Proxy Smart Card Systems*

Giuseppe Cattaneo[1], Pompeo Faruolo[1], Vincenzo Palazzo[2], and Ivan Visconti[1]

[1] Università di Salerno, ITALY {cattaneo,pomfar,visconti}@dia.unisa.it
[2] Bit4ID s.r.l, via Coroglio, 57 BIC Città della Scienza - 80124 Napoli
vpa@bit4id.com

Abstract. The established legal value of digital signatures and the growing availability of identity-based digital services are progressively extending the use of smart cards to all citizens, opening new challenging scenarios. Among them, motivated by concrete applications, secure and practical delegation of digital signatures is becoming more and more critical. Unfortunately, secure delegation systems proposed so far (e.g., proxy signatures) include various drawbacks for any practical system.
In this work we put forth the notion of a "Proxy Smart Card System", a distributed system that allows a smart card owner to delegate part of its computations (e.g., signatures of messages) to remote users. We stress the problematic aspects concerning the use of known proxy-cryptography schemes in synergy with current standard technologies. This in turn motivates the need of proxy smart card systems. Then we formalize the security and functional requirements of a proxy smart card system, identifying the involved parties, the adversary model and the usability properties. Finally, we present the design and analysis of a proxy smart card system which outperforms the current state of the art.

1 Introduction

Proxy cryptography is a widely developed research area that consists in providing cryptographic primitives that allow a user to safely delegate part of its tasks (typically signatures of messages) to another user. Concrete applications of proxy cryptography are becoming more and more critical. For instance digital signatures are now regulated and accepted by law in almost all countries and many entities playing crucial roles in both enterprises (e.g., CEOs) and public institutions (e.g., mayors, rectors), have to sign a large amount of documents per day. Moreover, it is often the case that documents have to be signed urgently, even when the signer is out of his office and unreachable. The possibility of delegating signing privileges should therefore be extended also to *digital* signatures.

Unfortunately we observe a huge gap between the results provided for proxy cryptography and their use in the real world. Indeed, it is well known that results produced by cryptographers need several years to be assessed and then

* This work has been supported in part by the joint project *"SmartSEC"*, with Bit4ID S.r.l., financed by Italian Ministry of Economic Development in the framework P.O.N. 2000-2006 - Misura 2.1.

used by practitioners. Moreover cryptography in stand-alone is not usable, it needs to be integrated in a system with security and privacy mechanisms that can make robust all the involved components. Proxy cryptography is affected by such delays, and indeed, while the literature already gives several provably-secure schemes enjoying many features and reasonable efficiency, almost nothing of it is actually used in the real world. This is in large part a consequence of the long distance between the requirements of proxy cryptography (e.g., system parameters, cryptographic operations) and the currently used technologies (e.g., PKIX [1], Smart Cards). It is therefore urgent to provide mechanisms that allow delegation of signatures using *current* standard technologies *only*.

Our contribution. In this work we study the problematic aspects of using proxy cryptography along with current standard technologies to implement delegation of signatures. Therefore, motivated by the world-wide spread of smart cards (SCs, for short), and their cryptographic operations (e.g., signatures) for implementing various cryptographic services, we put forth the notion of a *Proxy Smart Card System* (PSCS, for short). We investigate concrete real-world scenarios and according to them we formalize the security and functional requirements of a PSCS, identifying the involved parties, the adversary model and the critical usability properties. We finally present the design and analysis of a proxy smart card system based on the use of a network security appliance that outperforms the current state of the art. The development of our system required the combined use of several techniques and technologies in a novel way, which in some case could be also of independent interest.

Our solution is a "ready-to-use" framework that can be easily plugged in real-life scenarios. It does not resort to currently unadopted features of proxy cryptography and instead uses the synergy of existing crypto tools and security technologies to obtain a robust, easy to configure, scalable and cheap system to delegate, under some access control policies, signature privileges.

