Biometrics to Enhance Smartcard Security
Simulating MOC using TOC *

Giampaolo Bella[1,2], Stefano Bistarelli[3,4], and Fabio Martinelli[4]

[1] Computer Laboratory, University of Cambridge, UK
giampaolo.bella@cl.cam.ac.uk

[2] Dipartimento di Matematica e Informatica, Università di Catania, Italy
giamp@dmi.unict.it

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

[4] Istituto di Informatica e Telematica, CNR, Pisa, Italy
{stefano.bistarelli,fabio.martinelli}@iit.cnr.it

Abstract. A novel protocol is proposed to address the problem of user authentication to smartcards using devices that are currently inexpensive. The protocol emulates expensive *Match On Card* (MOC) smartcards, which can compute a biometric match, by cheap *Template on Card* (TOC) smartcards, which only store a biometric template. The actual match is delegated to an extension of the cryptographic module running on the card host, which is called *Cryptoki* according to the PKCS#11[9] standard. Compliance to such a standard increases the portability of the protocol. Informal reasoning confirms the protocol strenghts, though its formal verification in terms of established equational techniques appears to be at hand.

1 Overview

Smartcards are currently used as a secure and tamper-proof device to store sensitive information such as digital certificates, private keys and personal information. Access to smartcards has historically been regulated by a trivial means of authentication: the Personal Identification Number (PIN). A user gains access to a card if he enters the right PIN.

Experience shows that PINs are weak secrets [6] in the sense that they are often poorly chosen, and that they are easy to loose. Moreover, many actual implementation that use the PIN, consider the channel between host and smartcard secure. So, they simply send the PIN in a clear communication. This imply many easy attacks [6]. A simple Trojan on the host could easily sniff the PIN and store it for future usage[1].

* This work has been partially supported by MIUR project "Constraint-based Verification of Reactive Systems (COVER)"; by MIUR project "Tools, techniques and methodologies for the information society" and by a CSP grant for the project "SeTAPS II".

[1] We check four different brand of cryptographic smartcards using each his Cryptoki implementation and we was able to capture the PIN of all of them.

Biometric technologies have been proposed to strengthen authentication mechanisms in general by matching a stored biometric template to a live biometric template [3, 2, 4]. In the case of authentication to smartcards, intuition imposes the match to be performed by the smartcard. But this is not always possible because of the complexity of biometric information such as fingerprints or iris scans, and because of the yet limited computational resourses offered by currently available smartcards.

In general, three strategies of biometric authentication can be identified.

Template on Card (TOC). The biometric template is stored on a hardware security module (smartcard or USB token). It must be retrieved and transmitted to a different system that matches it to the live template acquired by special scanners from the user. Cheap *memory-cards* with no or small operating systems and onboard applications are generally sufficient for this purpose.

Match on Card (MOC). The biometric template is stored on a hardware security module, which also performs the matching with the live template. Therefore, a *microprocessor* smartcard is necessary, which must be endowed with an operating system running suitable match applications.

System on Card (SOC). This is a combination of the previous two technologies. The biometric template is stored on a hardware security module, which also performs the matching with the live template, and hosts the biometric scanner to acquire, select, and process the live template.

Our research aims at the development of a viable and easy-to-deploy protocol to allow secure authentication of users to smartcards, enforced by biometric matching. Clearly, if we could cast a spell, we would choose the third of the strategies sketched above. Indeed SOC cards appear to be the best in terms of security, as everything takes place on card. Unfortunately, the existence of SOC cards is still confined to research laboratories at present [8, 11]. The benefits deriving from MOC cards are valuable themselves, although these cards must be still considered rare and expensive. Reasonably cheap microprocessor cards do come with a cryptographic processor, which unfortunately is too specialised to perform biometric matches. Going down the hierarchy of prices, we find that TOC cards are obviously inadequate.

