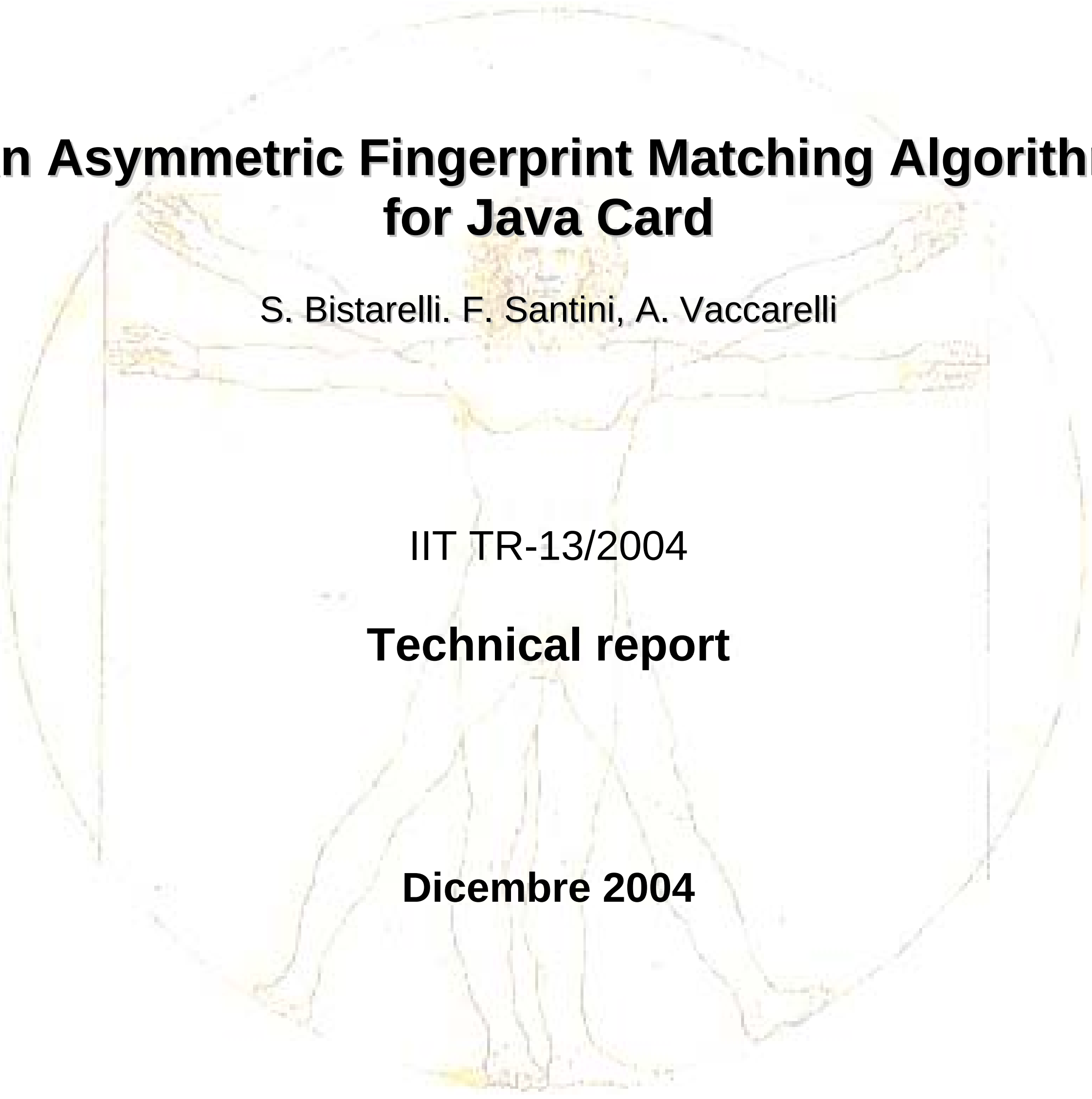
Consiglio Nazionale delle Ricerche



An Asymmetric Fingerprint Matching Algorithm for Java Card

S. Bistarelli. F. Santini, A. Vaccarelli

IIT TR-13/2004

Technical report

Dicembre 2004

Istituto di Informatica e Telematica

An Asymmetric Fingerprint Matching Algorithm for Java Card™

Stefano Bistarelli[1,2], Francesco Santini[2] and Anna Vaccarelli[2]

[1] Dipartimento di Scienze, Università "G. D'annunzio" di Chieti-Pescara, Italy
bista@sci.unich.it

[2] Istituto di Informatica e Telematica, CNR, Pisa, Italy
stefano.bistarelli,francesco.santini,anna.vaccarelli@iit.cnr.it

Abstract. A novel fingerprint matching algorithm is proposed in this paper. The algorithm is based on the minutiae local structures, that are invariant with respect to global transformations like translation and rotation. Match algorithm has been implemented inside a smartcard over the Java Card™ platform, meeting the individual's need for information privacy and the overall authentication procedure security, since the card owner biometric template never leaves the private support device and the match is computed inside a secure environment. The main characteristic of the algorithm is to have an asymmetric behaviour between correct positive matches (between two same fingerprint samples) and correct negative matches (between two different fingerprint images): in the first case, the match procedure stops as it finds that images belong to the same fingerprint, gaining high speed efficiency, while in the second case the verification process lasts longer, exploring all the minutiae pairings. The performances in terms of authentication reliability and speed have been tested on the databases from the Fingerprint Verification Competition 2002 edition (FVC2002) by taking in account the different hardware to run the algorithms. Moreover, our procedure has showed better reliability results when compared on a common database with a related algorithm developed specifically for Java Card™.

Keywords: smartcard, fingerprint matching, Java Card™.

1 Introduction

The term "biometrics" is commonly used today to refer to the authentication of a person by analyzing thr physical characteristics (like fingerprints) or the behavioral characteristics (like the voice or the gait).

Since these characteristics are unique to an individual, their measurement provide 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 be even forgotten by the owner preventing the whole identification process; biometric characteristics are instead generally more difficult to be duplicated and they naturally "follow" always the owner.

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, like permanence (measurement should be invariant with time), uniqueness (different values for different persons), universality (everyone should have this trait), acceptability (if people is willing to accept this technology), performance (achievable recognition accuracy and system requirements) and circumvention (how easy is 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 represents also the most mature and, consequently, explored technology among the others. There are two main applications involving fingerprints: fingerprint verification and fingerprint identification. While the goal of fingerprint verification is to verify the identity of a person, the goal of fingerprint identification is to establish his or her identity. In the rest of the paper we will focus our attention only on verification, since identification cab be generally seen as a 1 to N verification.

Implementing a biometric verification inside a smart card is notoriously difficult, since the templates tend to eat-up a large part of the card's memory, while the biometric verification algorithms are nearly beyond the processing capabilities of standard processors. With *Match On Card* (MOC) technology the fingerprint template is stored within 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. Our fingerprint verification MOC algorithm has been developed to work in this very strict-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, that would be a difficult task to be implemented inside a smartcard. Moreover, it shows on the average asymmetric execution time between correct positive match (same fingerprint) and correct negative match (two different fingers), and this because the match procedure stops immediately when few minutiae pairs result in a positive match. If this check doesn't succeed, for example if the two fingerprints 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.

Formerly the algorithm has been written in Java™ and later has been ported on Java Card™ platform; 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-time verification applications.

1.1 Paper structure

This paper is organized as follows: in Section 2 we present some general background information about fingerprints, like their use through history, their formation process and uniqueness (Section 2.1), and their appearance (Section 2.2); from 2.3 to 2.6 we respectively explore the background about systems for fingerprint automatic treatment, common fingerprint matching problems and a classification in literature about the algorithms to solve these problems, the Java Card™ platform characteristics and finally some comments about the fingerprint matching implementation on a smartcard.

Section 3 features an overview of our fingerprint matching algorithm developed expressly for the Java Card™ platform: in 3.1 we describe the information chosen to represent and match two fingerprints, in 3.2 we describe the matching procedure and in 3.3 we show the implementation problems and solutions.

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

Section 5 draws final conclusions and the intents about future works.

2 Background

2.1 Fingerprint history

Fingerprints are graphical flow-like ridge patterns present on the human tips of fingers and they are fully formed at about seven months of fetus development. Like physical appearance, they are a part of an individual's phenotype: the genes determine the pattern general characteristics, while the flow of amniotic fluids around the fetus changes the microenvironment around the skin; the fingerprint's finer details are determined by this changing microenvironment [1]. This process leads to the uniqueness of fingerprints even between homozygotic twins, and therefore their help can be used to securely distinguish the individual identity among others.

In history, even if thumb prints were used on clay seals in China to sign documents in 3rd century B.C., we can conveniently assume that modern fingerprint science started in the 16th century [6–8]. English plant morphologist Nehemiah Grew published a paper in 1684 reporting his systematic study on fingerprint structures, which is believed to be the first scientific paper about them [2].

After him, in 1788 Mayer theorised that the arrangements of friction ridges were unique [9].

In 1823 Purkinje proposed the first fingerprint classification [2] into nine categories according to the global ridge configurations. He recognized the classification element of friction ridge formations but he did not associate the friction ridges to a means of personal identification.