2 Proxy Signatures

The concept of proxy signature was introduced respectively by Mambo et al. [2] and by Mambo and Okamoto [3]. In such schemes a player called owner O delegates to another player, called user U, the power to execute his own cryptographic tasks. In a proxy signature system, U can sign messages on O's behalf. In general, in such systems, O generates some *proxy secret keys* which Us can use to sign documents verifiable through O's public key.

Originally, these building blocks were considered to be used in large enterprise scenarios, where a manager would like to delegate signature capabilities. Subsequently, the use of such schemes has been suggested in numerous other contexts.

Security requirements. According to the relevant literature [2, 4], and the requirements of real-world applications, a proxy signature schemes should enjoy the following (informal) properties.

`Proxy signature.` *Verifiability*: a verifier always accepts a proxy signature computed by a delegated honest user U; *Strong unforgeability*: it must be computationally hard for a player that is not a delegated honest U to compute a new proxy signature that is accepted by a verifier; *Strong identifiability*: from a proxy signature computed by a delegated user U, it must be possible to determine efficiently the identity of U; *Strong undeniability*: it must be computationally hard for a player that computed a proxy signature, to subsequently repudiate it.

The above properties have been formally defined along with several variations and extensions in the related literature. Here, for lack of space and the sake of focusing the paper on the core of our contribution, we will consider the above informal security requirements only.

Functional requirements. We notice that currently no proxy-cryptography scheme seems to be concretely used in practice. Our investigations about the available schemes, the above security requirements and the available cryptographic tools, raised the following issues. 1) Proxy-cryptography schemes often use number-theoretic constructions and procedures that heavily deviate from the currently available standard technology. Their introduction in real-life scenarios would require too much effort for users to move to new/different systems. 2) Several schemes do not combine gracefully security and flexibility, indeed most of the proposed systems enjoy some given properties and can not be easily adapted to relax some of them. 3) Several schemes suffer of practical limitations.

The work done so far on proxy cryptography mainly focused on the design of powerful cryptographic primitives, but unfortunately it substantially ignored the concrete functional requirements of a practical and easy to use system. In order to be more concrete about such requirements, we studied different contexts where proxy signatures are needed and we collected the *functional requirements* (beyond the usual security requirements) that we believe any practical proxy signature system should enjoy. We summarize those requirements in the following categories. *Compatibility*: schemes should use standard technologies only in order to be compatible with current software applications; *Flexibility*: schemes should allow users to configure and select the appropriate features dynamically. *Efficiency*: schemes should be reliable and satisfy some critical performance requirements.

Motivated by the above requirements, and the problematic use of proxy cryptography for satisfying them, we investigated the possibility of designing a system where all those security and functional requirements could be satisfied simultaneously. In the next section we show the design of our system that thus gives a positive answer to the challenging question of having a viable technology for digital signature delegation.

3 Design of a Proxy Smart Card System

Following the security and functional requirements identified in the previous section, we designed a PSCS, that is, a proxy smart cards system that can be

used to safely delegate signing capabilities of a personal smart card. In our system Os can allow authorized Us to remotely access to their SCs in order to sign messages using their private keys. Notice that smart cards are nowadays a standard technology deployed to all citizens by means of electronic ID cards. Moreover, the use of smart cards guarantees a high level of robustness of the system, thanks to the hardness of extracting private keys (i.e., the device is ideally considered tamper proof). Here we consider SCs as standard PKCS#11 [5] compliant smart cards, where the critical operations are protected by PIN (i.e., personal identification number).

A central role in our PSCS is the Proxy Server P, a hardware/software network security appliance equipped with smart card readers. The purpose of P is to allow Us to use the signing capabilities of SC without compromising any critical information (e.g., private keys, PIN). O shares his SCs by plugging them into readers connected to P, while Us remotely interacts with P to use them according to the role-based access control (in short, RBAC [6]) configured by O. These interactions are implemented by PSCS through a *Remote PKCS#11*, that is, a library that exposes to Us standard PKCS#11 functionalities while the computations are carried out on SCs plugged in P. Using this approach, Us can continue to use their standard applications also on O's SCs to compute proxy signatures.

