Smart Cards:
Remote Authentication using Biometrics

Filipe Marques[1], Ricardo Chaves[1], Carlos Lima[2]
filipe.marques@ist.utl.pt, *ricardo.chaves@inesc-id.pt*, *clima@pt.zetes.com*

[1]Instituto Superior Técnico/INESC-ID, [2]Zetes Burótica

Abstract. This paper presents an architecture for a secure and reliable remote authentication system, based on local biometric signature validation. Biometric systems are a more secure and reliable authentication mechanism than those that use PIN and passwords. This authentication is based on something that is unique to each person and cannot be lost or borrowed, ensuring the presence of the specific person during the authentication process. Additionally, biometric systems are a more elegant approach to authentication, eliminating the need for the user to memorize any additional information, also protecting him against the theft of his secret. Despite the advantages of biometric systems, they are not used in remote authentication applications, given that it is not viable to send the biometric template via remote connections. The solution herein proposed, uses the combination of Java Cards with a biometric API, for the secure local biometric authentication, with the public keys stored in the card, for the reliable remote authentication. This work shows that remote authentication using biometrics is possible in a reliable and trustworthy manner, using the already existing technology.

Keywords: Smart Cards, Java Cards, Biometrics, Fingerprints, Match on Card, Remote Authentication, Digital Signature.

1 Introduction

The fast growth of networks and remote services available every day, triggers the spotlight over public networks such as the Internet. However, with this emerges the need for security mechanisms to protect the users from the current informatics dangers. Only with efficient and convenient security mechanisms, it is possible to provide the necessary trust, for the sustainable development of this emerging new virtual world.

Network security politics aspire to prevent the unauthorized discloser, destruction, alteration, or copy of private digital information. Therefore, it is indispensable, among other measures, to ensure secure user authentication in remote scenarios. Password authentication schemes are the best-known and still the most used mechanism in contemporary computer systems [1]. In this authentication scheme, the user has to present the correct secret for a specific identity, which is then used to generate a cryptogram response, as in the EKE(Encryption Key Exchange) protocol.

The Password based authentication systems cannot be considered the most reliable and practical remote authentication option. The PIN base authentication systems are not the safest, since the PIN can be susceptible to dictionary attacks, or even be guest or observed by other people. Besides that, this systems implies the use of different passwords for each user Internet account, and the user has memorize all its passwords. Given that, new schemes for remote authentication have emerged in recent times. One solution is to perform the user authentication locally on a trusted device, using the PIN or Password approach, and perform the remote authentication using asymmetric encryption.

Smart cards are a simple and effective way to obtain this trusted devices [1]. Due to their design and architecture, they can provide identification, authentication, secure data storage, and application processing [10]. These trusted devices can securely store crucial information such as user identification and private keys, and provide the necessary computation to perform security algorithms as digital signing and cipher.

If a remote server trusts the Smart Card, it means that a remote authentication process can be performed by the card it self. In fact, the authentication is based on a challenge response, in which the response corresponds to the digital signature of the challenge, signed with the user's private key. Not even the user has access to this private key, ensuring that only the device knows it. The user authentication is performed locally in the card, via PIN, and the remote authentication is performed by the Smart card. The mechanism is based on something that users knows and possesses, his PIN and the Smart Card, respectively. This solution eliminates the need for user to have different Passwords in different systems.

One of the real applications using this remote authentication approach are the national citizen ID cards. These cards are responsible for the cryptographic computations required for the identification and authentication of a citizen. An example of this solution is the Portuguese citizen card, where a Java Card stores the user information, fingerprint templates, PIN, private keys and public certificates signed by the country CA. The applet installed to perform remote authentications follows the IAS (Identification, Authentication, and Signature) defined standard for European citizen cards.

However, in these approaches the local user authentication is based only on his PIN, forcing the user to memorize it. This can be a problem mainly for elderly people having memory problems. Moreover, most of the available mechanisms for the user to insert his value are not safe, allowing the theft of his secrets with simple techniques such as shoulder surfing. Therefore, the passwords are unable to provide non-repudiation, since they can be borrowed, stolen or guessed.

The biometric authentication is a good option to replace the traditional local PIN user authentication, since it has good advantages on commodity and security. Biometrics, unlike PIN or passwords, are specific, unique and nontransferable for each individual, removing the need for the user to memorize any additional information and improving the security of the authentication. Thus, biometric systems can be used to assure that the authentication data cannot be presented by another person. [11]. However, despite being a very good solution for user authentication, biometrics cannot be used in remote scenarios, otherwise the biometric template would have to be sent over the network.