Later, in 1880 Henry Fauld scientifically suggested their individuality and uniqueness [2]; few weeks after, William Herschel wrote “Skin Furrows of the Hand” asserting he had used fingerprints officially as “sign-manuals” or signatures, sanctioning the idea’s practicality [2]. In 1888 Sir Francis Galton, after conducting extensive study, introduced the minutiae features for single fingerprint classification [2]. Sir Edward Henry, starting from Galton’s works and consulting with him, set out to solve the problem of fingerprint classification and them fingerprints for criminal identification (dactyloscopy) in England and Wales in 1901. At the time, an alternative to fingerprints was the *Bertillonage* system, also known as *Anthropometry.* Developed by Alphonse Bertillon in 1879, Bertillonage consists of a meticulous method of measuring body parts to identify the criminals [10].

All the classification schemes currently used by police agencies are variants of the so-called Henry’s classification scheme. The five most common classes of Galton-Henry classification scheme are *arch*, *tented arch*, *left loop*, *right loop* and *whorl* and nearly all of the human fingerprints can fall into one of this sets based on global ridge flow pattern (see Figure 1).

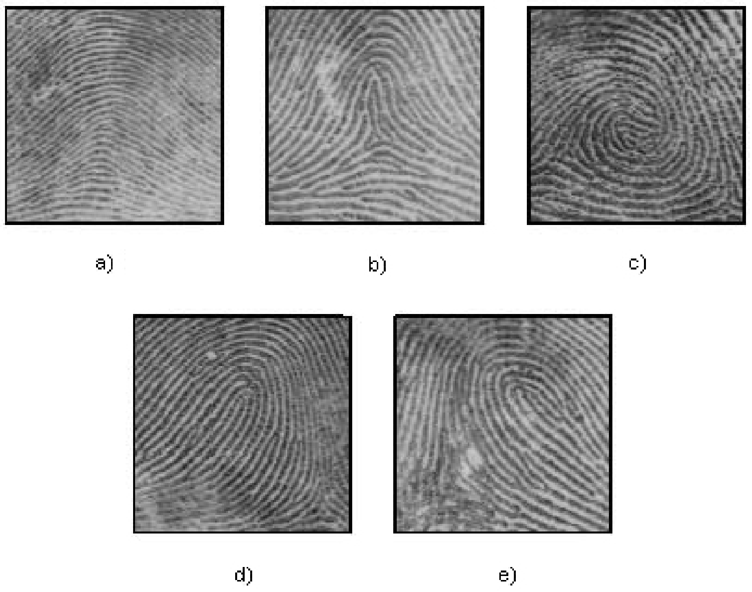


Fig. 1: Fingerprint pattern classes.
Arch (a), Tented Arch (b), Whorl (c), Left Loop (d), Right Loop (e).

New guidelines for the admission of scientific evidences were established in 1993 by the United States Federal Court ruling in Daubert vs. Merrill Pharmaceutical Company [13]. These rules require scientists to address the reliability and validity of the methods used in their analysis.

The two fundamental premises on which fingerprint identification is based are: *(i)* fingerprint details are permanent, and *(ii)* fingerprints of an individual are unique. The validity of the first premise has been established based on the anatomy and morphogenesis of friction ridge skin (except due to accidents such as bruises and cuts on the finger tips). The notion of the fingerprint individuality has been widely accepted, based on manual inspection (by experts) of millions of fingerprints, and there are several mathematical models which describe the probability of a same particular fingerprint configuration [3], but it still clearly remains an empirical observation and not an established fact.

2.2 Fingerprint appearance and minutiae

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 μm to 300 μm and typically the period of a ridge/valley cycle is about 500 μm. If analyzed at global level, almost all of the patterns exhibit one or more regions characterized by distinctive shape and called *singular regions*; these regions can be classified into three typologies according to the shape: loop, delta and whorl are characterized respectively by shape $\frown$, Δ and O. Particular presence of singular regions define whole fingerprint class: for example, whorl class can be identified by two loop zones or one whorl zone and two delta regions (see Figure 2); the natural proportion of fingerprints in classes arch, tented arch, left loop, right loop and whorl is respectively 3.7%, 2.9%, 33.8%, 31.7% and 27.97% [11]. Furthermore there are some "ambiguous" fingerprints, whose exclusive class affiliation cannot be reliably stated even by human experts.

This classification can be used to speed up the identification process by reducing the number of comparisons and analyzing only fingerprints of the same type. Automated fingerprint classification is a very difficult pattern recognition problem due to the small inter-class variability and the large intra-class variability among ridge patterns.

At local level, other important features called *minutiae* refer to the ridges 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. Most frequently the minutiae types can be individuated by the terminations, where a ridge line ends, and bifurcations, where a ridge bifurcates forming a "Y" (see Figure 3).

Most important minutiae characteristics are the location coordinates inside the image, their form type (e.g. termination, bifurcation, island, etc) and the ridge orientation (in degree) respect to the minutia point in the image.

While singular regions can aid in the classification, the minutiae can be used in fingerprint matching since they represent some of the unique details of the ridge flow and are considered as singularity evidences. The type, direction and location of the minutiae are also the features used in individuality studies [3].

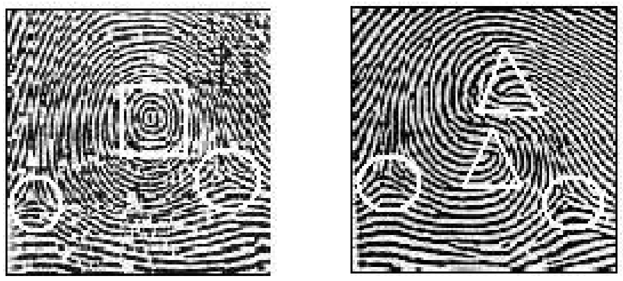


Fig. 2: Singular regions in whorl class.
In the two whorl fingerprint images, the whorl, loop and delta regions are highlighted respectively by a square, a triangle and a circle.

Minutiae matching is certainly the most well-known and adopted method for fingerprint automated matching, thanks to its strict analogy with the way forensic experts compare the fingerprints and its acceptance as a proof of identity in courts of law in almost all the countries. For this reason, many recognition automated systems, like FBI IAFIS [14] (*Integrated Automated Fingerprint Identification System*), use these details in their computation (see Section 2.3).

Although have been observed several types of different minutiae described by their shape, like dot, island, hook, lake, ridge crossing and multiple bifurcations, simply a coarser classification based on the ridge termination and bifurcation can be adopted during the automated comparison.

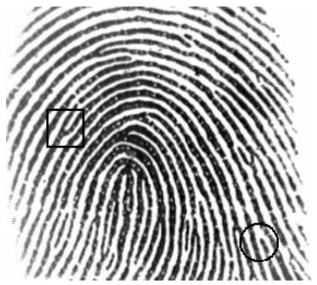


Fig. 3: Minutiae details.
The circle highlights a ridge termination, while the square shows a bifurcation.

2.3 Automated Fingerprint Identification System (AFIS) steps

An *Automated Fingerprint Identification System* (AFIS) is a biometric identification methodology that uses the digital imaging technology to obtain, store, and analyze the fingerprint data. The most "famous" of these systems, the FBI Integrated AFIS (IAFIS) [14, 15], maintains the largest biometric database in the world, containing the fingerprints and the corresponding criminal history information for more than 46 million subjects [43] (one ten-print card for each).

The most important services offered by AFIS are the "enrollment", used to insert a new person biometric measurement inside the system database, and the "verification"/"identification" phase, used to control the identity of a person by the aid of previously stored enrollment information.

AFIS first step of computation is known as *image acquisition*: in this part of the process, a user places his or her finger on an appropriate sensing device like a Charge-Coupled Device (CCD) scanner, a capacitative or a ultra-sound sensor [4], which captures the biometric raw data (*live-scan* fingerprint sensing). Nowadays, most civil and criminal AFIS accept the digital images acquired by directly sensing the finger surface with an electronic fingerprint scanner, although is still possible to find ink-techniques where the subject's finger is spread with black ink and then pressed against a paper card; later the card can be digitally scanned obtaining the final digital image to be used directly by the AFIS (*off-line* fingerprint acquisition).

The second step consists in the location and determination of the unique characteristics of the processed fingerprint raw image (*feature extraction*), like the minutiae details; the extracted features are then used in the biometric template creation. The template, in its generic definition, is a mathematical representation of fingerprint "uniqueness" to be used later during the matching phase.

High reliability systems often implements some sub-modules for this phase: in order to ensure that the performance of an automatic fingerprint verification system will be robust, it can be helpful to incorporate a fingerprint enhancement algorithm at the beginning of the feature extraction module [17, 19]). Moreover, the presence of a *quality checker* module can later check for the enrolling template quality, and consequently reject it if states that the template has a too low number of distinctive characteristics, or if their quality is too bad, forcing the user to re-enroll himself one more time; this check can be avoided during the matching phase, which clearly needs to be quick.

Enrollment phase ends with this second step and, in this way, the assured individual identity is assigned to the extracted template, which is then stored in the memory of the chosen storage device, like a smartcard, a central database, or within the fingerprint scanner itself.

The template acquired during enrollment is defined as the "reference template" and it is in some way associated with the system user identity, while the template acquired for verification phase, and compared with the database stored one, is defined as the "candidate template".

The last step in automated systems, executed only during verification or identification phase, is represented by *template matching*, discussed better in

Section 2.4. Concerning the identification service, the candidate template is typically compared with more than one reference template to assure the person identity among others. Vice versa, during the verification service the candidate template is compared only with that corresponding to the claimed identity.

All these steps are graphically reassumed in Figure 4.