Making SCs remotely available introduces the problem of filtering remote access to the SCs. This requires the assumption that P is a *tamper proof/evident network security appliance* designed to provide the same services of a local smart card reader through the net.

Remote smart card. The smart cards that P makes available to Us do not necessarily correspond to the smart cards plugged in card readers. Indeed, in our system Os have the possibility to configure SCs in different operating modes giving to Us a virtual view of the SCs available. In detail, Os can define the *Remote Smart Card* (RSC) as *Single* (SRSC) or *Parallel* (PRSC). In the former case, a RSC corresponds exactly to a real SC while in the latter case several SCs, offering the same objects, will appear to Us as a single RSC. A request on a PRSC can be executed indifferently by any SC linked to it. Notice that an O can have several certificates (and thus several public keys) associated to his identity, therefore PRSC is achievable by using a smart card for each certificate, so that each smart card stores a different private key. Indeed, a critical feature concerning the use of smart cards is that the private key should never leave the smart card (and thus cloning procedures should not be performed). Another important requirement is that the associated PIN should never be memorized in permanent storage, and we will deal with this later when we will discuss our PIN management system.

From the above discussion, we have that SCs with different keys can still be used for signatures delegation. The above mechanism makes signature delegation more efficient, indeed, a PRSC allows one to parallelize the load of requests across its SCs.

Set up of the system. All Us and Os must enroll the system by registering their public keys. O plugs his SCs into the smart card readers connected to P. Through a remote administration web interface O sets the configuration of his RSCs and defines the related access policies for the delegated Us. An authorized U for a given RSC, receives a special PIN that does not correspond to the real SC's PIN, but instead is a *virtual PIN* that allows him to access that RSC. We discuss in the next section the problematic issues concerning PIN management, and the technical motivation of our non-trivial solution. Os can revoke the delegated capabilities to each U in any moment by simply updating the access control policies. Such updates have immediate effects, indeed a revoked U will not be able to invoke any further service on P. The past signatures will remain valid. The system allows Os to authorize the delegation only for a given time interval and/or on specific documents. Moreover, O can decide if the proxy signatures will be with or without warranty (in the former case, the signature will contain also a warning about performed delegation).

Proxy signatures. First of all we remark that U can use his standard applications, that are PKCS#11 compatible, to sign documents through the O's SCs. These applications must only set the client side of *Remote PKCS#11* as PKCS#11 layer. This module has the task of interacting with P in order to accomplish remotely the operation invoked by the application. Obviously, this task is done transparently to the application. Its first step is to access to P by means of a strong authorization mechanism (i.e., TLS [7] client authentication through digital certificates). Once the secure channel has been established, according to U privileges, it enumerates to the application all the RSCs available as PKCS#11 slots. When an RSC has been selected by U to sign documents, the client component of *Remote PKCS#11* will sign the request with U's private key and will send it to the server component of the library. This signature is required in order to log on P the request, that thus can not be repudiated by U. If the PIN is correct and U has the required privileges, the operation is executed by the selected SC and the result is sent back to local component of *Remote PKCS#11* that will forward it to the application. More specifically, the system will dispatch the requests on a PRSC to the first available SC linked to that PRSC through a Round Robin scheme that therefore will balance the load of requests. Since the sign functions are slow and long term operations, this mechanism radically improves system performance linearly scaling with the number of SCs configured for the PRSC. The system allows obviously Os access (even remotely) to all the logs, in order to let them monitor completely the activity of their delegates.

Security model. Given the critical use of smart cards in real world scenarios, a security model is required in order to show that a proposal is resilient to attacks mounted by malicious players. First of all, we follow the standard approach that assumes that an adversary has complete control over the communication channel. This includes the capability of reading and updating all messages that cross the network, of delaying the delivering of messages, and so on.

