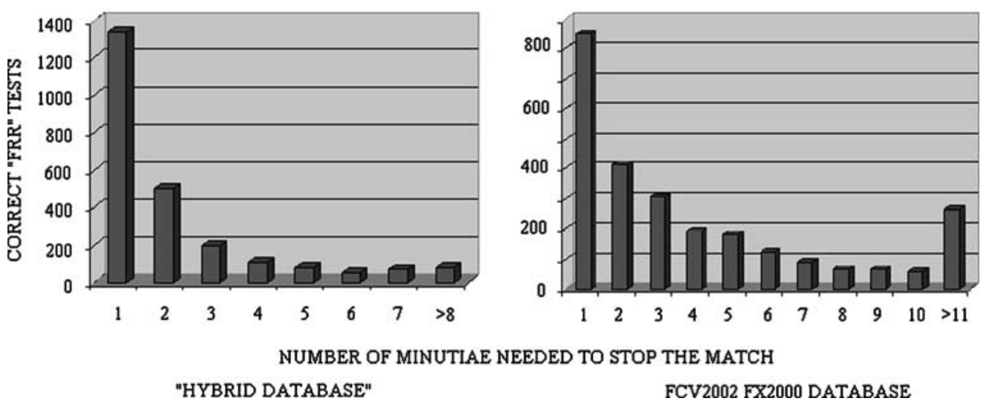
Here we summarise some of the results obtained from the FVC2002 database collected with the FX2000 optical scanner, which show some of the best global performances among the other FVC databases:

- *EER* 4.6%
- *FAR100* 5.5%
- *FAR1000* 6.3%

Figures 7 and 8 show respectively the ROC and the FAR–FRR curves for the tests on the FX2000 database.

One very important aspect to be highlighted is that using the two stopping conditions of Sect. 3.3 does not weaken the overall security: only about 1.6% of the total false matches accepted were introduced by

Fig. 9 Distribution of the minutiae number needed to positively stop the match, with respect to correct FRR tests. From this graphic we can derive the average match time: the min–max times to match a single minutia are 0.3–1 s



relaxing the problem and stopping the execution before the complete minutiae lists scan. This value was measured at EER conditions on the FX2000 database of FVC2002.

All the security performance results were obtained running the tests on a PC to reduce the experiment time, thanks to the fact that the algorithm code was the same as the one implemented directly on the smartcard (where, therefore, the same exactly results are achievable): the basics of Java Card^{TM} are a subset of Java^{TM} language. Our performance results can compete only with those of the last classified algorithms in FVC2002 (see [42]), but they are developed without the smartcard environment restrictions. In FVC2002 the performance has been calculated on a PentiumIII at 933 MHz, which is indeed much faster than our smartcard CPU, without considering other architecture gaps. Comparing our architecture with the test environment of FVC2002, we can see that our algorithm is faster than all the FVC2002 algorithms.

We noticed that a minimum time of about 0.3 s is needed for the entire match procedure and this result can be achieved frequently using a good enrollment fingerprint image, since in this case the two stopping conditions of Sect. 3.3 can be met earlier in the computation. Maximum time is instead more than 20 s, but this result, unfeasible for real time smartcard verification, is performed only in circumstances where the two acquisitions belong to different fingerprints (not interesting for our purposes), or when the image acquired at the verification phase is very disturbed: these premises often prevent the algorithm from stopping quickly without exploring all the minutiae pairings (thus, the image quality affects the average match time).

In Fig. 9 we report, for two databases, the distribution of the minutiae number needed to stop the match, with respect to the correct FRR tests. This distribution was computed at FAR1000 conditions, so the correct FRR tests are 93.7% of the total FRR tests (failed FRR tests are 6.3%). From on-card experiments, we verified that the maximum time needed to match one minutia is about 1 s, while the minimum time is 0.3 s. Thus, for the hybrid database, we can observe from Fig. 9 that an on-card matching time of about 0.3–8 (min–max) s is obtained for nearly all of the matches (more than 95%). Regarding instead the FX2000 database, 85% of the matches are executed within 8 s, and 60% within a maximum of 3 s (Fig. 9). The results obtained over the hybrid database (see Sect. 4.3.1) are better than those of the FVC2002 FX2000 database, since the average image quality of the first one is evidently higher.

