Proxy Smart Card Systems *Revision* : 1.19

paper identification number: SEC-114

ABSTRACT

In this work we put forth the notion of "Proxy Smart Card System", a distributed system that allows a smart card owner to delegate part of its computations (e.g., decryptions and signatures of messages) to remote users. Such digital transactions are nowadays more and more critical because of the established legal value of digital signatures and of the continuous increment of the available identity-based digital services.

Our contribution is three fold. 1) We first stress the problematic aspects concerning the use of proxy cryptography in synergy with current standard technologies. This motivates the use of new proxy systems. 2) 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. 3) Finally, we present the design and analysis of a proxy smart card system which outperforms the current state of the art.

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 and decryption privileges.

Keywords: smart card, proxy cryptography.

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 decryptions and signatures of messages) to another user. Concrete applications of proxy cryptography are nowadays 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. Another major example is the increasing use of encryption features for e-mails, in order to keep private some relevant data. Again, one would like to delegate to someone else the capability of decrypting some of the emails (e.g., the ones with a specific subject) in order to reduce his own amount of work and not to stop his activities when he is disconnected from the Internet.

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 used by practitioners. Moreover cryptography in standalone 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, indeed, while the literature already gives several provably-secure schemes with several features and reasonable efficiency, almost nothing of it is actually used in the real world. One of the reasons is that there still is a solid barrier between the requirements of proxy cryptography (e.g., system parameters, cryptographic operations) and the currently used technologies (e.g., PKIX, smart cards). Since this barrier will very likely take long time to be removed, it is therefore urgent to provide mechanisms that allow at least part of the features of proxy cryptography, to be used in the real world using current standard technologies.

Our contribution. In this work we study the problematic aspects of introducing the ideas of proxy cryptography in current standard technologies. Motivated by the world-wide spread of smart cards (SCs), and their cryptographic operations (e.g., signatures and decryptions) for implementing various cryptographic services, we put forth the notion of a *Proxy Smart Card System* (PSCS). We investigate the 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.

Our solution is a "ready-to-use" framework that can be immediately activated in real-life scenarios. It does not resort to the 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 proxy smart card system.

2. PROXY CRYPTOGRAPHY

The concepts of proxy signatures and proxy encryptions were introduced respectively by Mambo et al.[15] and by Mambo and Okamoto[14]. In such schemes a player called owner O delegates to another player, called user U, the power to execute his own cryptographic tasks.vIn a proxy signature system, U can sign messages on O's behalf, while in a proxy encryption system he can decrypt ciphertexts encrypted under O's public key. The key-idea of such systems is that O can generate some *proxy secret keys* which Us can use to sign documents verifiable through O's public key and decrypt ciphertexts encrypted under O's public key.

In literature many generalizations and extensions have been proposed in the past, such as threshold proxy signatures[18], blind proxy signatures [12, 17, 9], proxy signatures with warrant recovery [1], nominative proxy signatures [16], one-time proxy signatures [13], and proxy-anonymous signatures [19, 7, 8, 20, 21, 5, 3]. Originally, these building blocks were considered to be used in large enterprise scenarios, where a manager would like to delegate signature capabilities or he could delegate someone else to decrypt some messages (e.g., e-mails) encrypted with his public key. Subsequently, the use of such schemes has been suggested in numerous other contexts as, mobile agent environment [10], grid computing [4], distributed shared object systems [11], global distribution networks [2], and mobile communications [16].

Security requirements. According to the relevant literature, a proxy signature scheme should enjoy the following properties. 1) **Verifiability**: a verifier always accepts a proxy signature computed by a honest user U; 2) **Strong Unforgeability**: it must be computationally hard for a player that is not a honest to compute a new proxy signature that is accepted by a verifier; 3) **Strong Identifiability**: from a proxy signature computed by a user U, it must be possible determine efficiently the identity of U; 4) **Strong Undeniability**: it must be computationally hard for a player who computed a proxy signature, to subsequently repudiate it.

A proxy encryption scheme should also enjoy the following properties. 1) **Correctness**: a honest user U always correctly decrypts an encryption of a message under O's public key; 2) **Indistinguishability**: it is computationally hard for a player that is not O and neither U to distinguish the plaintext encrypted in a ciphertext w.r.t. any other possible plaintext.