In the present setting, how can we implement biometric authentication on smartcards that are commercially available? We address this issue by developing a novel protocol that employs inexpensive TOC cards as if they were MOC cards. The actual match is carried out by a module added to the *Cryptoki* application running on the card host according to the PKCS#11 standard defined by RSA laboratories [9]. In principle, if we trust the Cryptoki module, then we should protect the extra exchange that we expect is necessary to transmit the stored template from the card to the Cryptoki. By doing so, the resulting protocol should not be more vulnerable than the corresponding protocol for MOC cards. Our ongoing work on formally analysing the proposed protocol appears to confirm such a claim.

The presentation of the protocol messages may reveal possible optimizations. The current design is imposed by the restricted set of functions that we want to implement in the applet we need to load on the smartcard. We have a working alpha implementation of the protocol, whose details will be published in upcoming papers [1]. Another important remark is that our design is obtained incrementally from a shorter design for MOC cards. Hence, it will be easy to simplify the protocol into one for MOC cards, as these become more affordable.

After a brief introduction to PKCS#11 Cryptoki (§2), this paper describes our novel protocol (§3), sketches its ongoing formal analysis (§4), and concludes (§5).

2 PKCS#11: Cryptographic Token Interface Standard

The PKCS#11 standard specifies an API, called "Cryptoki" (cryptographic token interface), to devices that hold cryptographic information and perform cryptographic functions [9]. Cryptoki follows a simple object-based approach, addressing the goals of technology independence (any kind of device) and resource sharing (multiple applications accessing multiple devices), presenting to applications a common, logical view of the device called a "cryptographic token". The primary goal of Cryptoki is a lower-level programming interface that abstracts the details of the devices, and presents to the application a common model of the cryptographic device, called a cryptographic token (or simply token).

Cryptoki's general model is illustrated in Figure 1. The model begins with one or more applications that need to perform certain cryptographic operations, and ends with one or more cryptographic devices, on which some or all of the operations are actually performed. A user may or may not be associated with an application. Usually, a cryptographic device can perform some crypto-

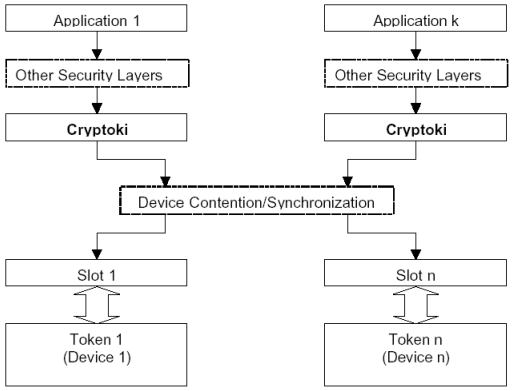

Fig. 1. A general Cryptoki model.

graphic operations, following a certain command set. For example, a smartcard

can generate a nonce, but in practice cannot perform any operation by itself. To have the smartcard generate a nonce, the application should invoke function `C_GenerateRandom(...)` of the Cryptoki library. Figure 2 represents such interactions. To let the smartcard able to interact by itself and follow the pro-

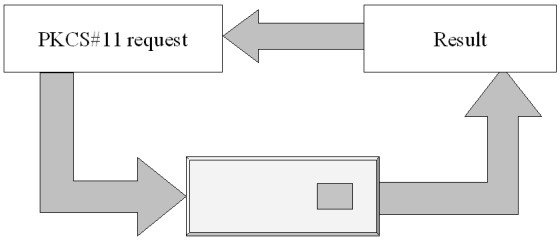

Fig. 2. The request-answer PKCS#11 model.

tocol rule, we used a javacard compliant smartcard and we load a small applet responsible to run the protocol with the Cryptoki.

The most important operation that is requested using the PKCS#11 interface is the login request. Following PKCS#11 standard, we implement a biometric login starting the protocol when a *login*(*NULL*) request is performed from the application.

3 Simulating MOC using TOC

This section presents our protocol.