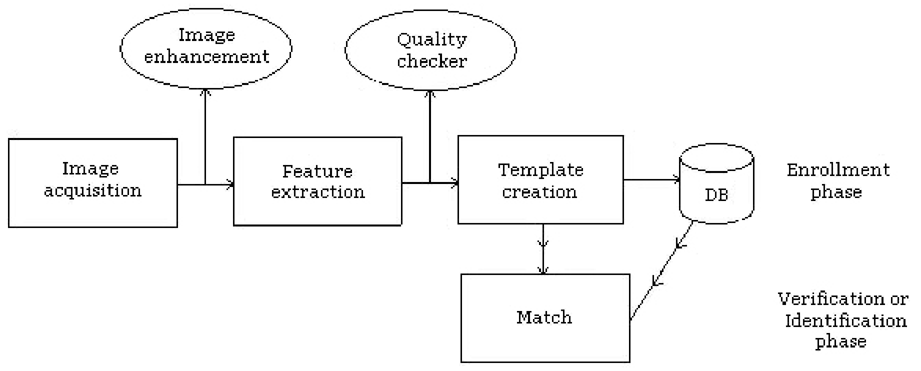


Fig. 4: Typical AFIS architecture.
Double arrows are used only for the identification/verification phase.

2.4 Fingerprint matching problems and solutions

In an AFIS, the fingerprint verification phase needs a further step after the first two discussed in Section 2.3: the third and final step of fingerprint recognition is the template matching. Here is where the system will attempt to verify the identity, by comparing the enrollment template corresponding to the claimed identity against the verification template real-time obtained. The response of a matcher in a fingerprint recognition system is typically a matching score s that quantifies the similarity or the difference between candidate and reference templates; the system 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" of matching algorithm.

Matching the fingerprint templates represents an extremely difficult problem because of the variability in different impressions of the same fingers; most important "noisy" factors introduced during image acquisition or fingerprint feature extraction are:

– *Displacement* depending on the different positioning of the finger on the acquisition sensor. It results in a global fingerprint area translation.
– *Rotation* depending on the different rotation in positioning the finger on the sensor between different acquisitions. It results in a global fingerprint area rotation.

- *Partial overlap.* Still because of the imperfect positioning of the fingerprint on the sensor, 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 smaller area.
- *Non-linear distortions* due to skin plasticity. Forces non-orthogonal to the sensor surface can cause the finger image to be distorted in some zones: the ridges can be stretched or compressed.
- *Pressure* on the sensor surface can result in a different thinning of the ridge pattern; samples can present also different scaling factor due to the pressure.
- *External factors.* Other external factors, not depending from the finger positioning on the sensor surface, can be represented by skin conditions as dryness, sweat, dirt, grease, or possible skin disease or consumption in manual workers or old persons. Even the atmospheric factors, like the humidity in the air, can negatively influence the image acquisition.
- *Feature extractor errors.* During the features extraction stage, the processing software can introduce some errors by adding some "false" minutiae (not really present in the fingerprint) or by not detecting true existing features.

Concerning about the algorithms to resolve fingerprint matching, the approaches can be classified in three main branches [4]:

Correlation-based matching: In this class, match is accomplished by superimposing two fingerprint images and computing the correlation between corresponding pixels [32, 33]; in this case the template is directly the finger image. Some problem could derive from non-linear distortions that make the impressions of the same finger significantly different in terms of global structure. Even the skin condition and pressure cause the image brightness and contrast to vary across different impressions;

Minutiae-based matching: Theory behind this algorithm class is fundamentally the same as for manual fingerprint examiners. Matching essentially consists in finding the maximum number of corresponding minutiae between the two templates (read the following discussion in this section);

Ridge feature-based matching: The approaches belonging to this family compare fingerprints in terms of ridge pattern features other than the minutiae or pixels intensity: some examples can be the shape features [31], spatial relationship and geometrical attributes of the ridge lines [30] (like the frequency and shape), texture information [28] and sweat pores [29]. In principle, the first two methods could be seen as subsets of this, since even the minutiae and pixels intensity can be considered as ridge features.

The minutiae extraction can be difficult in very low-quality fingerprint images and, in this case, other ridge features may be extracted more reliably than the minutiae, even if their discrimination efficacy is generally lower; sweat pores are instead very discriminatory, but require a high-resolution scanner during the image acquisition.

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 adds some information respect to this classic problem. Point pattern matching has been extensively examined because of its importance in many pattern recognition tasks; for example, *Hough transform-based* approach are quite popular for resolving the fingerprint match: the transformation parameters space that can relate two sets of points is firstly discretized, then the correct parameters can be derived by accumulating evidences in this space, and finally it will be chosen the most "voted" transformation [34]. The use of an algebraic technique for point pattern matching is instead described in [54].

Focusing only on the minutiae based algorithms, we can subdivide them in two more classes. *Global minutiae matching* (example in FBI IAFIS [15]) requires a first fingerprint alignment phase that subsequently admits to match the aligned template; 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 to be compared together. Relative pre-alignment is generally more effective because the features in stored template can finely help the process; an interesting approach based on minutiae ridge registering, that exploits ridge features for relative pre-alignment, is proposed in [26]: the minutiae representation is converted into a symbolic string and then the two strings are matched with a dynamic programming technique [50] finding their *edit distance*, and at last comparing this distance with a threshold.

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 as translation or rotation; it is consequently appropriate for matching without any a priori alignment. The local matching supplies simplicity, low computational complexity and higher distortion tolerance, while a global matching grants high distinctiveness. This algorithm class typically is based on a neighboring concept, for which the local structure is represented by the nearest minutiae characteristics: some examples are in [20–22]. Other solutions are based on grouping minutiae inside a bounding box [23, 24], or by the triangular matching proposed in [25].

Modern high-security verification applications typically adopt multimodal biometric systems, using in addition other fingerprint features beside the classical minutiae or even other biometric characteristics, like fingerprints from different fingers of the same person or one fingerprint and face of the same person (changing therefore also the biometric technology). The match result is derived from multiple sub-matcher modules working on the different biometric measurements, like the algorithm in [27] which uses the minutiae plus the texture information: these results are then merged in one single final decision (fusion of the matchers).

2.5 Java Card™

A smartcard [36] resembles classic a credit card in the size and the shape, but inside may contain an embedded 8-16 bit microprocessor (microprocessor card) under a gold contact pad on one side of the card, which grants computational power and enforces the access to the on-card data; the smartcards with memory only (memory card) offer simply a protection for data storing. Common smartcards may have up to 1kbyte of RAM, more than 64kbytes of ROM, 16-32kbytes of programmable EEPROM and a microprocessor running at 5-10MHz; in the ROM memory is usually masked the card operating system and other permanent data, while the EEPROM can be used to store applications or personal information on the card. The smartcard uses a serial interface and receives its power from external sources like a card reader, more generally defined as the "Card Acceptance Device" (CAD).

Java Card™ technology [37] adapts the Java™ platform for the 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 PC. This adaptation produces a global reduction of the platform functionalities and its result is a substantial decreasing of the expressive capacity. Java Card™ technology is described in Figure 5, which shows the different architectural layers present on the smartcard: the native smartcard operating system supports the entire on-card side of this technology. Each layer offers an interpretation service to the immediately above one.

The Java Card™ Runtime Environment (JCRE) specification defines the life-cycle of the Java Card™ Virtual Machine, the card application (in this technology called "applet") life-cycle, how the applets are selected and isolated from each other, the transactions and the object persistence and sharing. The JCRE provides a platform-independent interface to the services presented by the card's operating system. It consists in the Java Card™ Virtual Machine, Java Card™ API, and any vendor-specific extensions (see Figure 5).

The Java Card™ Virtual Machine (JCVM) is divided in two different parts: the on-card section interprets the bytecode and manages classes and objects, while the on-Pc section loads, verifies and further prepares the Java™ classes in a card applet for the successive on-card execution. JCVM specification defines a subset of the Java programming language (for example, it supports only *boolean*, *byte* and *short* primitive types) and additional constraints on many program attributes, like the maximum number of classes in a single package [37]. The Java Card™ API specification defines a small subset of the traditional Java programming language API: for example, there is no support for *String* class or for multiple threads. There are no wrapper classes like *Boolean* and *Integer*, and no *Class* or *System* classes [37]. In addition, the Java Card™ Framework defines its own set of core classes specifically created to support the Java Card™ applications.

The JCVM's lifetime coincides with that of the card itself: it begins some time after the card is manufactured and before it's issued to the cardholder; it ends only when the card is discarded or destroyed. The JCVM does not stop when power is removed from the card, as its state is retained in the card's non-volatile memory.

The Java Card™ platform is a multiple-application environment, as one or more Java Card™ applets may reside on the same card. An applet behaves as a server and it's passive, since it works only "on-demand". After a card is powered up, each applet remains inactive until its selection, and at that time the initialization may be done. The applet can be activated only when an external message has been dispatched to it (read the following APDU description).

Applet objects are created in the EEPROM persistent memory and so their state is saved between different CAD sessions, but it's a thousand times slower to write to the EEPROM than to RAM due to their different technology. Since a garbage collector is not always available in a Java Card™ 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's essential to reuse the same objects during several CAD sessions, otherwise the memory space will be sooner or later saturated.

Different applets can safely coexist in the same card, since each applet is assigned to a particular execution context (whose boundary is called *firewall*) that controls the access to its objects. Java Card™ platform supports also a secure object sharing mechanism across the firewalls.

The "Application Protocol Data Unit" (APDU) is the logical data packet that is exchanged between the CAD and the Java Card™ Framework: the Framework receives and forwards to the appropriate applet any incoming command APDU sent by the CAD, then the applet processes the command APDU and finally returns a response APDU after having computed the result.