This paper presents a solution to this problem, in order to allow the use of biometrics in remote authentication systems. Several Java Cards possess the capability to manage biometric templates and perform MOC (Match-On-Card) operations. The solution herein presented consists in replacing the local password authentication approach by a local biometric authentication, thereby creating authentication systems more secure, reliable, and easy to use. In this solution, an applet implementing the IAS standard is responsible for the remote authentication of the user, with the user's private key only released after a successful user biometric authentication in the MOC applet.

The solution herein presented shows that it is possible to use biometric authentication in remote scenarios, taking advantages of their potential as a more secure and easy to use authentication mechanism. The implementation of this concept can be deployed using existing technologies, thereby ensuring low development costs, and a high compatibility with existing remote authentication systems. This solution is therefore design to be considered as one reliable possibility to integrate the future authentication mechanisms, in remote scenarios.

Document Roadmap

This paper is organized as follows: Section 2 addresses the current state of the art on Java Cards. Section 3 presents an overview on biometrics as a reliable solution for people authentication, and illustrates their characteristics, with a particular emphasis on fingerprints. Section 4 describes the proposed solution, elaborating on the implementation of the proposed solution and its usage. An security analysis of the presented solution is performed in Section 5. Section 6 concludes this presentation with some final remarks.

2 Java Card Technology

SC (Smart Cards) are small, portable and secure devices very useful in people's lives, enabling services as ATM, telecommunications and ID. As defined in [8], a Smart Card is an electronic device able to participate in secure automated electronic transactions. SC cannot be easily forged or copied, and are typically tamper-resistant devices, in order to support attacks made directly to the chip.

The development and functionality of Smart Cards are strongly driven by international standards, being the more important ones the ISO/IEC 7816 for contact Smart Cards, and the ISO/IEC 14443 for contactless Smart Cards. They are typically composed by CPU, RAM, ROM, and EEPROM. The ROM is used to store the operating system, while some operating system components and the running applications, with their persistent associated data, are stored in the EEPROM. The RAM is a working memory used to hold data during operations.

Identically to computer operating systems, Smart Card operating systems are responsible for providing the control of the communications between the card and the outside world, ensuring the proper and effective execution of the received commands, managing the file system, managing and executing the program code of the applications previously installed, and handling all encryption operations [10]. These operating systems have been developed to run a simple application, until the creation of the current ones, which support multiple applications and the management of applet installation after the card has been issued [12].

Currently, Java Cards are one of the most supported and used Smart Card platforms in the world, when performance and security are the main requirements. The Java Card is a Smart Card platform to run apples written in a constrained Java programming language. Java Cards provide the separation between the Smart Card system and the applications, by implementing a Runtime Environment that supports the Smart Card memory, communication, security, and application execution model [2]. The next sections detail the architecture and the functionality of the Java Cards.

2.1 Runtime Environment

The JCRE (Java Card Runtime Environment) runs on top of the Smart Card hardware and native system. This module is intended to abstract the operating systems of the card, by implementing a common and useful API layer. The native methods abstraction enables the interoperability between various Smart Cards devices and systems. The JCRE can be described as an on-card system module, divided in tree main layers. The first and lower one is composed by the JCVM(Java Card Virtual Machine) and native system methods. The second corresponds to the system classes and are composed by applet and transmission management, I/O network communication and other services. The highest layer aggregates the installer, the framework classes API and industry specifications libraries [2].

The JCVM, much like the typical JVM (Java Virtual Machine) is a virtual machine capable of executing Java Card bytecode. It provides an environment in which Java bytecode can be executed, through a set of standard class libraries that implement the Java API (Application Programming Interface). The architecture of the JCVM is split into two components, the converter and the interpreter. While the converter is placed out of the Java Card in order to process the input used in the applet installation, the interpreter runs inside, providing the environment where the applets run [2].

JCRE system class are the core of the operating system abstraction and are responsible for managing applets, transactions, communications and other services. These class have permissions to directly invoke native methods of the Smart Card.

The framework classes are placed on the upper layer, composed by packages providing an API for the development of Smart Card applets. The industry specifications are a set of extra libraries to provide additional features directed to the purpose of specific industry needs [2].

Due to the resource constraints in Smart Cards, the implementation of the Java platform is not equal to the generic one. This implementation incorporates only a subset of features of the Java language and some additional mechanisms that provide specific support for the Smart Card Java applets. These extra features, implemented by the Java Card Runtime Environment are the definition of persistent and transient objects, the specification of atomic operations and transactions, and the specification of an applet firewall with sharing mechanisms [2].