We assume that P is a trusted player, this means that when it is active it follows the prescribed procedures and his behavior can not be compromised. This assumption is both 1) necessary, and 2) achievable in practice. Indeed, in case P is under the control of an adversary, since SCs are plugged into its readers, and remotely accessed through its software, the adversary would obtain the PINs of the SCs and thus could also ask them non-authorized services (e.g., signatures). Notice that while it is known how to design protocols that are secure even in presence of such adversaries, the known solutions require that honest players (in this case SCs and honest Us) perform computations that go much beyond the simple PKCS#11 interface that is currently available for accessing to standard smart cards. The need of obtaining a proxy system on top of standard technologies, therefore requires that P behaves honestly. Honest behavior can moreover be enforced by using some run-time integrity check techniques, as proposed in [8, 9], and by integrating the support of smart cards directly in the kernel of the operating system, as proposed in [10].

The above assumption about P is also achievable in practice since the hardware infrastructure of P can be placed into a restricted access area (basically implementing a tamper evident mechanism) and moreover his software could be placed in EEROM (i.e., Electrically Erasable Read-Only Memory). Therefore the software is rewritable only when a special password is known. There must be instead a read-write (RW, for short) memory that will contain for instance log files and the RBAC policy files. We do not assume special requirements about such an RW memory, indeed its content remain valid and used by P as long as there is a valid message authentication code (MAC, for short) associated to them. Indeed, this memory could be adversarially corrupted and we require that the adversary must not be able to produce new valid data. Moreover, erasing such data or trying to restore previous data will have no (substantial) effect since P is assumed to periodically send through S/MIME [11] encrypted and signed backups of such data to the addresses associated to Os.

We assume that *qualified* Us are honest while other Us can be corrupted. The distinction between such two categories depends on the RBAC policies configured for each smart card. Us that can access to services provided by some SCs are assumed to be honest for those SCs and potentially dishonest for the remaining services of those SCs and for the other SCs. Notice that since RBAC policies are dynamic, the set of qualified users is dynamic as well, and thus a user can be considered honest only temporarily (therefore one can not simply assume that the owner of a SC gives the PIN to qualified Us). All honestly produced SCs are assumed to be incorruptible, instead an adversary can produce some non-legitimate SCs that can be plugged into the readers of P and Us.

Pin management. A major requirement for the design of a proxy smart-card system is the transparent use of remote smart cards as they were local. Indeed, clients would like to recycle their applications that access to local smart in order to also access to the remote smart cards connected to the proxy smart-card system. Notice that access to a smart card is possible through a log on procedure where a personal identification number (PIN) has to be provided by the user and

sent to the smart card. The need of recycling standard applications implies that one can not simply assume that qualified users are identified by the system through passwords. This restriction is enforced could be enforced by laws that mandatory require the use of PINs for accessing smart cards. Moreover, after a prescribed number of PIN log on failures a Personal Unblocking Key (PUK) is needed to restore access to the smart card.

The above problem could in general be solved by the following trivial solution: the PIN of the smart card is communicated to all users that have sufficient privileges to access the smart card. This solution however does not satisfy the flexibility requirement of a proxy smart-card system since user privileges are in general dynamic and thus removing a user from the system would require the generation of new PINs that then should be distributed to all qualified users. This is clearly unacceptable in systems with many users and dynamic assignment of privileges. We have therefore developed a more sophisticated system.