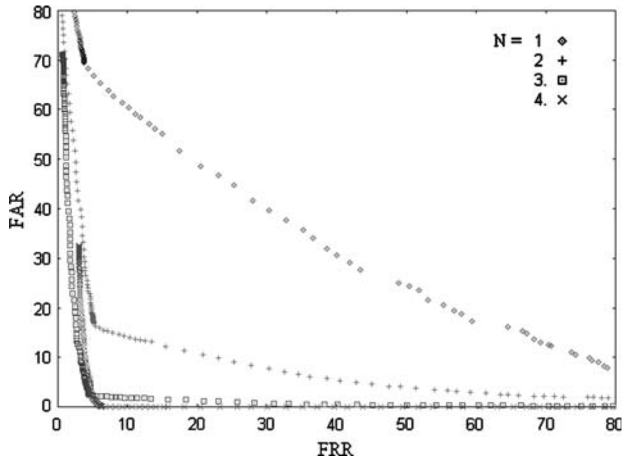


Fig. 10 The receiver operating characteristic (*ROC*) curves changing the *N* threshold in Sect. 3; we decide if two minutiae can be matched as soon as *N* neighbor pairs in their neighborhoods have been matched

The enrollment time using a template with 20 minutiae is about 3.5 s; this time can be further reduced by using a simple data structure like an array. Both the match and enrollment average times can be slightly influenced by the particular CAD model.

In Fig. 10 we show some ROC curves obtained by changing the *N* threshold explained in Sect. 3.3.

After that, we tested our algorithm using all the images from sets A and B (110 instead of 100) of the FVC2002 FX2000 database, for a total of 9,075 matches, distributed in 5,995 to test FAR and 3,080 to test FRR; this distribution is derived with the same FVC criteria. The performances on 100 images at the beginning of this section were slightly improved, showing a good stability of the algorithm:

- *EER* 4.6%
- *FAR100* 5.5%
- *FAR1000* 6.4%

After that, we additionally tested the algorithm using the FVC2002 database acquired with the TouchView II optical sensor. Using the 100 images of the database set A (7,750 tests), we achieved an EER of 6.7%.

- *EER* 4.6%
- *FAR100* 5.7%
- *FAR1000* 6.7%

In this way we proved that these good results can be carried out from at least two different fingerprint scanners, further proving the stability of our procedure.

With our internal database, we performed all the possible tests between different fingers (for a total 51,040 matches to test FAR) and between the same fingers (a total of 1,120 matches to test FRR). The results obtained are slightly better than those achieved on the FVC databases:

- *EER* 3.0%
- *FAR100* 3.3%
- *FAR1000* 5.0%

During the tests, we observed that using a good quality enrollment image considerably improves the performances. This hypothesis of functioning is not pervasive at all and is easily applicable, since the enrollment phase is accomplished only once at the release/initialization of the smartcard, and can also be controlled by a quality checker software module or possibly even by a human operator.

At the end, we analyzed the performances with the first 100 images (7,750 tests) of the FVC2004 database generated with the SFINGE software. Preliminary results show that the degradation performance is significant, achieving a FAR100 of 25% and a FAR1000 of 29%. These results are not surprising, since the databases from FVC2004 were intentionally made more difficult than FVC2002 by exaggerating perturbations such as skin distortion and condition [38]: the top five algorithms score an average FAR1000 value of 0.4% in FVC2002, and about 4% in the FVC2004 Light category (considering the best database of each competition), thus, ten times more. Hardware limitations of the FVC2004 Light category are still not comparable to smartcard resources (Sect. 3.1).

4.3.1 A comparison with related work

We also compared our work with the algorithm in [35], developed to be executed in a similar Java Card^{TM} environment. To obtain a significant comparison, we tested our algorithm with the hybrid database granted to us by the authors. We achieved the following results:

- *EER* 0.24%
- *FAR100* 0.40%
- *FAR1000* 0.61%

The tests were executed using the same FVC2002 guidelines for the match distribution between FAR and FRR tests. We nearly reduced to one-fourth the EER percentage of 0.8% achieved for the algorithm shown in [35] (Table 1), even if this algorithm uses two different matching modules and at the end merges their results. Even executing all the possible FAR and FRR tests with the database images, the EER found with our algorithm was still significantly lower, at about 0.4%.

This last experiment also confirms that the quality of the database images (good, in this case) can greatly improve the global reliability performances.

In Table 2 we show all the reliability results previously presented.