The objects can be persistent in memory or be cleaned across CAD (Card Acceptance Device) sessions. In fact, the space allocated to an object is reserved while it is referenced, so an applet should create just one transient object, during its lifetime, and save the object reference in a persistent field. By default, all the objects instantiated are persistent. The atomicity in Java Cards ensures that any update to a single field, in a persistent object or class, is atomic. Java Card technology also supports a transactional model, in which a set of updates can be done inside a transaction. In this case, each and all of the updates are successfully done or all the fields return to their previous states [2].

The Java Card platform is a multi-application environment with security requirements, which is the reason why it has an applet firewall. This applet divides the context of the applets into separate and protected spaces, the contexts of the applets, protecting them against the alteration of sensitive data by other applets. The isolation provided by this applet can be compared to the sandbox where the web browser applets run. However this strict separation is not always convenient, especially in situations where it is really necessary to exchange data between applets. In this sense Java Card technology is provided with sharable mechanisms, allowing applets to share data across the contexts of applet's firewall. These mechanisms are privileges, entry point objects, global arrays and shareable interfaces (to be discussed in section 2.5) [2].

2.2 I/O

The interaction between a Java Card and the external devices is performed using a CAD. A CAD can be considered a reader or a terminal, depending on his capability to also process information. The CAD reader works as a communication interface to the external world, but also as a source of power supply. During a CAD session the JCRE has a typical Smart Card behavior with the host, exchanging APDU (Application Protocol Data Unit) in a master-slave model. In this scheme the card is the slave, therefore just responding to the commands received from the host [10].

Java Cards in its normal state are constantly waiting for APDU commands, targeted to one of the previously installed applets. Each APDU command has a class and an instruction field indicating the instruction to be performed and defining the category of the command request and response. The mandatory header also includes two parameters fields, P1 and P2, containing information for the command instruction. The optional APDU body is composed by the length of the data, followed by the data itself. The response has a mandatory two byte status word, and optionally a response body preceded by its length [2].

Before the first instruction, the applet has to be selected. When a command arrives to the card, the previously selected applet runs the specific instruction called. After the execution of the command, the card sends the response APDU to the host. The communication provided by the JCRE supports the T=0, T=1 and contactless transport protocols, and the APDU format defined in the ISO7816-4 standard [2].

2.3 Middleware

The interaction with an applet entails the development of a Middleware, to run on the host side, with the propose of encapsulating the complexity of the communication with the card. This Middleware has to be robust and cross platform, in order to provide an independent and useful API to the highest level host applications. In a real system, the Middleware resides in the terminals or in the card acceptance devices such as workstations, POS (Point Of Sale) terminals, mobile phones among others.

PC/SC (Personal Computer/Smart Card Interface) is an industry standard from Microsoft, defining an API for communication with Smart Cards. A wide number of the available device readers for PC, already have this Middleware implementation available. The central component of its architecture is the ICC (Integrated Circuit Card) Resource Manager, to control all the interface devices and service providers [4].

There are several ways of implementing a Middleware. In order to develop an applet with the propose of remote authentication, one of the best ways is to ensure that the Middleware can be used by many systems, implicating the use of PKCS#11. The PKCS#11: Cryptographic Token Interface Standard, specifies an API, to devices which hold cryptographic information and perform cryptographic operations.

The main propose of the PKCS#11 standard interface is to encapsulate the complexity of the communication with the card and with the applet. With the Middleware providing this standard interface, any high level code can run the standard methods to a wide variety of implementations, in a wide variety of devices. To the applications, this Middleware presents a common and logical view of a device, called cryptographic token.

2.4 Applets

Each applet is a persistent object in the Java Card. If the card is not prepared to delete applets, once installed, an applet lives in the card for its entire lifetime [2]. In fact, in the newest versions of the Java Card technology it is already mandatory to be able to remove applets. All the applets are identified by an AID, defined in the registration step. The applets have to extend from the *javacard.framework.Applet* class, to implement some of the crucial methods, such as *Install*, *Register*, *Select*, *Process* and *Deselect* [2].

An instance of the applet is created with the *Install* method. Thus, each applet must implement in the constructor all the necessary initializations. Due to the fact that not all implementations of the Java Card have a garbage collector available, it is useful to instantiate each application just one time in the life time of the card. The *Register* method should be the last instruction called in the installation method implementation. After that, the applet is ready to be selected and set to run [2].