In conclusion, the Java Card™ is an open platform that can be used to develop applications on smartcards without considering the underlying proprietary operating system or hardware. Java Card™ is certainly less expressive than Java, but it's still a high level programming language. One drawback can be represented by the additional bytecode interpretation layer that could slow down the computation, with respect to compiled applications.

For more exhaustive information about Java Card™ see [35, 44].

2.6 Smartcards and fingerprint matching algorithms

To make the biometric verification secure, it is important to store the biometric data in a secure way. To be able to keep the biometric information in a closed environment, it's 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 realizes the description of *Match On Card* (MOC) technology, in which security is significantly improved 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, mostly executed outside in the connected PC. In MOC, candidate template has to be sent every time to the smartcard where the card-owner reference template is stored. The *System On Card* (SOC) technology, in which the card is further enhanced with the fingerprint scanner and the feature extraction function (so achieving the best security), is still far to be easily realized.

While TOC technology is today fully realizable since the card memory is usually wider than a typical template space occupation (a few Kbytes), imple-

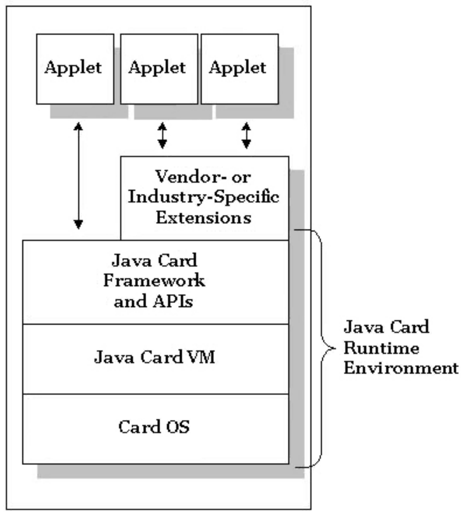

Applet
Applet
Applet
Vendor- or Industry-Specific Extensions
Java Card Framework and APIs
Java Card VM
Card OS
Java Card Runtime Environment

Fig. 5: Java Card™ architecture.

menting a matching algorithm on it (MOC) is a difficult task due to the still limited smartcard computational power, and since this type of computation usually demands some relatively complicated calculations. Moreover, often a smart card realizing the MOC technology is mainly used for resource or building access, and so the verifying procedure has to be relatively quick, avoiding in this way to frustrate the card-owner with a long waiting.

Actually many companies announces their proprietary MOC solutions in the smartcard market: some examples are given by smartcard issuers like Gemplus or Schlumberger, which implement two matching algorithms on their cards, developed respectively by Veridicom and Precise Biometrics. Precise BioMatch™ is an hybrid fingerprint matching algorithm combining both the traditional minutiae and the pattern comparison to derive the final match decision. It can be provided as an integrated function of Java Card™ operating system (native support), or as a normal Java Card™ library to be downloaded on the card; these functions are offered with Java Card Forum™ API methods (see Section 3.3). In terms of accuracy and speed, the matching performances are told to be comparable to those of PC-like systems [46].

Related work. Regarding scientific literature, in [47] is briefly explained a very simple $O(n^2)$ (where n is the number of the minutiae in one single template) matching algorithm that can be implemented inside a smart card. For a given minutia in reference template, it finds all the minutiae in the reference template for which the distance between position coordinates and the difference in orientation angles, are below the predefined thresholds; however it considers only the minutiae of the same type (bifurcation or termination). If more than one minutia in the candidate template could be matched with the same reference minutia, the conflict is resolved by choosing the geometrically nearest. Matched minutiae are deleted from the successive comparisons.

In [48] is reported an algorithm developed for embedded devices, which usually offer greater computational resources than a traditional smartcard. Even this algorithm is based on the minutiae local information, and more precisely is based on the neighbor features like the distance from central minutia and the orientation respect to the central minutia: 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 matching neighbors, in relative neighborhoods, is above a certain predefined value. The final decision regarding the two entire templates is taken by deriving from the number of total minutiae matched in this way.

One specific algorithm for fingerprint matching on the Java Card™ platform, using a feature extraction environment identical to ours, is described in [51]; it uses two distinct algorithms on different feature types (hybrid matcher) like Precise BioMatch™, and at the end the overall score is calculated 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 and then visiting the neighbor minutiae; the matching procedure has been inspired from the point-pattern matching algorithm in [52] 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 and has been implemented exactly as described in [28], by computing the euclidean distance between the fingerprint feature vectors called “Fingercode” (the templates for this specific situation). Performance tests have been executed on a proprietary database and show an ERR point at 0.8%.

3 Our Matching Algorithm

3.1 Features

In our algorithm implementation, the image processing stage used to extract a minutiae set from the fingerprint is based on the NIST Fingerprint Image Software [5], an open source toolkit which includes the MINDTCT minutiae data extractor (written in the C programming language) used to extract the minutiae from a given fingerprint image. From the result of the minutiae detection step, information such as x and y coordinates, local ridge direction, type (found between ridge ending or bifurcation classes) and reliability derived from fingerprint image zone quality, are available for each minutia. Moreover, for every such singularity point is returned a list of neighbor minutiae, that are the nearest minutiae to the considered one. Also an estimation of *ridge count* between central and neighbor minutiae 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 have often used the ridge count to increase the reliability of their analysis [12].

We have used this information extracted by MINDTCT to derive additional features, directly used in our fingerprint matching algorithm; these features are calculated from each minutia in respect to its neighbors, so each neighbor is described by the following four features graphically described in Figure 6, where D is the segment linking central minutia A and its neighbor minutia B, θ_1 and θ_2 are the minutia orientation angles and α is the angle between D and central minutia orientation θ_1:

– the euclidean distance between central minutia and its neighbor minutia (segment D in Figure 6). In the rest of this article we will refer to this feature as *euclideanDistance*.
– the angle between segment D and the central minutia ridge direction (angle α in Figure 6); referred as *distanceRelativeAngle*.
– the difference angle between central minutia and neighbor ridge orientation angle ($\theta_1 - \theta_2$ in Figure 6); referred as *orientationDifferenceAngle*.
– the ridge count between central and neighbor minutia (in the example in Figure 6, segment D intersects only one ridge, so ridge count value is 1); referred as *ridgeCount*.

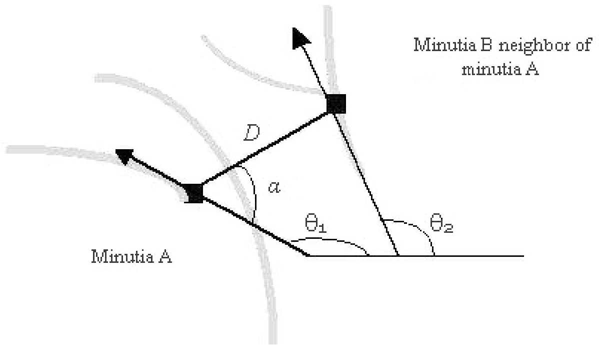


Fig. 6: Features graphical description.

The selection of the neighbors number is very important for the system performance in reliability terms (but even for the matching speed), and so we have chosen to increase the maximum neighbor number to be found, from the default MINDTCT value (5) to the new value of 8. We have also modified the MINDTCT C source code to consider only the neighbors with a minimum reliability threshold: the modified MINDTCT finds for every minutia its eight nearest neighbors respect to the euclidean distance, with a good reliability estimation given by a predefined threshold value. If the number of neighbors for a minutia found in this way is low (i.e. less than 5), then the neighbors are searched again with a lower reliability threshold (the reliability evaluation is found by MINDTCT). We have introduced all these changes to build a good neighborhood with more information, enough to face the possible lack of some minutiae in the template due to a highly disturbed image or a finger misplacement on the scanner.

We have decided not to use the minutiae type information (bifurcation and ridge ending) since sometimes MINDTCT fails to correctly identify them because of disturbed ridge zones.

Given a minutia, the collection of all of its neighbors features describes its neighborhood view, a sort of minutia "panorama" of its surrounding neighbor minutiae.

3.2 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. When two minutiae are "enough" similar, then they are considered as matched; the same algorithm is repeated for each candidate template minutia. The sum of the similarity measures obtained from the matching minutiae pairs is computed during the process, and then the matching algorithm can decide by applying a threshold on this score. Our procedure is based on the minutiae local structures and has tolerance over translation, rotation and small non-linear distortions.

As said before, matching on smartcard environment is bounded by the low computational complexity due to the hardware simplicity (CPU limitations first of all), and thus waiting for a complete minutiae match could lead to a too long waiting time for the user. In our algorithm we solve this problem by stopping the computation as soon as it's possible to assert, with satisfactory confidence, that the considered templates belong to the same fingerprint. To realize 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 similarity measure, or even promptly when only the last examined minutiae pair has a matching value less than a very rigourous threshold; otherwise, if these two conditions are not fulfilled, the algorithm explores all the minutiae pairings space. This relaxation showed a very good security performance in our tests and provided evident speed improvement in the matching decisions regarding positive matches (Section 4). The delay for unsuccessful matches scanning all the minutiae list is not of much interest, because it's clearly more important to gain a high execution speed while verifying the true card-owner identity than quickly rejecting an impostor!