We follow an incremental design. First, we build a (PKCS#11 compliant) protocol for MOC cards. Then, we extend this protocol to function with the cheaper TOC cards. The price paid is one extra round of messages. While the latter protocol can be used at present, the former is its obvious "modification" that will become mandatory when technology makes MOC cards cheaper.

3.1 A MOC protocol

A MOC protocol can be defined as in Figure 3. In short, when the application send the *login*(*NULL*) request to the criptoky, it start the protocol with the smartcard, by asking for a nonce. This will be used for a challenge-response phase to authenticate the Cryptoki to the smartcard (messages 1 and 2 and 3). In message 3 the Cryptoki also transmit the just-acquired live template to the card, which check the match and acknowledges (message 4).The session key generated by the smartcard could be used for all the next communication between the host and the smartcard. The live template is sent encrypted with a session key to ensure user privacy. A fuller description is given below, where this protocol is extended to delegate the biometric match to the Cryptoki. The key K_{sess} is the

session key, N_{SC} is the nonce, K_{CK} and K_{CK}^{-1} are the public and private key of the Cryptoki, respectively.

$$
\begin{array}{lll}
1.\ \mathsf{Cryptoki} & \rightarrow \mathsf{SmartCard}: & hello_{MOC} \\
2.\ \mathsf{SmartCard} & \rightarrow \mathsf{Cryptoki} : & \{\!|\{\!|N_{SC}, K_{sess}|\!\}_{PIN}|\!\}_{K_{CK}} \\
3.\ \mathsf{Cryptoki} & \rightarrow \mathsf{SmartCard}: & \{\!|\{\!|N_{SC}, Template_{live}|\!\}_{K_{CK}^{-1}}|\!\}_{K_{sess}} \\
4.\ \mathsf{SmartCard} & \rightarrow \mathsf{Cryptoki} : & \{\!|N_{SC}, Hash(Template_{live})|\!\}_{K_{CK}}
\end{array}
$$

Fig. 3. A MOC protocol.

3.2 A MOC-using-TOC protocol

While the previous protocol requires MOC cards, the protocol presented here can do with cheap TOC cards. Figure 4 shows the new protocol, which extends the previous with two additional steps between 3 and 4. The additional steps essentially delegate the matching to a special Match Module of the Cryptoki. The keys K_{MM} and K_{MM}^{-1} are the public and private key of the Match Module, respectively.

$$
\begin{array}{lll}
1.\ \mathsf{Cryptoki} & \rightarrow \mathsf{SmartCard} : & hello_{MOC} \\
2.\ \mathsf{SmartCard} & \rightarrow \mathsf{Cryptoki} : & \{\!|\{\!|N_{SC}, K_{sess}|\!\}_{PIN}|\!\}_{K_{CK}} \\
3.\ \mathsf{Cryptoki} & \rightarrow \mathsf{SmartCard} : & \{\!|\{\!|N_{SC}, Template_{live}|\!\}_{K_{CK}^{-1}}|\!\}_{K_{sess}} \\
3_1.\ \mathsf{SmartCard} & \rightarrow \mathsf{Cryptoki}_{\mathsf{match}} : & \{\!|Template_{live}, Template_{storedAC}|\!\}_{K_{MM}} \\
3_2.\ \mathsf{Cryptoki}_{\mathsf{match}} & \rightarrow \mathsf{SmartCard} : & \{\!|Hash(Template_{live}, Template_{storedAC})|\!\}_{K_{MM}^{-1}} \\
4.\ \mathsf{SmartCard} & \rightarrow \mathsf{Cryptoki} : & \{\!|N_{SC}, Hash(Template_{live})|\!\}_{K_{CK}}
\end{array}
$$

Fig. 4. A MOC-using-TOC protocol.