The life-cycle of an applet starts with its installation and registration on the card. After that, each applet switches its state between inactive and active. This happens, at least in version 2.2, because the JCRE is a single thread environment, which means that just one applet can be active at each moment [2]. To activate an applet one needs to send a *Select* command to the card, with the AID of the corresponding applet. The *Deselect* command resets the state of the applet to inactive.

The APDU commands received by the card are executed by the current active applet through the *Process* method. The applet, receiving the APDU as an argument of the *Process* method, is responsible for analyzing it and executing the correspondent instruction.

2.5 Shareable Interface Object

With the restriction policy imposed by the applet firewall, applets out of the context of other applets cannot have access to their methods or attributes, even if they are public. This is a good security measure to prevent attacks to the applets. However in some cases it is useful for an applet to share its data and methods with applets from other contexts.

One of the mechanisms available in Java Card technology to enable this method exchanging is the Shareable Interface. The object of a class that implements a shareable interface is called SIO (Shareable Object Interface). The applets that want to call at least one of the methods of the SIO have to declare an object with the type of the interface and get a reference to the SIO on the JCRE. Whenever an applet calls a method from a SIO, the JCRE switches the context of the applet to the context of the SIO. The result is then returned to the caller applet, after its context becomes activate again.

With this mechanism, applets out of the context of the Shareable Object Interface are allowed to access the methods defined in the shareable interface. This interface works like a public interface for messages that have permission over the applet firewall, allowing the transmission of information between objects from different contexts.

2.6 Biometrics in Java Card

A biometric security system is a secure and reliable method to provide person identification and authentication. Considering the safety and convenience of smart cards, becomes interesting to join these two technologies, in order to develop a system with the advantages of both. Besides storing the biometric templates, these devises implement the match-on-card mechanism, ensuring a closed authentication system and protecting the templates from any outside authority [3].

In order to create interoperability on the biometric management and match template on Java Cards, the Java Card Forum [1] proposed a JC-BioAPI (Java Card Biometric API) [3]. This API takes into account the enrollment phase of the biometric templates on the card, and a security biometric validation, avoiding sensitive data to be exposed to the outside of the card. Since the Java Card technology tries to ensure interoperability between Smart Card operating systems, this API can be used in a wide range of different cards. Besides this, the JC-BioAPI also supports multiple biometrics on a single card [3].

The JC-BioAPI is designed to use limited amounts of the Smart Card's memory and as few computational cycles as possible. This API allows the independent development of applets and biometrics, creating all the needed abstraction to the developers. Its architecture is divided in two modules, the server and the client applets. The server applet is intended to manage the enrollment and configuration of the biometrics on the card and to provide the shareable interfaces to other applets. It also controls the access to the Match-On-Card, and prevents the access to the stored templates. Therefore, the other applets have to call the JC-BioAPI applet through a SIO, in order to obtain access to the biometric methods [3].

The interfaces defined by the native JC-BioAPI are the *BioTemplate*, the *OwnerBioTemplate*, and the *SharedBioTemplate*. The *BioTemplate* is the top hierarchical interface from which the other two interfaces are extended. The *OwnerBioTemplate* defines the interface to the management of the biometric system. These methods are designed to be called from outside of the card, with APDU commands, allowing the enrollment and the verification of the biometric templates. When the server applet is instantiated, it creates a SIO instance of the *SharedBioTemplate* interface, providing the methods to be used internally by the other applets [3].

3 Biometric Authentication

"Biometric recognition is the use of distinctive anatomical and behavioral characteristics or identifiers to automatically recognize a person" [7]. The anatomical properties are related to the shape of the body, while behavioral properties are related to the behavior of a person. A biometric authentication system can be achieved through strong security, efficiency, reliability and the convenience for the users [7]. These advantages come from the fact that biometric characteristics are part of each individual and specific to each one. In this sense, no one can share their biometrics with a friend or even lose them. Biometrics are, thus, more difficult to abuse than traditional methods of identification and extremely difficult to guess, share, misplace, copy, or forge [9].

3.1 Biometric Characteristics

The main characteristics desired in a biometric feature are universality, distinctiveness, permanence, and collectability. The universality ensures that all people have the biometric characteristic, thereby avoiding limiting the system to a group of specific users. The distinctiveness ensures that

[1] http://www.javacardforum.org/

all biometric samples of different persons are different, which is important to avoid that someone authenticates himself as another user. The permanence ensures that a biometric sample is time-invariant and thus, an individual can be identified through a sample collected for a long time. Ultimately, the collectability ensures that biometrics can be taken from their carrier and their traits quantified. It is this characteristic that allows for an automatic system to perform a biometric authentication of a person [9].