As input, the matching procedure receives both the neighbor features information for the one by one candidate minutia to be matched and the entire reference template. The algorithm scans sequentially the minutiae of the reference template until a good match for the input minutia is found (reference 1 in Fig 7). Both candidate and reference minutiae list are stored accordingly 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, since a minutia with a high reliability in a given template, if not cut away from partial overlapping (Section 2), will have probably a high reliability also in other templates obtained from the same finger. So the stopping conditions can be met before than in a casual disposition of the minutiae in the list. Moreover, it's obviously better to prematurely stop the procedure with few but "good" minutiae than with low quality ones. The chosen matching minutia in the reference template is then marked as "already matched" and it's not considered in the successive match iterations.

To compute the dissimilarity between two minutiae in different templates, the algorithm uses the information about of neighbor features and executes the following four steps (4 in Fig 7):

1. To find the difference in absolute value between corresponding features: $EdDiff = \mid Ed_1 - Ed_2 \mid$, $rcDiff = \mid Rc_1 - Rc_2 \mid$, $draDiff = \mid Dra_1 - Dra_2 \mid$ and $odDiff = \mid Oda_1 - Oda_2 \mid$.
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 neighborhoods ($edDiff$ must not be greater than $edDiffThr$, $rcDiff$ 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 relative weight value: $edWghtDiff = edDiff * edWght$, $rcWghDiff = rcDiff * rcWght$, $odWghtDiff = odDiff * odWght$ and $draWghtDiff = draDiff * draWght$. The different weight values are necessary to attribute more importance to the features that match better, for example the euclidean distance. To obtain each weight value, we have also divided for the respective feature difference bounding box threshold, since we want these differences to be normalized and homogenous.
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 steps, the algorithm finds for the first neighbor (in the casual neighborhood order) 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. 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*. The procedure is repeated exactly for all the other neighbors (excluding the already marked ones, 3 in Fig 7) or until the required minimum number N (i.e. 4) of neighbors is matched. At the end of the two neighborhoods scanning (at the end of the *for*, 2 in Fig 7), if the procedure has found less than N matching neighbor pairs between the two minutiae (6 in Fig 7), then these two minutiae are not considered as matching because their neighborhoods agree on too few evidences to be a reliable matching minutiae pair, even if the *NeighDissimilarity* value is very low. At the same time, this procedure stops immediately as we match the previous N threshold value of neighbors (5 in Fig 7), because we have seen that stopping before the whole neighborhood scan is sufficient to grant a good reliability and, meanwhile, the match time is considerably speeded up.

The *MinDissimilarity* score between the minutiae is finally divided by the number of matched neighbor pairs and then added to the global dissimilarity value between the candidate and reference templates (7 in Fig 7): the *TemplDissimilarity*; the same algorithm is then executed for the next candidate template minutia in reliability order. When all of the input minutiae have been processed, this global *TemplDissimilarity* value on templates is divided by the number of matched minutiae *MinutiaeNMatched*, finding in this way the mean. A comparison between a match threshold and this mean value can consequently be used to decide if the two templates belong to the same fingerprint (if the mean is below the threshold): lower *TemplDissimilarity* expresses more affinity.

That just explained is the full algorithm description, but as said before, the matching procedure will probably end before the complete minutiae list of the candidate template has been processed by the algorithm: if at the end of the minutiae matching routine the dissimilarity value between two matched minutiae is "very good", that is below a tightening threshold *OptValue*), the counter *OptMinNumber* is incremented and as soon as it reaches a predefined constant value corresponding to the threshold *OptNumberThreshold*, the whole matching procedure can be stopped with a positive result (8 in Fig 7). The algorithm can be positively stopped also as soon as it finds only one minutiae pair with an "exceptionally good" *MinDissimilarity* value below the *VeryOptValue* threshold (9 in Fig 7), which is intended to be much stricter than the previous *OptValue*.

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

3.3 Algorithm implementation

The fingerprint matching algorithm described in Section 3.2 has been fully developed, debugged and tested on a PC with a Pentium4 CPU (2.66Ghz), a RAM of 512Mb and using JDK1.4.3. Then the same algorithm has been ported on Java Card™ using the Java Card™ 2.1.2 API and finally deployed on a Cyberflex Access 32Kb Java Card™ with the Cyberflex Access SDK (version 4.3). Chosen smartcard has 32Kbyte of EEPROM, about 1Kbyte of RAM memory distributed between the transaction mirror, stack and transient space, 8 bit CPU micro-controller at about 8Mhz clock frequency, supports cryptographic operations and the garbage collector is not present; the transmission protocol used with the smartcard reader is the *T=0* at 9600 bit/sec.

The algorithm has been developed by implementing the Java Card™ Biometric API [42] realized by Java Card Forum™ [45] (JCF): this application programming interface (approved by the *Java Card Forum Biometric Task* and

```
{MINUTIAE MATCHING PROCEDURE}

   - Input:* one candidate template minutia m1;
           * minutiae list of the reference template;

1  For each minutia m2 in reference template not yet matched{
2    For each neighbor n2 of minutia m2 {
       - MinDiff = upperLimit;
       - ChosenNbr= null;
3      For each not already matched neighbor n1 of m1 {
4        - Executes the four steps between the n1-n2
           corresponding features (directly processes next n1
           if the bounding box rejects the controls);
         If (NeighDissimilarity < MinDiff) {
           - MinDiff = NeighDissimilarity;
           - ChosenNbr = n1; }
       }
       If (ChosenNbr != null) {
         - ChosenNbr is marked as "matched";
         - MinDissimilarity += MinDiff;
         - number of matched neighbors NM= NM + 1; }
5      If (NM > N)
         - m1 and m2 are "matched": break from this For;
     }
6    If (NM < N)
       - Continue with the next minutia m2
     else {
7      - m1 and m2 are "matched": TemplDissimilarity+=
         (MinDissimilarity \ NM);
       - break from this For;
     }
   }
   - m1_m2_MatchCost = MinDissimilarity \ NM;
   If (m1 and m2 are "matched") {
     - MinutiaeNMatched++;
     - Mark reference minutia m2 as "matched";
8    If (m1_m2_MatchCost < VeryOptValue)
       - STOP: the match is accepted;
     If (m1_m2_MatchCost < OptValue)
       - OptMinNumber++;
9    If (OptMinNumber == OptNumberThreshold)
       - STOP: the match is accepted;
   }
   - Process another minutia m1 if no stopping condition
     has occurred or if m1 and m2 are not "matched";
```

Fig. 7: Matching core function: text reference is in the first column.

the *NIST Biometric Consortium Working Group*) ensures the interoperability of many biometric technologies with Java Card™ and allow multiple independent applications on a card to access the biometric functionalities (like identity verification); this is ideal to secure digital signature, storing and updating account information, personal data (health information) and even monetary value. Clearly, our application manages even the enrollment and match requests coming from the external PC applications through several CAD sessions. The other AFIS phases like the image manipulation and feature extraction are executed on the PC.

We have chosen to adopt the Java Card™ technology to implement our matching algorithm since this platform offers a good security and a high-level programming language, close to classic Java™. Benefits and drawbacks of this platform are identical to those of its "mother technology": high portability and programming/developing quickness, but also a reduced execution speed due to the additional bytecode interpretation layer.

In this case the difficulty consists not only in improving the verification time by simplifying the algorithm steps, but even adapting the code to this very restricted environment: for example, since the garbage collector is not present in our smartcard, all the data structure objects have to be instantiated at the beginning of the applet life-cycle and have to be reused every time, because the on-card space of the dereferenced objects is lost forever and new repeated allocations would quickly consume the entire EEPROM memory: the objects lifetime coincides with that of the card itself and they can't be explicitly deleted. Moreover, when possible it's always better to move the data memorization from objects (stored in EEPROM) to transient space in RAM, because it's about 1000 times faster to write in RAM than in EEPROM memory. Furthermore it has to be remembered that the card processor supports only fixed point arithmetic and the Java™ math API are totally absent, so the possible operations are very simple.

Aside this global simplification, the verification reliability performance has to be kept as high as possible.

For these reasons, due to the Java Card™ environment constraints we have 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 in our implementation. We have limited also the maximum number of the candidate template minutiae to be sent during the matching phase: the decision is taken anyway after the 20th one is received. For the same reason of speed improvement and memory occupation contraction, the maximum number of neighbors per minutia has been limited to 8.

The neighbor feature values must be sampled to be then stored in the low capacity Java Card™ primitive data types like *byte*, preferring its use respect to other types like *short*, that has a size of 16 bits instead of 8. Nearly all the feature values have been sampled to entirely fit in the *byte* type, which has a maximum value limit of 127. Features *euclideanDistance*, *distanceRelativeAngle* and *orientationDifferenceAngle* have been normalized in order to fit in the interval [0, 127], while *ridgeCount* never exceeds the *byte* maximum value. This sampling also prevents the feature differences sums (stored into a *short*) to exceed the capacity of the Java Card™ data types.

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

We have set the *VeryOptValue* value to the half of the *OptValue* threshold, *OptNumberThreshold* is set to 2 and N to 4; *MinutiaeNMin* threshold is instead set to 3. These and other parameters like final threshold MT can be configured basing on the desired speed and security performance. The configuration above has provided the best security results for FAR100 and FAR1000 and also the best match speed time, as described in Section 4.3.

The EEPROM allocation space to be reserved on the card for our MOC algorithm is about 6Kbyte, while additional EEPROM memory has to be used to store the reference template minutiae, depending from their predefined maximum number: for a maximum of 20, the total space requested is about 10Kbyte, about one third of the whole EEPROM memory. Every minutia is stored as a Java Card™ object.