First phase: Cryptoki's authentication to SmartCard, and transmission of the Live Template

$$
\begin{array}{lll}
1.\ \mathsf{Cryptoki} & \rightarrow \mathsf{SmartCard}: & hello_{MOC} \\
2.\ \mathsf{SmartCard} & \rightarrow \mathsf{Cryptoki} : & \{\!|\{\!|N_{SC}, K_{sess}|\!\}_{PIN}|\!\}_{K_{CK}} \\
3.\ \mathsf{Cryptoki} & \rightarrow \mathsf{SmartCard}: & \{\!|\{\!|N_{SC}, Template_{live}|\!\}_{K_{CK}^{-1}}|\!\}_{K_{sess}}
\end{array}
$$

In this phase a challenge-response authentication between SmartCard and Cryptoki takes place. The SmartCard issues a fresh nonce and the Cryptoki

returns it. The challenge response phase implies the presence of the user that know the PIN associated to the smartcard. In fact, only with the knowledge of the PIN message 2 can be decrypted to obtain the nonce (this is a first step of authentication where a *what I know* feature of the protocol is used).

The next step is to create a message representing the biometric information of the user (a *what I am* feature). A *live* and *trusted* template is generated. The idea is to accept biometric template only if coming from a trusted device (the signature show this) and only if acquired during the current authentication session (the presence of the nonce ensure this). We can assume that the Cryptoki public key is already available to the smart card, otherwise the Cryptoki should just send the appropriate digital certificate with message 1.

The session key sent to the Cryptoki from the SmartCard will be useful to encrypt the biometric template for privacy reasons. The main reason to use a session key is because the user may want to secure his biometric information from eaves-droppers for a number of reasons. For example, one is that the fingerprint can reveal certain diseases. The same session key could be used for all the communication following the logon phase.

Second phase. SmartCard's transmission of stored template to Cryptokimatching module for the match.

3_1. $\mathsf{SmartCard} \rightarrow \mathsf{Cryptoki_{match}}$: $\{\!|\mathit{Template}_{live}, \mathit{Template}_{storedAC}|\!\}_{K_{MM}}$

3_2. $\mathsf{Cryptoki_{match}} \rightarrow \mathsf{SmartCard}$: $\{\!|\mathit{Hash}(\mathit{Template}_{live}, \mathit{Template}_{storedAC})|\!\}_{K^{-1}_{MM}}$

The SmartCard sends the stored template to the Cryptoki Matching Module, who is delegated the responsibility of doing the match. This step is secured under the key of the Cryptoki Matching Module for privacy reasons. If the live template matches the one stored in the attribute certificate $\mathit{Template}_{storedAC}$ then the Match module issues a signed message confirming such a match [2]. The use of an Hash function ensure that no information related to the fingerprints template is leaked (as before for privacy reasons).

Last phase: Successful login

4. $\mathsf{SmartCard} \rightarrow \mathsf{Cryptoki}$: $\{\!|N_{SC}, \mathit{Hash}(\mathit{Template}_{live})|\!\}_{K_{CK}}$

[2] If we also assume that the acquisition of the live template by the Cryptoki is performed in a confidential way, this message may embody a nice form of challenge/response authentication, using the pair $(\mathit{Template}_{live}, \mathit{Template}_{storedAC})$ as a nonce, due to the unpredictable nature of the $\mathit{Template}_{live}$. Such a form of freshness, although is not crucial for the security of the protocol, since, in principle the result of the matching among the live template and the stored should be always the same (thus replay attacks are not worthy). However, in a real implementation, such a form of freshness could be useful is we imagine that the accuracy of the matching algorithm may change during time (e.g., due to same legislative acts).

If the SmartCard generate the last message, means that a successful biometric match has been performed and the *login* is successful. The message 4 containing the nonce and the (hashed) live template is also used to respond to the authentication challenge sent by the Cryptoki in message 3.