3.2 Biometric System Architecture

An authentication system based on biometrics consists of two important phases, the enrollment and the recognition. The enrollment phase consists on the registration of the user in the system, by collecting and storing his biometric templates, while in the recognition phase the authentication is performed.

In the enrollment phase the identification of the user is linked with his biometric template. This phase is composed by 4 main steps, the capture, the feature extraction, the template creation and the data storage [9]. The capture step consists on the sensing of the biometric traits. Normally, in this phase, a quality analysis is done in order to determine if the capture has reached a certain level of quality, or if it is necessary to perform another capture [6]. After the capture step, comes the feature extraction, in which the capture previously done is analyzed. This step consists basically on the creation of the biometric representation, resultant from the computing of the biometric particularities. The template is the raw material that is used in the matching phase of recognition. This template should be stored in the system database linked to the identification of the person, who is being enrolled in the biometric authentication system [7]. This database can be central, on the authentication biometric system, or distributed in tokens like smart cards [6].

The recognition phase follows a protocol very similar to the enrollment. First a capture of the biometric traits is done, followed by a feature extraction step. The difference is on the final steps of template creation and data base storage that are replaced by the matching stage. The capture follows the same quality analysis to provide a reliable source to the feature extraction step, and the obtained biometric data is compared with the stored template [7].

3.3 Fingerprints

Fingerprints are one of the most used biometric characteristics. They are considered as a very useful and reliable source of authentication, being already a mature technology. Their main biological principles are the individual epidermal ridges and furrows on each person. Another important characteristic lays on their invariance, making the fingerprint permanent and unchanged in a person's life. Despite the existence of a wide range of characteristics to analyze in fingerprints, these characteristics are within reasonable limits, allowing their classification [5].

The extraction feature and analysis phase considers two levels of information on each fingerprint image. The first level considers the ridge line flow, ranging between left and right loop, whorl, arch and tented arch. The second level of analyzes is able to obtain up to 150 different ridge local characteristics between bifurcation, ridge ending, dot, ridge crossing, etc. All this features and reference location are collected and stored in the biometric template [7].

4 Proposed Solution

The proposed solution is a secure and reliable remote authentication system, using biometric signature validation. The remote authentication is performed by a Java Card, using asymmetric cipher with the private key in that card. However, the access to operations with this private key is restricted, requiring a valid local authentication procedure. In the proposed solution this local authentication is performed with the use of fingerprint biometrics.

The remote authentication implies the use of an applet inside the Java Card, responsible for the remote authentication procedures. The most common solution is the use of the IAS (Identification, Authentication, and Signature) standard, used in the most of the ID authentication solutions, as is the case of the Portuguese Citizen Card. The local user biometric authentication is provided by the MOC (Match-On-Card) applet. This applet is also present in ID cards, as a source of local fingerprint validation.

However, in the existing ID cards, the MOC applet only provides local biometric authentication to the external interfaces, not communicating with the other applets in the card. The solution herein proposed is to design a proxy applet that replaces the PIN login procedure in the IAS applet, by the fingerprint biometric usage. This applet is installed in the card, to provide the necessary management and correlation between the IAS and the MOC applets, becoming the front end of the system, and the communication door to the outside world of the card. Therefore, this solution aims to be generic and compatible with existing systems.

The following starts by giving an overview of the IAS applet and its usage in today's world, followed by the presentation of the architecture of the proposed solution and developed prototype.

4.1 IAS

Most of the current ID solutions using Smart Cards in UE use the IAS (Identification, Authentication, and Signature) standard. The IAS is responsible for the digital identification and authentication of an user, among other e-services. The IAS ECC (European Citizen Card) is a standard for European cards for e-services and national e-ID applications, used for example in the Portuguese Citizen Card. This standard is intended to standardize the digital authentication of the European Union citizens, as well as normalize the Middleware used to communicate with the e-ID systems of each country.

The IAS ECC applet performs the user remote authentication using an asymmetric encryption mechanism. The CA of each country generates a public and private keys and a signed public key certificate of each card. Nevertheless, each country CA is certified by a global CA. This chain of certification ensure the universal acceptance of this mechanism, as a reliable and secure way for citizen authentication. With this scheme, the in the card local user authentication is ensured by a PIN mechanism, protecting the operations that requires the use of the private key stored in the card.