4 Performances

4.1 Fingerprint test database

Usually the performances of fingerprint matching algorithm are tested with a proprietary fingerprint database and most of times it's impossible to compare the systems results with uniform operating conditions. An algorithm can achieve good performance for its chosen image acquisition system, and so certifying the overall (acquisition plus matching) system performance, but it can also be useful to test the functioning over a common reference. We have decided to use the *Finger Verification Competition 2002* [38] edition (FVC2002) fingerprint databases, since, as we know, is the only public benchmark (aside FVC2000 [39] and FVC2004 [40] editions of the same contest[3]) allowing the developers to unambiguously compare their algorithms.

Each of the four databases is 110 fingers wide and 8 impressions per finger deep (880 fingerprints in all); the benchmark is then 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.

The four database images have been collected respectively using the optical sensor "TouchView II" by Identix, optical sensor "FX2000" by Biometrika, capacitive sensor "100 SC" by Precise Biometrics and the *Synthetic Fingerprint Generator* software [49] (SFINGE).

[3] In our tests we have used the databases from FVC2002, since the use of FCV2004 databases has been granted to us only a few weeks ago.

Regarding fingerprints, the only other large public domain databases are *National Institute of Standards and Technology* (NIST) collections, but they are not totally appropriate for automated systems working on live-scan images: for example the NISTDB 4 [41] contains images scanned from enrolled inked impressions.

Moreover, we have personally collected a smaller database using the FX2000 scanner to directly acquire some image 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. Next, we will refer to this as the "Internal Database".

In addition to the image databases already discussed, we have analyzed our algorithm in respect with that described in [51], using even the proprietary database provided by the authors. This very good quality image collection is made up of from 9 to 11 different repetitions of 55 fingers, for a total of about 550 samples; these images have been collected using the same FX2000 optical scanner model by Biometrika [53]. After, we will refer to this as the "Hybrid Database".

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 happenings are respectively defined as the *False Acceptance Rate* (FAR) and *False Rejection Rate* (FRR).

A further performance description can be given by the *Equal-Error Rate* (EER), which denotes the error rate for which FAR and FRR are identical at some operating point t (the matching threshold). Although it's an important indicator, rarely the biometric verification systems are used with a threshold corresponding to the EER, because is generally preferred to achieve improved security with a more stringent operating point, therefore reducing the FAR and consequently increasing the FRR. The trade-off curve associated with fingerprint authentication systems is known as the *Receiver Operating Characteristic* (ROC) curve. The ROC curve plots the FAR (on x axis) against the FRR (on y axis) for the same match threshold value, and can be generated by obtaining FAR and FRR values under many different operating points on a given system.

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.

Even the maximum template size has its significant importance, because the smartcard EEPROM memory is usually limited to 16-32Kb and is used also to store the matching applet code and other applications.

Other interesting performance indicators can be derived by increasing or decreasing the matching threshold (or generally changing some execution parameters), to show the algorithm's behaviour for application that need more or less security: for example the FAR100 (the lowest achievable FRR for a FAR $\leq$ 1%), FAR1000 (the lowest FRR for FAR $\leq$ 0.1%), ZeroFAR (the lowest FRR for FAR = 0%) and finally ZeroFRR (the lowest FAR for FRR = 0%). For example, configuring the operating point to work under the FRR1000 conditions characterizes applications with rigourous security requisites (like airport buildings access), while for forensic purposes it's better to operate under ZeroFRR conditions to include all the possible crime suspects in the identity research.

Other measurements that can be observed are REJ_{ENROLL} or REJ_{MATCH}, which respectively quantify the number of rejected fingerprints during the enrollment and match.

All of these performance result indicators can be substantially altered in our algorithm by modifying the algorithm parameters such as the matching threshold (FAR in spite of FRR) or the maximum number of minutiae and neighbors in the template (security in spite of speed); this can be useful to adapt the algorithm for the chosen application environment. It can be clearly deduced that a simple performance results report is always relative to a particular parameters configuration and so it cannot describe completely the algorithm behaviour, but only one of its aspects.

4.3 Results on our algorithm

The test distribution between positive and negative matches can greatly influence the declared performances. For this reason we have decided to run the same tests as the FVC2002 competition (see [38] 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 the same exactly fingerprint images as in the international competition.

We have tested our algorithm using the configuration exposed in Section 3.3 and here we reassume some of the results obtained from the FVC2002 database collected with the FX2000 optical scanner, which has showed some of the best global performances among the others:

– *EER* 8.5%.
– *FAR100* 10.6%.
– *FAR1000* 12.5%.

In Figure 8 and Figure 9 are showed respectively the ROC curve and the FAR-FRR curves for that configuration, with EER point at their intersection. The strange shape of the graph lines (respect to classic ones) comes from the decision to stop the algorithm as soon as possible, even with few but "very good" minutiae pairs: this decision is independent from the final matching score and so setting the match threshold to a low or a high value doesn't correspondingly lead to have a FAR or a FRR of 0% or 100%; in particular the zones in the graph near FAR or FRR at 100% or 0% cannot be represented. Consequently some indicators like the ZeroFRR and ZeroFAR cannot be correctly measured, because the FAR and the FRR curves don't intersect that areas. However, all the thresholds and constants present in the algorithm description (changing those proposed in Section 3.2) can be simply adapted to fulfill other different goals: for example, another parameters configuration adopted can improve EER at about 7%, while another more has reached even ZeroFAR zone at 15.6%. For our main parameters configuration, we were essentially interested in giving the best FAR1000 performance.

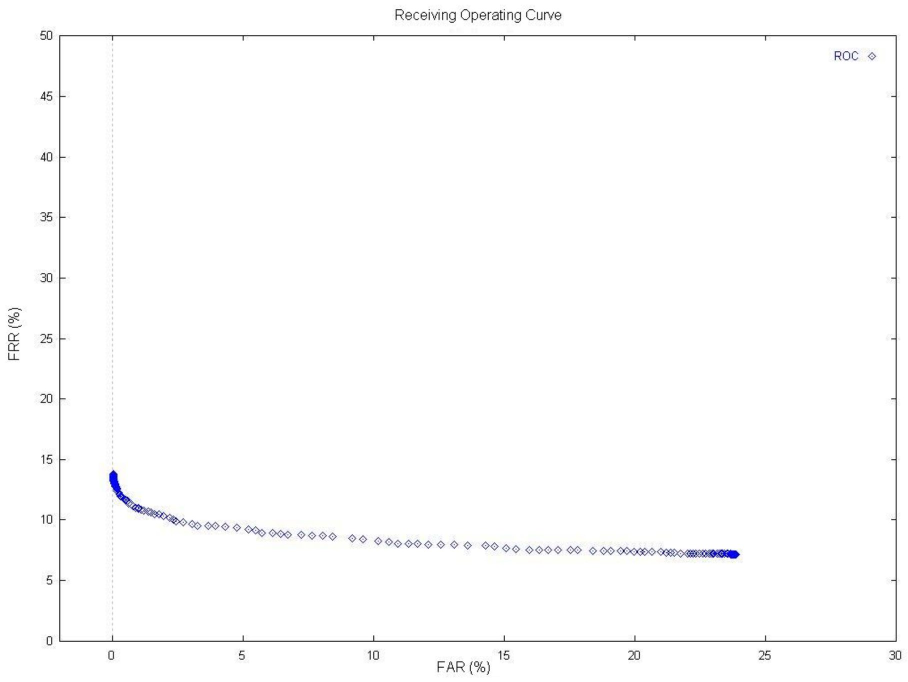


Fig. 8: ROC curve for FVC2002 FX2000 database.

These security performance results have been obtained running the tests on a PC to reduce the experiment time, thanks to the fact that the algorithm is exactly the same as the one implemented directly on the smartcard (where therefore the same results are completely achievable). The performance showed can compete only with those of the last classified algorithms of FVC2002 (see [38]), but they are realized without the smartcard environment restrictions. In FVC2002 the performance have been calculated on a PentiumIII at 933Mhz, which is about 120 times faster than our smartcard CPU, without considering other architecture gaps like the system bus, memory speed and the external transfer rate at only 9600 baud. Confronting our architecture with the FVC2002 test environment, we can see that our algorithm is faster than all the FVC2002 algorithms.

Additionally our results can be greatly improved using a good enrollment image, as exposed later in this section.