4 Towards a Formal Analysis of the MOC-using-TOC Protocol

We may relate the MOC-using-TOC protocol and MOC one, by using two criteria: functional behavior and security features. We may use a variant proposed in [7] of the CCS process algebra of R. Milner to formally describe the behaviour of the components involved in the protocol, i.e. the smart-card (S), the Cryptoki (C) and the match module of the Cryptoki (M).

The functional behavior of the MOC-using-TOC is clearly an extension of the MOC one. Indeed, the steps $3_1 - -3_2$ are added to the MOC ones. The formal specification of the protocol exactly put in evidence this fact. Indeed, the specification of the whole Cryptoki consists of the specification C in parallel with M, i.e. $C \mid M$. Note that this is possible since C and M do not need to communicate each other and they do not need to share the same knowledge to proceed in the computation. Indeed, the criptoki needs to know the PIN (it could ask the user to provide it) while the $\mathsf{Cryptoki}_{\mathsf{match}}$ does not need it. In particular, the $\mathsf{Cryptoki}_{\mathsf{match}}$ could in turn delegate the match to another (remote) module.

Thus, we may expect that future cryptoki applications that will implement MOC may easily adapted from the ones that use MOC-using-TOC. For instance, a MOC-smart-card may decide if performing the steps $3_1 - -3_2$ and so demanding the verification of the match among the $Template_{live}$ and the $Template_{storedAC}$, or to directly perform by itself this match. It may be reasonable to assume that the smart card will ask for either a MOC-using-TOC protocol or a MOC one depending on its internal features. The $\mathsf{Cryptoki}_{\mathsf{match}}$ will act depending on this request. In the case of MOC protocol, is will simply avoid to perform steps $3_1 - -3_2$. This is formally described by the fact that the set of possible execution traces of C is contained in the ones of $C \mid M$.

The security features of the MOC protocols are more interesting. The MOC protocol guarantees a mutual authentication of a smartcard that knows a PIN with a cryptoki that also knows the PIN and has a certificate. We assume that the $Template_{live}$ is acquired by a secure device and so no-one else the identified person is able to provide the cryptoki with the fingerprint. (The $Template_{live}$ could be also acquired by a trusted device from the smart-card.) We also assume that the cryptoki is honest. Then, what it is done is an authentication of the cryptoki to the smartcard followed by the creation of a session key. This session key may be used for secure the following connections.

What we are going to check is that whenever the smartcard authenticates the cryptoki then the cryptoki wanted to be authenticated by the smartcard and the $Template_{live}$ provided by the cryptoki is equal to $Template_{storedAC}$ stored in the smartcard that authenticates the cryptoki and both the smart-card and the

Cryptoki shares the PIN. From a theoretical point of view, the correctness of the protocol depends on the secrecy of the PIN and the private key of the criptoki.

Thus, the security property is a form of agreement on the message sent (the template) among the smartcard and the cryptoki. This may be modeled through common techniques as the correspondence analysis and tools and techniques as [5, 10, 7]. In particular, we follow the approach based on non-interference developed in [7]: The idea is that a system is secure if its observable behavior cannot be significantly altered by an enemy. Clearly, such a definition relies on the notion of *observable* behaviour. Indeed, honest agents in the protocol are able to issue special actions that denote their beliefs on the current status of the computation. For instance, an user A raises an action $start(A, B, m)$ meaning that it is starting a run of the protocol with B for exchanging the message/template m. In the following the observable behavior consists only of this kind of actions.

A way to check that the protocol MOC-using-TOC is correct is to show that MOC is correct and that the security features of the MOC-using-TOC are simulated/equal to the ones of MOC-using-TOC.

The analysis scenario is the following. Consider an external enemy X, described through a process X, which does not hold either the private key of the Cryptoki or the PIN (but could know the $Template_{live}$). Consider also that the Cryptoki issues the control action $start(C, S, Template_{live})$ when it starts a communication with the smart-card for an user that prompts the live template $Template_{live}$; moreover assume that the smart-card issue the action $finish(S, C, Template_{storedAC})$ whenever it considers it just ended a run of the protocol by recognizing that an honest Cryptoki that knows the PIN has provided a live template that matches with the one store in the attribute certificate $Template_{storedAC}$.