The main procedures provided by the IAS applet, to ensure the remote authentication of an user, are *Select*, *Compute Digital Signature*, and *Verify*. With the first it is possible to select the IAS applet in the card. The second is intended to obtain a digital signature of a challenge sent to the card, while the *Verify* is used to perform the user local authentication.

The solution herein presented aims to be as generic as possible, in order to be deployed in already existing systems. The work presented in this paper considers the use of an applet, following the widely use IAS standard. Nevertheless, any authentication applet solution can be used, being enough to share some public methods with other applets, as detailed throughout this section.

4.2 Architecture

Figure 1 illustrates the overall structure of the proposed solution, and the several needed components. The IAS applet is the considered source of remote authentication, given its properties and generalization. The local user authentication using biometrics is performed by the MOC (Match-On-Card) applet. Given that the MOC and the IAS applets are completely independent, the proxy applet is used. Thus, allowing for the local user biometric authentication in the IAS, using the MOC.

The proxy applet is the card front end, therefor all the commands that needed to be sent to the IAS applet, are done so via the proxy applet. This applet performs the redirection of these commands to the IAS applet throw a SIO. This solution enables the use of biometric rather than PIN based authentications, in already existent systems, being only added the SIO to the IAS.

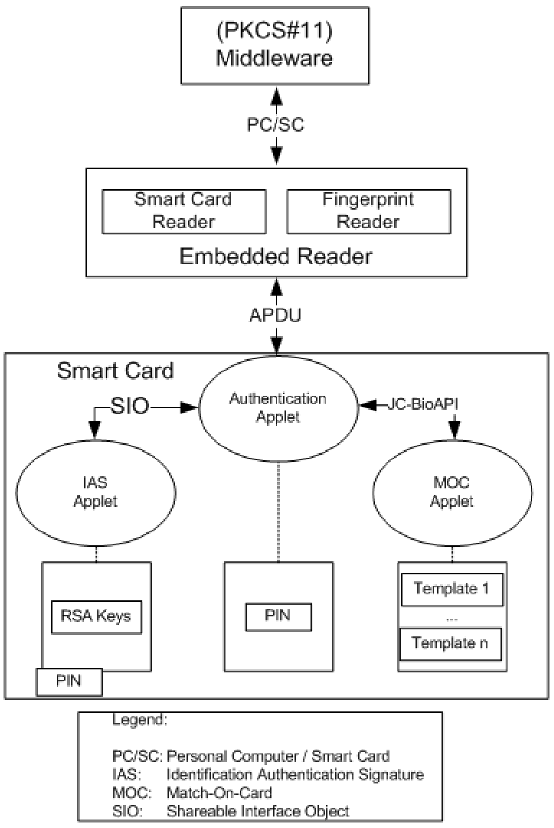

Fig. 1. Proposed solution

The flow of an authentication inside a card, starts in the proxy applet. After receiving the remote authentication request, which corresponds to a sign command with the authentication private key, the proxy applet redirects it to the IAS. The IAS will throw an error exception, indicating the need for PIN input. The proxy applet catches this request and generates a external APDU message, indicating the need to be performed a biometric authentication.

The following biometric authentication commands, also pass through the proxy applet, but are redirected to the MOC applet via the JC-BioAPI, described in section 2.6. If a successful biometric authentication is performed, the MOC returns a valid authentication message to the proxy applet. At this point the proxy unblocks the private key usage in the IAS applet. As designed, the IAS only unblock this key usage, when a correct PIN is entered. Thus the proxy sends this PIN to the IAS.

The proxy applet, after being installed, change the PIN used by the IAS to perform the user local authentication. Therefore, the user is unaware of this PIN value, since only the proxy "know" this value. After the successful PIN authentication in the IAS, the proxy applet returns to the external world with an authentication *OK* status code, and the card will be on an "authenticated mode" to perform that PIN-protected operation, waiting for the correspondent command (APDU) to be submitted. Therefore, after a successful user authentication, the authentication *sign* command can then be called and correctly executed.

The host side applications use the PKCS#11 standard interface to abstract from the Smart Card details. Between the proposed system and other existing schemes, the difference resides only in the Middleware implementing the PKCS#11 interface. The Middleware implementation takes advantage of the PC/SC (Personal Computer/Smart Card) communication interface, in order to abstract the complexity of the communication with a wide variety of the existing Smart Card readers.