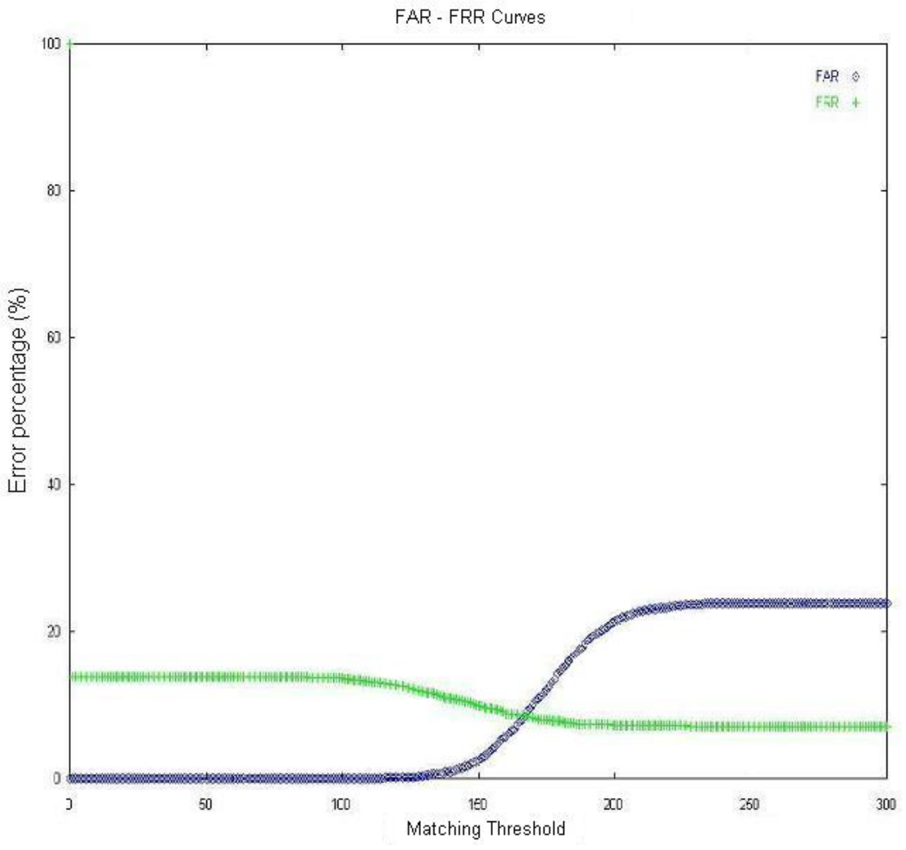


Fig. 9: FAR and FRR curves for FVC2002 FX2000 database.
EER is at the intersection point.

The execution time of matching phase has been tested directly for the smartcard realization: we have noticed that a minimum time of about 1 seconds is needed for the entire procedure and this result can be achieved frequently using a good enrollment fingerprint image, thanks to the algorithm option to exit deciding only from few minutiae pairs. Maximum time is instead about 45 seconds, 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 to quickly stop without exploring all the minutiae pairings. Nearly all of the correct positive matches are resolved in 1-8 seconds, depending from the quality of the candidate template, and obviously of the corresponding fingerprint image. In Figure 10 we report, for the FVC2002 database and the “Hybrid Database”, the number of the minutiae in the candidate template needed to be sent to the card to stop the match and obtaining a final match decision; the graph bars show this distribution between correct matches performed to test FRR. Fewer minutiae means less time employed to match two fingerprints and we can observe that a number between 1 and 4-5 minutiae corresponds to an on card matching time of about 1-8 seconds. The results obtained over the “Hybrid Database” are quite better than the FVC2002 FX2000 database, since its image quality is evidently higher.

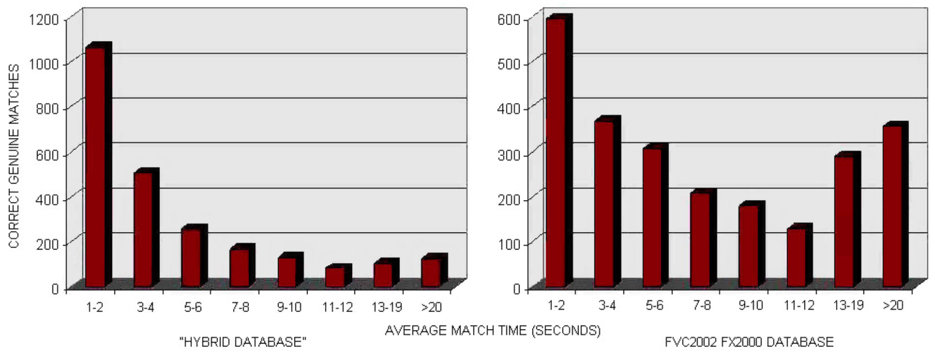


Fig. 10: Number of minutiae needed to stop the match,
with respect to the tests performed to measure FRR (tests between corresponding fingerprint images).

After that, we have tested our algorithm with the same configuration and using all the images (110 instead of 100) in the FVC2002 FX2000 database, for a total of 9,075 matches, distributed in 5,995 to test FAR and 3,080 to test FRR; the performances on 100 images at the beginning of this section have been slightly improved, showing a good stability of the algorithm in the results: reached EER is at 8.2%, FAR100 at 10.4% and FAR1000 at 12.2%.

Moreover, we have additionally tested the algorithm using the first FVC2002 fingerprint database, the one scanned with TouchView II optical sensor. Using the same exactly configuration optimized for the second database and the first 100 images (7,750 match tests), we have achieved similar performances: EER at 8.5% , FAR100 at 10.6% and FAR1000 at 12.3% (this last result is even better respect to the previous database). In this way we have proved that these good results can be carry out from at least two different fingerprint scanners, more attesting the stability of our procedure.

We have also performed some other tests using our “Internal Database”: we have accomplished all the possible tests between different fingers (for a total 51,040 matches to test FAR) and between the same fingers (for a total 1,120 matches to test FRR). The results obtained are between half and one third respect those achieved on the FVC2002 database:

- *EER* 3.8%.
- *FAR100* 4.3%.
- *FAR1000* 4.5%.

One very important aspect to be highlighted is that using the two stopping conditions exposed in Section 3.2 has showed excellent security results: respect to all the used databases, only less than 4-5% of total false matches accepted have been introduced by relaxing the problem and stopping the execution before the complete minutiae lists scan; this result has been obtained by setting a high threshold value to reach at least 100 false acceptance errors.

A comparison with related work. We have also compared our work with the algorithm in [51], developed to be executed in a similar Java Card™ environment. To obtain a significant comparison, we have tested our algorithm with the "Hybrid database" granted to us by the authors. Since even the "Hybrid database" has been collected with the FX2000 scanner, we have properly used the same configuration optimized for the FVC2002 FX2000 database exposed in Section 3.3. With this configuration we have achieved the following results:

- *EER* 0.4%.
- *FAR100* 0.4%.
- *FAR1000* 0.5%
- *ZeroFAR* 0.5%.

The tests have been executed using the FVC2002 guidelines for the match distribution between FAR and FRR tests; with this working conditions we have halved the EER percentage of 0.8% achieved for the Java Card™ algorithm exposed in [51] (see Table 1), even if this algorithm uses two different matching module and at last merges their results.

Algorithm	EER on "Hybrid Database"
Hybrid matching algorithm in [51]	0.8%
Our algorithm	0.4%

Table 1: EER comparison between our algorithm and the algorithm in [51].

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

Moreover, we have seen that using a good quality enrollment image improves considerably the overall security performances: FAR1000 value can be nearly reduced to one third to about 4-5%, with respect to FVC2002 results and eliminating for example the partial overlapping problem. This functioning hypothesis is not pervasive at all and it's easily applicable, since the enrollment phase is accomplished only one time at the release/inizialization of the smartcard, and can be also controlled by a quality checker software module or possibly even by a human operator. Instead we place no restrictions on the quality of the image acquired during the verification phase. So, a MOC application for our matching algorithm can fully benefit from this hypothesis in the terms of speed and reliability performances.

In Table 2 we show all the reliability results previously presented, where "-" means that the result is not achievable with the used particular parameters configuration.

Table 2: Overall performance results.

Fingerprint database	EER	FAR100	FAR1000	ZeroFAR
FVC2002 FX2000 db (100 img)	8.5%	10.6%	12.5%	-
FVC2002 FX2000 db (110 img)	8.2%	10.4%	12.2%	-
FVC2002 TouchView II db (100 img)	8.5%	10.6%	12.3%	-
"Internal database"	3.8%	4.3%	4.5%	-
"Hybrid Database"	0.4%	0.4%	0.5%	0.5%

5 Conclusion

In this paper we have proposed a new fingerprint minutiae matching algorithm, thought and developed to face the Java Card™ platform restrictions. Our algorithm, tolerant to typical match problems like the rotation, translation and ridge deformation, achieves a very good speed performance for the smartcard environment (1-8 seconds for the most of positive match tests). The high reliability, as determined from our analysis obtained from several different databases, can be further greatly improved using a good enrollment image: a FAR less than 0.1% and a FRR of about 4-5% (the FAR1000 score) make the algorithm implementation feasible in the live-scan applications for the identity verification, like a MOC system. With very good quality enrollment and reference images, like those in [51] database, the performances can reach a ZeroFAR result of about 0.5%.

Our algorithm shows an asymmetric behavior respect to the verification execution time: the procedure is stopped as soon as the templates are considered to belong to the same finger, and so the algorithm stops before in correct FRR tests and later in the correct FAR ones; in the case of errors, this behaviour is exactly the opposite.

The MOC realization attests moreover a better security, since the match decision is taken inside the card: the template never leaves the card and therefore it cannot be intercepted, blocking in this way possible "man-in-the-middle" type attacks.

We have chosen the Java Card™ platform to make the implementation fully portable towards all the Java Card™ compliant smartcards; moreover this platform provides a good security and a high level programming language, completely independent from the particular card operating system.

A MOC architecture with our matching algorithm has been already successfully introduced in a digital signing tool developed at the *Institute for Informatics and Telematics* (National Research Council of Pisa): 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 could involve the image treatment to better define the fingerprint ridges, improving in this way the algorithm reliability; moreover we could add a core point structure to introduce some global information in the template representation and using it in the algorithm. We could observe even the inclusion of a different matching algorithm, like the one described in [28], to perform a multimodal verification.

Acknowledgements

We would like to thank the organizers of Fingerprint Verification Competition (FVC) [38–40] 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 even 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 Java Card™ algorithm in [51].

References

[1] A. K. Jain, S. Prabhakar, and S. Pankanti, *On The Similarity of Identical Twin Fingerprints*, Pattern Recognition, Vol. 35, No. 11, pp. 2653-2663, 2002.