Table 1 EER comparison between our algorithm and the algorithm in [35]

Algorithm	EER on hybrid database
Hybrid matching algorithm [35]	0.8%
Our algorithm	0.24%

5 Conclusion

In this paper we have proposed a new fingerprint minutiae matching algorithm, thought and developed to overcome the Java Card^{TM} platform restrictions. Our algorithm, tolerant to typical match problems such as rotation, translation and non-linear deformations, achieves a very good speed performance for the smartcard environment (0.3–8 s for the most of positive match tests). A FAR less than 0.1% and a FRR of about 5–6% (the FAR1000 score) make the algorithm implementation feasible in smartcard applications for identity verification. The reliability can be greatly improved even further using a good enrollment image. Moreover, the requirements of the algorithm perfectly fit the exiguous resources of this environment.

Our algorithm shows an asymmetric behavior with respect to the execution time: the procedure is stopped as soon as, with the aid of two particular conditions, the templates are considered to belong to the same finger,

Table 2 Overall performance results

Fingerprint database	EER (%)	FAR100 (%)	FAR1000 (%)
FVC2002 FX2000 database (100 images)	4.6	5.5	6.3
FVC2002 FX2000 database (110 images)	4.6	5.5	6.4
FVC2002 TouchViewII database (100 images)	4.6	5.7	6.7
Internal database	3.0	3.3	5.0
Hybrid database	0.24	0.40	0.61

and so the algorithm usually stops earlier in correct FRR tests (between different images of the same finger) and later in correct FAR tests (between different fingers).

The MoC architecture provides, moreover, better security, since the match decision is taken inside the card and the user template is not stored in a centralized database.

A MoC architecture with our matching algorithm has already been successfully introduced in a digital signing tool developed at the *Institute for Informatics and Telematics* (National Research Council of Pisa [47]). The biometric verification unblocks the certificate stored in the card and associated with the smart-card owner. Consequently, it can be retrieved from the PC to digitally sign a document.

Our future works will include discovering new minutiae features, correlated with the core point or with texture information near the minutia point, that could improve the reliability and the speed by trying to match a lower number of neighborhoods. We are also considering to test the inclusion of a different matching algorithm, like the one described in [9], to perform a multimodal verification.

Acknowledgments We would like to thank the organizers of Fingerprint Verification Competition (FVC) [42, 43, 37] who have kindly granted to us the use of all of the fingerprint image databases of the different editions; in particular, we would like to acknowledge Raffaele Cappelli, Dario Maio and Davide Maltoni from the Biometric Systems Lab (University of Bologna). Moreover, we would like to thank Tommaso Cucinotta and Riccardo Brigo from ReTiS Lab of Sant'Anna School of Advanced Studies (Pisa), who have kindly granted to us the use of the fingerprint image database employed to test their algorithm in [35]. Thanks are due also to the anonymous referees that, with their interesting comments, helped to improve the readability and the technical contribution of the paper.