Note that MOC protocol is such that we have for each enemy X that satisfies our analysis assumption:

$$(S \mid C \mid X) =_{Obs\_traces} start(C, S, Template_{live}).finish(S, C, Template_{storedAC})$$

where $Template_{live}$ and $Template_{storedAC}$ match.

We may show that when we are interested in such a property then the MOC and MOC-using-TOC agree on their observable behavior (that is to say enjoys the same property):

$$(S \mid C \mid M \mid X) =_{Obs\_traces} (S \mid C \mid X)$$

where Obs_traces denotes the equalities of set of traces where only the set of observable actions is taken into account. This analysis may be defined for each security property of interest.

5 Conclusions

Modern, inexpensive TOC smartcards cannot compute a biometric match as MOC smartcards. We have developed a protocol which simulates the MOC

strategy through the use of TOC cards. In practice the actual match is delegated to the Cryptoki module of the card host. The computational burden that the protocol puts on the smartcards amounts to a classical PIN match.

Our protocol achieves the goal of authenticating a user to a smartcard by means of the user's biometric information, although the smartcard does not perform the match. The design we have presented has been fully implemented using the PKCS#11 standard, which is the most common interface language for current smartcards.

Acknowledgements

We would like to thank the workshop attendees for the exciting discussion environment and, in particular, for pointing out a possible weakness of a previous version of our protocol due to a non appropriate usage of weak secrets.

We are also indebted to Luca Bechelli and Stefano Frassi for valuable suggestions about the PKCS#11 implementation constraints.

References

1. L. Bechelli, S. Bistarelli, and S. Frassi. A protocol for simulating match-on-card authentication through the use of a template-on-card technology. *Working draft.*
2. L. Bechelli, S. Bistarelli, F. Martinelli, M. Petrocchi, and A. Vaccarelli. Integrating biometric techniques with electronic signature for remote authentication. *ERCIM News*, (49), 2002.
3. L. Bechelli, S. Bistarelli, and A. Vaccarelli. Biometrics authentication with smartcard. Technical Report 08-2002, CNR, Istituto di Informatica e Telematica, Pisa, 2002.
4. L. Bechelli, S. Bistarelli, and A. Vaccarelli. Certificati per impronte digitali memorizzati su smartcard. Technical Report 09-2002, CNR, Istituto di Informatica e Telematica, Pisa, 2002. In Italian.
5. G. Bella. Inductive Verification of Smart Card Protocols. *Journal of Computer Security*, (1), 2003. In press.
6. Mike Bond and Piotr Zielinski. Decimalisation table attacks for pin cracking. Technical Report UCAM-CL-TR-560, University of Cambridge, Computer Laboratory, feb 2003.
7. R. Focardi and F. Martinelli. A uniform approach for the definition of security properties. In *Proc. of FM'99*, volume LNCS 1708, pages 794–813. Springer, 1999.
8. N. J. Henderson and P. H. Hartel. Pressure sequence - a novel method of protecting smart cards. In J. Domingo-Ferrer, D. Chan, and A. Watson, editors, *Proceedings 4th Int. IFIP wg 8.8 Conf. Smart card research and advanced application (CARDIS), Bristol, UK*, pages 241–256. Kluwer, 2000.
9. "RSA Security Inc.". Pkcs#11 v2.11: Cryptographic token interface standard. Technical Report PKCS#11 v2.11 r1, RSA Security Inc., nov 2001. RSA Security Inc. Public-Key Cryptographic Standard (PKCS).
10. L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6:85–128, 1998.

11. B. Struif. Use of biometrics for user verification in electronic signature smartcards. In *Smart Card Programming and Security - Proceedings of the International Conference on Research in Smart Cards (E-smart), Cannes, France*, pages 220–228, sep 2001.