[2] H. C. Lee and R. E. Gaensslen (editors), *Advances in Fingerprint Technology*, Elsevier, New York, 1991.

[3] S. Pankanti, S. Prabhakar and A.K. Jain, *On the Individuality of Fingerprints*, Proc. IEEE CVPR, pp. 805812 , Hawaii, Dec 2001.

[4] *Handbook of Fingerprint Recognition* D. Maltoni, D. Maio, A.K. Jain and S. Prabhakar, Springer, 2003, ISBN 0-387-95431-7.

[5] *User's Guide to NIST Fingerprint Image Software (NFIS)*, NISTIR 6813, National Insitute of Standards and Technology.

[6] *Federal Bureau of Investigation: The science of fingerprints: Classification and uses*, US Government Printing Office, Washington DC (1984).

[7] L. Hong, A.K. Jain, R. Bolle and S. Pankanti, *Identity Authentication Using Fingerprints*, Proc. of First Int'l Conf. On Audio and Video-Based Biometric Person Authentication, Switzerland, pp. 103-110, March 1997.

[8] E. Newham, *The Biometric Report*, SJB Services, London 1995.

[9] Cummins, H., and M. Midlo. 1961, *Finger prints, palms and soles: an introduction to dermatoglyphics*, Dover Publications Inc., New York.

[10] *The Henry Classification System Copyright*, International Biometric Group, 2003.

[11] C. L. Wilson, G. T. Candela, and C.I. Watson, *Neural Network Fingerprint Classification*, J. Artificial Neural Networks, Vol. 1, No. 2, pp. 203-228, 1993.

[12] E.R. Henry, *Classification and Uses of Fingerprints*, Routledge, London (1900).

[13] *Daubert v. Merrill Dow Pharmaceuticals*, 509 US 579 (1993).

[14] *The FBI Fingerprint Identification Automation Program: Issues and Options–Background Paper*, OTA-BP-TCT-84 (Washington, DC: U.S. Government Printing Office, November 1991).

[15] J.H.Wegstein, *An Automated Fingerprint Identification System*, U.S. Government Publication, Washington, 1982.

[17] L. Hong, Y. Wan and A.K. Jain, *Fingerprint Image Enhancement: Algorithms and Performance Evaluation*, IEEE Transactions on PAMI ,Vol. 20, No. 8, pp.777-789, August 1998.

[19] L. Hong, A.K. Jain, S. Pankanti and R. Bolle, *Fingerprint Enhancement*, Proc. IEEE Workshop on Applications of Computer Vision, Sarasota, Fl, pg. 202-207, Dec. 1996.

[20] X.D. Jiang and W.Y. Yau, *Fingerprint Minutiae Matching Based on the Local And Global Structures*, Proc. 15th Int'l Conf. Pattern Recognition, vol. 2, pp. 1042-1045, 2000.

[21] N. K. Ratha, R. M. Bolle, V. D. Pandit, V. Vaish, *Robust Fingerprint Authentication Using Local Structural Similarity*, IEEE 2000.

[22] Andrew K. Hrechak , James A. McHugh, *Automated fingerprint recognition using structural matching*, Pattern Recognition, v.23 n.8, p.893-904, 1990.

[23] Fan, Kuo Chin, Cheng Wen Liu, and Yuan Kai Wang, *A randomized approach with geometric constraints to fingerprint verification*, Pattern Recognition, volume 33, pp. 1793-1803 (2000).

[24] Willis, A.J., Myers, L., 2001, *A cost-effective fingerprint recognition systems for use with low-quality prints and damaged fingerprints*, Pattern Recognition 34 (2), 255-270.

[25] Zs. Miklos Kovacs-Vajna, *A Fingerprint Verification System Based on Triangular Matching and Dynamic Time Warping*, IEEE Trans. on PAMI, Vol.22, No.11, pp. 999-1010, 2000.

[26] A.K. Jain, L. Hong and R. Bolle, *On-line Fingerprint Verification*, IEEE Transactions on PAMI, Vol. 19, No. 4, pp. 302-314, 1997.

[27] A. K. Jain, A. Ross, and S. Prabhakar, *Fingerprint Matching Using Minutiae and Texture Features*, Proc. I nternational Conference on Image Processing (ICIP), pp. 282-285, Greece, October 7-10, 2001.

[28] A. K. Jain, S. Prabhakar, L. Hong and S. Pankanti, *Filterbank-based Fingerprint Matching*, IEEE Transactions on Image Processing, Vol. 9, No.5, pp. 846-859, May 2000.

[29] J.D. Stosz and L.A. Alyea, *Automated system for fingerprint authentication using pores and ridge structure*, Proceedings of SPIE, Automatic Systems for the Identification and Inspection of Humans, San Diego, vol. 2277, pp. 210-223, 1994.

[30] Kaymaz, E. & Mitra, S. (1992), *Analysis and matching of degraded and noisy fingerprints*, Applications of Digital Image Processing XV, Vol. 1771 of SPIE, pp. 498-508.

[31] M. Takeda, S. Uchida, K. Hiramatsu, and T. Matsunami, *Finger image identification method for personal verification*, Proceedings of the 10th International Conference on Pattern Recognition, June 1990, Vol 1, International Association for Pattern Recognition, pp 761-766.

[32] Hatano, T., Adachi, T., Shigematsu, S., Morimura, H., Onishi, S., Okazaki, Y., Kyuragi, H., *A fingerprint verification algorithm using the differential matching rate*, ICPR02, III volume: pp. 799-802, 2002.

[33] Bazen, A.M., G.T.B. Verwaaijen, S.H. Gerez, L.P.J. Veelenturf and B.J. van der Zwaag, *A Correlation-Based Fingerprint Verification System*, ProRISC 2000 Workshop on Circuits, Systems and Signal Processing, Veldhoven, The Netherlands, (November 2000).

[34] N. Ratha, S. Chen, K. Karu and A.K. Jain, *A Real-time Matching System for Large Fingerprint Databases*, IEEE Trans. PAMI, Vol. 18, No 8, pp. 799-813, 1996.

[35] *Technology for Smart Cards: Architecture and Programmer's Guide*, Zhiqun Chen, Addison-Wesley Pub Co, 1st edition (2000), ISBN: 0201703297.

[36] *Programmazione delle Smart Card*, Ugo Chirico, Gruppo Editoriale Infomedia, (2003), ISBN: 8881500140.

[37] *An Introduction to Java Card™ Technology*, Part 1, Part 2, Part 3, by C. Enrique Ortiz, (2003), accessed from http://java.sun.com/products/javacard/ .

[38] D. Maio, D. Maltoni, R. Cappelli, J. L. Wayman and A. K. Jain, *FVC2002: Second Fingerprint Verification Competition*, Proc. of International Conference on Pattern Recognition, pp. 811-814, Quebec City, August 11-15, 2002.

[39] D. Maio, D. Maltoni, R. Cappelli, J. L. Wayman, and A. K. Jain, *FVC2000: Fingerprint Verification Competition*, Proc. 15th International Conference Pattern Recognition, Barcelona, September 3-8, 2000.

[40] D. Maio, D. Maltoni, R. Cappelli, J. L. Wayman and A. K. Jain, *FVC2004: Third Fingerprint Verification Competition*, To appear in Proceedings of International Conference on Biometric Authentication (ICBA), Hong Kong, July 2004.

[41] C.I. Watson and C.L. Wilson, *NIST Special Database 4*, Fingerprint Database, National Institute of Standards and Technology, March 15, 1992.

[42] *Java Card™ Biometric API White Paper (Working Document)*, Version 1.1, NIST/Biometric Consortium Interoperability, Assurance, and Performance Working Group 7 August 2002.

[43] A. K. Jain, S. Pankanti, S. Prabhakar, L. Hong, A. Ross, and J. L. Wayman, *Biometrics: A Grand Challenge*, To appear in Proc. International Conference on Pattern Recognition (ICPR) , Cambridge, UK, Aug. 2004.

[44] Java Card™ official web site: http://java.sun.com/products/javacard/ .

[45] Java Card Forum™ official web site: http://www.javacardforum.org/

[46] Precise™ Biometrics official web site: http://www.precisebiometrics.com .

[47] Y.S.Moon, H.C. Ho, K.L. Ng, *A Secure Card System with Biometric Capability*, 1999 IEEE Canadian Conference on Electrical and Computer Engineering, Volume 1, pp 261-266, 1999.

[48] S. Yang and I. Verbauwhede, *A secure fingerprint matching technique*, ACM Workshop on Biometrics: Methods and Applications, pp. 89-94, November 2003.

[49] SFINGE web site (Biometric Systems Lab of Cesena, Italy): http://bias.csr.unibo.it/research/biolab/bio_tree.html.

[50] Cormen T.H., Leiserson C.E. and Rivest R.L., *Introduction to Algorithms*, McGraw-Hill, New York, 1990.

[51] Tommaso Cucinotta, Riccardo Brigo, Marco Di Natale, *Hybrid Fingerprint Matching on Programmable Smart-Cards*, TrustBus 2004.

[52] P. B. van Wamelen, Z. Li, and S. S. Iyengar, *A fast algorithm for the point pattern matching problem*, November 2000.

[53] Biometrika s.r.l. web site: http://www.biometrika.it

[54] S. Bistarelli, G. Boffi, F. Rossi, *Computer Algebra for Fingerprint Matching*, Proc. International Workshop on Computer Algebra Systems and Their Applications, CASA'2003, Saint Petersburg, June 2-4 2003.