The failure of the trivial solution discussed above implies that the PIN on the client's side must be different from the real PIN that allows one to succeed in the log on procedure with the smart card. It is therefore fundamental to establish a virtual PIN system where users know some virtual PINs that can be translated into real PINs by the proxy smart-card system. In this direction one can consider the following simple but conceptually wrong solution. The RBAC policy is encoded through a table where each U has associated a mapping between virtual PIN and real PIN. Therefore, upon receiving a remote log on request with a given virtual PIN, P simply accesses the table and translates the virtual PIN to a real PIN to be used for the log on procedure with the smart card. This procedure would match the flexibility requirement of the system. However, it still includes a security drawback that we want to exclude from our architecture. Indeed, the above table should be stored somewhere in the permanent memory of P and would include the real PIN. Storing a real PIN on a permanent memory is conceptually wrong and in contrast with the common philosophy about the correct use of smart cards. Taking into account these issues, our solution is more sophisticated and requires the use of the virtual PIN as a key for the symmetric encryption of the real PIN. Therefore, when a new virtual PIN is generated and associated to a real PIN, P will be updated by adding a new entry in an access control table and it will contain an encryption of the real PIN computed by means of the virtual PIN as key. When U accesses remotely to a SC, he has to send the virtual PIN that then will be used by P to decrypt the corresponding entry in the RBAC table and to perform the log on procedure on the SC. Notice that using this approach we can still have flexibility and at the same time no key or PIN is stored unencrypted in the permanent memory of P.

Implementation details. In this section we illustrate the main implementation details of our PSCS. First of all we implemented our PSCS using a Client/Server schema, between the PKCS#11 local component (on client side) and the PKCS#11 engine (on server side). The first one exposes a standard PKCS#11 interface to U's local application, but when the applications invoke its functions, the module remotely calls the corresponding engine function on P. Invocations are encap-

sulated in a proprietary format and sent using the HTTP protocol through a secure channel (HTTPS) with mutual authentication based on the exchange of X.509 certificates [1]. The server engine forwards the requests to the plugged SCs and returns to the client the results. In the standard PKCS#11 interface some functions must be coded by the library and some others must be executed natively by SC. Some computations (e.g., AES symmetric encryptions, cryptographic hashing), are executed locally by the client module while others (e.g., signatures) by SC on P through the engine component. We stress that this mechanism is transparent to Us and requires only the availability of U's authentication capabilities in the standard PKIX [1] setting.

4 Conclusion

We have conducted several performance measurements with different use cases. In all of them, our system resulted sufficiently practical, flexible, efficient and secure as no other currently available proposal in the literature. Our system is also easy to set up and we expect that our work will also give a chance for further extensions and improvements, thus generating follow up research on this topic. Further details about the implementation and the security of our system will be shown in the full version of this paper.

References

1. Housley, R., Ford, W., Polk, W., Solo, D.: Internet X.509 Public Key Infrastructure Certificate and CRL Profile (1999)
2. Mambo, M., Usuda, K., Okamoto, E.: Proxy signatures for delegating signing operation. In: ACM Conference on Computer and Communications Security. (1996) 48–57
3. Mambo, M., Okamoto, E.: Proxy cryptosystem: delegation of the power to decrypt ciphertexts. In: IEICE Trans. Fundamentals E80-A(1). (1997) 54–63
4. Lee, B., Kim, H., Kim, K.: Strong proxy signature and its applications. In: SCIS. (2001) 603–608
5. RSA Laboratories: PKCS #11: Cryptographic Token Interface Standard. http://www.rsa.com/rsalabs/node.asp?id=2133
6. Ferraiolo, D.F., Kuhn, D.R.: Role based access control. In: 15th National Computer Security Conference. (1992) 554–563
7. Network Working Group : The Transport Layer Security (TLS) Protocol Version 1.2. http://tools.ietf.org/html/rfc5246 (2008)
8. Catuogno, L., Visconti, I.: A Format-Independent Architecture for Run-Time Integrity Checking of Executable Code. In: SCN 2002, (2003): Vol. 2576 of Lecture Notes in Comuter Science, 219-233, Springer.
9. Catuogno, L., Visconti, I.: An Architecture for Kernel-Level Verification of Executables at Run Time. In: Comput. J. 47(5): 511-526 (2004).
10. Catuogno, L., Gassirà, R., Masullo, M., Visconti, I.: Securing Operating System Services Based on Smart Cards. In: TrustBus 2005, (2005): Vol. 3592 of Lecture Notes in Comuter Science, 321–330, Springer.
11. Network Working Group : S/MIME Version 3 Message Specification. http://tools.ietf.org/html/rfc2633 (1999)