References

1. Bistarelli S, Santini F, Vaccarelli A (2005) An asymmetric fingerprint matching algorithm for java card^{TM}. In: Proceedings of AVBPA 2005. Springer, Berlin Heidelberg New York, pp 279–289
2. Jain AK, Maltoni D (2003) Handbook of fingerprint recognition. Springer, Berlin Heidelberg New York, Secaucus, NJ
3. Lee HC, Gaensslen RE (1991) Advances in fingerprint technology. Elsevier, New York
4. Pankanti S, Prabhakar S, Jain AK (2002) On the individuality of fingerprints. IEEE Trans Pattern Anal Mach Intell 24(8):1010–1025
5. Hatano T, Adachi T, Shigematsu S, Morimura H, Onishi S, Okazaki Y, Kyuragi H (2002) A fingerprint verification algorithm using the differential matching rate. In: ICPR '02: proceedings of the 16th international conference on pattern recognition (ICPR'02), vol 3. IEEE Computer Society, Washington, pp 799–802
6. Bazen AM, Verwaaijen GTB, Gerez SH, Veelenturf LPJ, van der Zwaag BJ (2000) A correlation-based fingerprint verification system. In: Proceedings of the ProRISC workshop on circuits, systems and signal processing, Veldhoven, The Netherlands. STW Technology Foundation, pp 205–213
7. Takeda M, Uchida S, Hiramatsu K, Matsunami T (1990) Finger image identification method for personal verification. In: 10th international conference on pattern recognition, vol 1. IEEE Computer Society,Washington, DC, pp 761–766
8. Kaymaz E, Mitra S (1993) Analysis and matching of degraded and noisy fingerprints. In: Tescher AG (ed) proceedings of SPIE, vol 1771, Applications of digital image processing XV, pp 498–509
9. Jain AK, Prabhakar S, Hong L, Pankanti S (2000) Filterbank-based fingerprint matching. IEEE Trans Image Process 9(5):846–859
10. Stosz JD, Alyea LA (1994) Automated system for fingerprint authentication using pores and ridge structure. In: David Murley J, Richard J. Mammone (eds) proceedings of automatic systems for the identification and inspection of humans (SPIE), vol 2277. IEEE Computer Society, Washington, DC, pp 210–223
11. Ratha NK, Karu K, Chen S, Jain AK (1996) A real-time matching system for large fingerprint databases. IEEE Trans Pattern Anal Mach Intell 18(8):799–813
12. Bistarelli S, Boffi G, Rossi F (2003) Computer algebra for fingerprint matching. In: International conference on computational science ICCS03, vol 2657/2003, pp 811–820
13. Jain AK, Hong L, Bolle RM (1997) On-line fingerprint verification. IEEE Trans Pattern Anal Mach Intell 19(4):302–314
14. Luo X, Tian J, Wu Y (2000) A minutiae matching algorithm in fingerprint verification. In: 15th international conference on pattern recognition, vol 4. IEEE Computer Society, Washington, DC, pp 210–223
15. Jiang X, Yau W-Y (2000) Fingerprint minutiae matching based on the local and global structures. icpr 02:1042–1045
16. Ratha NK, Bolle RM, Pandit VD, Vaish V (2000) Robust fingerprint authentication using local structural similarity. In: 5th IEEE workshop on applications of computer vision. IEEE Computer Society, Washington, DC, pp 29–34
17. Hrechak AK, McHugh JA (1990) Automated fingerprint recognition using structural matching. Pattern Recognit 23(8):893–904
18. Fan K-C, Liu CW, Wang Y-K (2000) A randomized approach with geometric constraints to fingerprint verification. Pattern Recognit 33(11):1793–1803
19. Willis AJ, Myers L (2001) A cost-effective fingerprint recognition systems for use with low-quality prints and damaged fingerprints. Proc Pattern Recognit 34(2):255–270
20. Kovacs-Vajna ZM (2000) A fingerprint verification system based on triangular matching and dynamic time warping. IEEE Trans Pattern Anal Mach Intell 22(11):1266–1276
21. Bhanu B, Tan X (2003) Fingerprint indexing based on novel features of minutiae triplets. IEEE Trans Pattern Anal Mach Intell 25(5):616–622
22. Jain AK, Ross A, Prabhakar S (2001) Fingerprint matching using minutiae and textures features. In: Proceedings of international conference on image processing (ICIP), Thessaloniki, Greece, pp 282–285
23. ISO 7816: integrated circuit(s) cards with contacts—part 1, 1998. International Standardization Organization, JTC 1/SC 17