4.3 Proxy Applet

Current identification and authentication Smart Cards applets, as the IAS, are responsible for the cryptographic operations performed in the remote authentication of users, ensuring first his local PIN authentication. In order to be developed a local biometric authentication instead of a PIN authentication, a proxy applet is added to the Card.

The methods that needed to be defined in the IAS *ShareableInterface* are intended to allow the redirection of the commands from the proxy applet and to allow the proxy applet to perform some operations, namely the PIN change and PIN verification. Therefore, the IAS methods that need to be shared are *process*, *verifyPin*, and *changePin*. Sharing the *process* method, enables the proxy applet to check all received commands, and after verifying that they are destined to the IAS, call the IAS SIO. With this, the IAS applet processes the command as if it was directly sent to it.

The other two methods are the key to ensure that the proxy applet becomes the front-end for the card, preventing the direct access to the IAS Applet from the outside. Hence, the proxy applet has to change the user PIN on the IAS, in order to be the only to know it. Thereafter, whenever a user tries to authenticate himself towards IAS applet, by directly selecting this applet, a wrong PIN will be entered. The authentication mechanism becomes thus, fully managed by the proxy applet.

Given that the function of the proxy applet is to coordinate and redirect the operations, the needed cryptographic operations are still performed by other applets such as the IAS and the MOC applets. With this solution, the modifications needed in the applets used for the user remote authentication are based only in the sharing of methods already implemented in the already deployed applets. The proxy applet is only responsible for the coordination of the local user authentication process, when the main code are in the already existent applets. Thus, considering the current user remote authentication scenarios using smart cards, the increased of memory and processing required by the proposed solution is rather small.

4.4 Developed Prototype

In order to test the presented solution, a prototype implementation was developed and tested in a web remote authentication system, as depicted in Figure 2. In this figure, the *BioSCAuth* represents the proposed solution depicted in Figure 1. The authentication is based on successfully establishing an HTTPS (HyperText Transfer Protocol Secure) connection between the browser and a remote web server. In this connection the asymmetric cipher operations used in the SSL (Secure Sockets Layer) authentication phase are performed with the private key of the user, by the Java Card.

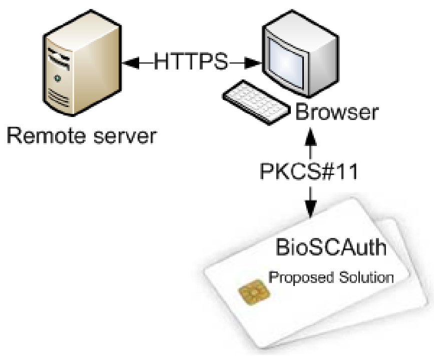

Fig. 2. Test scheme

Given that the existing IAS applets are proprietary, a IAS applet was developed, implementing the need methods for the validation test. The proxy applet and the required Middleware were also developed. The applets were developed using the JCDK (Java Card Development kit) developed by Sun [2]. The PKCS#11 Middleware interface was implemented using C++.

This prototype uses a fingerprint biometric sensor from Precise Biometrics [3], with a Smart Card reader embedded. Their framework was used to generate the biometric template to be match by the on card MOC applet. Moreover, wherever a signature command is sent to the proxy applet, the Middleware stores the command in memory, if a biometric authentication is required by the card. Thus, after the user has performed a successful authentication, the Middleware automatically performs the stored signature command call, without the user having to make the request again.

5 Security Analysis and recommendations

The security analyze to the proposed system is performed in this section, considering the possible breaches and the conditions that have to be warranted during its use. This analysis covers the entire system and gives an important notion of what precautions have to be taken when using it.

The Middleware is the entity outside the card responsible for coordinating the user local biometric authentication process. When the proxy applet in the Java Card, requests the biometric authentication, the Middleware requests the user to put his finger on the biometric reader. The biometric template is then generated using the image captured by the sensor, and sent to the Java Card. The ideal solution is to use an embedded device with the Smart Card reader and the biometric sensor. With this solution it can be ensured that the generated biometric template does not leaved the secure and exclusive environment between the biometric reader and the Smart Card, and thus no outside entity can access it.

In this solution, the reader device is responsible for the template generation and to send it directly to the card, assuring that biometric data is acquired in the moment of the authentication, an no external pre-generated template can be used. A secure channel can be established between the host side application and the proxy applet inside the card during the biometric authentication, in order to assure that only certificated Middleware, using embedded biometric and Smart Card readers, can be used.

Despite the fingerprint biometric validation being a very secure and reliable mechanism, in comparison with the usual security systems based only in passwords, the consequences of a forged biometric are rather serious. When a biometric system is compromise, the forged template cannot be used again. Forging a fingerprint requires expertise and laboratory equipment to create a fake finger, making it a less viable type of attack. To improve the biometric readers additional protection mechanisms can be added such as verification of blood pressure, conductivity, temperature, and detection of the human skin [9]. This system can also use other biometrics, more difficult to forge than fingerprint, or even combine the biometric authentication with a PIN base user authentication.

At the applet level it is necessary to ensure that the IAS applet cannot be called directly from the outside world. To ensure this, it is necessary for the proxy applet to set a different PIN in the IAS, only known by the proxy. Identically to the PIN authentication, the biometric authentication also has a limited amount of authentication attempts, imposed by the MOC applet.

The protection of the user biometric template and private information is assured since it is stored in a secure area of the Java Card, a trusted device capable of achieving a Common Criteria certification of EAL 5. However, the installation of the applets in the card, must be performed following the Global Platform specifications in order to ensure that only checked applets are installed in the system Java Cards. Moreover, the IAS applet can share its SIO implementation only with an authenticated proxy applet, by checking its AID (Applet Identifier) and changing a secret password.

[2] http://java.sun.com/javacard/devkit/
[3] http://www.precisebiometrics.com/

6 Conclusion

This paper presents a solution to enable the use of biometrics in remote authentication systems using SC (Smart Cards). The biometric authentication cannot be directly used in remote scenarios, given that it is not secure to send the users biometric template over public networks. Herein a solution is proposed, looking at the existing authentication systems with SC and using local Biometric authentication. The proposed solution solves the main problems with the use of biometrics, in current remote authentication systems.

Most of the current e-ID solutions use the IAS standard to provide remote authentication via asymmetrical encryption. The use of the private key is restricted to the trusted device, the SC, and are only allowed after a successful PIN entry. In the proposed solution a proxy applet is used in order to coordinate the biometric authentication in the on card MOC, and the local PIN entry in the IAS. The proxy applet is the front end of the system, therefore the Middleware only needs to communicate with it. However, the effort and requirements needed to implement and support this applet are minimal, compared with the cost of the current applets used in the user remote authentications. This proxy can be easily modified to support other biometrics systems than fingerprints.

In a authentication operation, the proxy applet uses the received user biometric template to perform the biometric authentication in the MOC applet. If this biometric authentication succeeds, is the proxy applet that executes the PIN authentication in the IAS. After the proxy applet installation, the user no longer has direct access to the IAS applet, since the its PIN is change by the proxy. This solution ensures a reliable and secure way to perform remote authentication, with a local biometric user identification, and can be implemented in the current solutions with few modifications, and low footprint.

References

1. C.-C. Chang and T.-C. Wu. Remote password authentication with smart cards. *IEE Proceedings E Computers and Digital Techniques*, 138(3):165, 1991.
2. Zhiqun Chen. *Java Card Technology for Smart Cards: Architecture and Programmer's Guide.* Addison-wesley, Boston, USA, 2004.
3. Biometric Consortium and Working Group. Java card: Biometric api white paper. *Architecture*, (August):02–0016, 2002.
4. D Husemann. Standards in the smart card world. *Computer Networks*, 36(4):473–487, Jul 2001.
5. A.K. Jain, S. Pankanti, and R. Bolle. An identity-authentication system using fingerprints. *Proceedings of the IEEE*, 85(9):1365–1388, 1997.
6. Anil K. Jain, Patrick Flynn, and Arun A. Ross. *Handbook of Biometrics.* Springer, London, UK, 2007.
7. Davide Maltoni, Dario Maio, Anil K. Jain, and Salil Prabhakar. *Handbook of Fingerprint Recognition.* Springer, London, UK, 2 edition, 2009.
8. Keith E. Mayes and Konstantinos Markantonakis. *Smart Cards, Tokens, Security and Applications.* Springer, London, UK, 2008.
9. S. Prabhakar. Biometric recognition: security and privacy concerns. *IEEE Security*, 4677(2):275–42, mar 2003.
10. Wolfgang Rankl. *Smart Cards Applications: Design models for using and programming smart cards.* Wiley, West Sussex PO19 8SQ, England, 2007.
11. U. Uludag. Biometric cryptosystems: issues and challenges. *Proceedings of the IEEE*, 42(8):136 to 960, June 2004.
12. Fifth Usenix, Nordu Conference, Damien Deville, Antoine Galland, and Gilles Grimaud. Smart card operating systems : Past , present and future. pages 1–18, 2003.