22. Enrique Ortiz C (2003) An introduction to Java Card™ technology, Parts 1-2-3, Java Developer Web Site
25. Chen Z (2000) Java Card technology for smart cards: architecture and programmer's guide. Addison-Wesley, Longman Publishing Co., Inc., Boston
26. Ratha NK, Connell JH, Bolle RM (2001) An analysis of minutiae matching strength. In: AVBPA '01: proceedings of the 3rd international conference on audio- and video-based biometric person authentication. Springer, London, pp 223–228
27. Kocher PC, Jaffe J, Jun B (1999) Differential power analysis. In: CRYPTO '99: proceedings of the 19th annual international cryptology conference on advances in cryptology. Springer, London, pp 388–397
28. Pan SB, Moon D, Gil Y, Ahn D, Chung Y (2003) An ultra-low memory fingerprint matching algorithm and its implementation on a 32-bit smart card. In: IEEE transactions on consumer electronics, vol 49. IEEE Computer Society, Washington, pp 453–459
29. Moon YS, Ho HC, Ng KL, Wan SF, Wong ST (2000) Collaborative fingerprint authentication by smart card and atrusted host. In: Canadian conference on electrical and computer engineering, vol 1. IEEE Computer Society, Washington, pp 108–112
30. Reisman J, Uludag U, Ross A (2005) Secure fingerprint matching with external registration. AVBPA 720–729
31. Ross A, Jain AK, Reisman J (2003) A hybrid fingerprint matcher. Pattern Recognit J 36(7):1661–1673
32. Ishida S, Mimura M, Seto Y (2001) Development of personal authentication techniques using fingerprint matching embedded in smart cards. In: EICE transactions on information and systems, vol E84-D, pp 812–818
33. Tang TY, Moon YS, Chan KC (2004) Efficient implementation of fingerprint verification for mobile embedded systems using fixed-point arithmetic. In: SAC '04: proceedings of the 2004 ACM symposium on applied computing. ACM Press, New York, pp 821–825
34. Yang S, Verbauwhede I (2003) A secure fingerprint matching technique. In: WBMA '03: proceedings of the 2003 ACM SIGMM workshop on biometrics methods and applications. ACM Press, New York, pp 89–94
35. Cucinotta T, Brigo R, Di Natale M (2004) Hybrid fingerprint matching on programmable smart cards. TrustBus 232–241
36. van Wamelen P, Li Z, Iyengar S (2000) A fast algorithm for the point pattern matching problem. Technical report 1999-28, Louisiana State University, Dept. of Mathematics
37. Maio D, Maltoni D, Cappelli R, Wayman JL, Jain AK (2004) Fvc2004: third fingerprint verification competition. ICBA 1–7
38. Cappelli R, Maio D, Maltoni D, Wayman JL, Jain AK (2006) Performance evaluation of fingerprint verification systems. In: IEEE transactions on pattern analysis machine intelligence, vol 28. IEEE Computer Society, Washington, pp 3–18
39. National Institute of Standards and Technology. *User's guide to NIST fingerprint image software (NFIS)*. NISTIR 6813
40. NIST/Biometric Consortium Interoperability, Assurance, and Performance Working Group 7. Java Card™ Biometric API White Paper (Working Document), August 2002. Version 1.1
41. Java Card Forum™ official, web site: http://www.javacardforum.org/
42. Maio D, Maltoni D, Cappelli R, Wayman JL, Jain AK (2002) Fvc2002: second fingerprint verification competition. ICPR (3):811–814
43. Maio D, Maltoni D, Cappelli R, Wayman JL, Jain AK (2002) Fvc2000: fngerprint verification competition. IEEE Trans Pattern Anal Mach Intell 24(3):402–412
44. Watson CI, Wilson CL (1992) Nist secial dtabase 4. Fingerprint database
45. Ortega-Garcia J, Fierrez-Aguilar J, Simon D, Gonzalez J, Faundez-ZanuyM, Espinosa V, Satue A, Hernaez I, Igarza JJ, Vivaracho C, Escudero D, Moro QI (2003) Mcyt baseline corpus: a bimodal biometric database. IEE Proc Vis Image Signal Process 150(6):395–401
46. Biometrika s.r.l., web site: http://www.biometrika.it
47. Institute of informatics and telematics (c.n.r.), biometric group web site: http://www.iit.cnr.it/organizzazione/sicurezza_informazione/ biometria/index.htm

Author Biographies

Stefano Bistarelli is Associate Professor of Computer Science at the Department of Science of the University "G. d'Annunzio" in Pescara and External Researcher of the IIT of C.N.R., Pisa. He obtained his Ph.D. in Computer Science in 2001 at the Computer Science Department of the University of Pisa. His thesis was awarded by the Italian Association of Artificial Intelligence and by the Italian Chapter of the EATCS. His research interests range from Artificial Intelligence to Programming Languages, with particular attention to Soft Constraints and Security issues. Recently, some of his work has been published in the LNCS 2962 book by Springer.

Francesco Santini is currently (since 2006) a Ph.D. student in Computer Science and Engineering at the IMT (Institutions, Markets and Technologies) Institute of Advanced Studies in Lucca. In 2003, he graduated from the University of Pisa in Computer Science, and afterwards he had been a Research Fellow for 2 years at the Institute of Informatics and Telematics of C.N.R., Pisa. His interests cover Biometric Identity Verification and, more generally, Security related issues. He is interested also in Network Management issues, with a special attention on "Quality of Service" and related formal models/frameworks based on Constraint Programming and Soft Constraints.

Anna Vaccarelli is Senior Technologist at the Institute of Informatics and Telematics (IIT) C.N.R., Pisa (since 2001), where is also the head of the Information Security Research Group. She received her doctoral degree in Electronic Engineering from Pisa University in 1984, and has been a researcher at C.N.R. since 1987. She has been coordinating several research projects and scientific agreements in information security. She was Contract Professor at Pisa University for the course ‘‘Aperture antennas’’. Currently, she is professor at the ‘‘Internet technologies’’ Master organized by IIT and Pisa University. She is co-author of several scientific publications concerning information security.