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

Functional requirements. We notice that currently no proxy-cryptography scheme seems to be largely 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 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 enjoys some given properties and can not be easily adapted to relax some of them properties. For instance, in some concrete applications no O would use a system where Us can repudiate their proxy signatures. Moreover, no Us would use a system where O could make a proxy signature on his behalf. 3) Several schemes suffer of practical limitations (e.g., an efficient revocation mechanism, mechanisms to filter the type of document that can signed or decrypted, tools for monitoring Us activities).

Summing up, 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 and encryptions are needed an we collected the *functional requirements* (beyond the usual security requirements), that we believe any practical proxy signature/encryption system should enjoy. We summarize those requirements in the following categories. 1) **Compatibility**: schemes should use standard technologies only in order to be compatible with current software application; 2) **Robustness**: schemes should be secure and reliable: 3) **Flexibility**: schemes should allow users to configure and select the appropriate features dynamically; 4) **Efficiency**: schemes should satisfy some critical performance requirements.

3. DESIGN AND IMPLEMENTATION OF AN USABLE PSCS

Following the security and functional requirements identified in the previous section, we designed a PSCS, that is, a proxy cryptography system based on smart cards. In our system Os can allow authorized Us to remotely access to their SCs in order to sign and decrypt 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 assure a high level of robustness of the system, thanks to the hardness of extracting private key (i.e. the device is considered tamper proof). SCs are a standard PKCS#11-compliant smart card, where the private key operation are protected by PIN.

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 make sign or decrypt operation through the O's SC without compromise any critical information like private key or PIN. O share his

SCs by plugging them into the readers connected to P , while Us remotely interacts with P to use them according to the role-based access control (in short, RBAC) configured by O. These interactions are implemented by the PSCS through a sort of *Remote PKCS#11*, that is, a library that exposes to Us standard PKCS#11 functionalities while the operation are carried out on SCs plugged in P. Using this approach, Us can continue to use his standard applications also on O's SCs to compute proxy signatures or to decrypt messages.

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. Our system also includes a special player, referred to as system manager SM, with the role of to manage system configurations, particularly keeping the system on-line when Os leave their smart cards in P and the system has to be functional also over reboots. Notice that when Os prefer to have the system online only when they are physically close to P, then SM does not need to exist.

Remote Smart Card. SCs exposed by P to Us are not exactly those 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 a **Single Remote** SRSC or a **Parallel Remote** PRSC. In the former case, the RSC is exactly 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. The above mechanism makes the system more efficient, indeed, a PRSC allows 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. Then through a remote administration web interface he sets the configuration of his RSCs and defines the related access policies for the delegated Us. When U is authorized to use a RSC, he will receive a PIN that is not the real SC's PIN, but it is a *virtual PIN* needed to enabling him to operate on that RSC. We discuss in details in the next section the problematic issues concerning PIN management, and we will present our 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 period and/or on specific documents (e.g., decryption only of e-mails with a given subject). 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 the delegation).

Proxy signatures/decryption. U first gets access to RSC by means of a strong authorization mechanism (i.e. TLS client authentication through digital certificates). Us, by means of standard applications, can sign or decrypt documents through O's SC. According to the U privileges the application will enumerate all the RSCs available as PKCS#11 slots. U selects a RSC and invoke the sign/decrypt operation, providing the virtual PIN when asked. The client component will sign the request with the U's private key and will sent it. This will be logged by P, so that it cannot be repudiated anymore. If the PIN is correct and U has the required privileges, the operation is executed by the related SC and the result is sent back to U. The system allows Os access to logs database in order to check completely the activity of their delegates. If the request affects a PRSC, the system will dispatch the request to the first available SC linked to that PRSC through a Round Robin scheme that is able to balance the load of requests. Since the sign/decrypt functions are long term operations, this mechanism radically improves system performance linearly scaling with the number of SCs configured for the PRSC.

3.1 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 played, 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, decryptions). 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 in standard smart cards.

The above assumption about P is also achievable in practice since the hardware infrastructure of P can be placed into a special hard to access area (basically implementing a tamper evident mechanism) and moreover his software could be placed in ROM (e.g., a CD physically hardwired in the device with a tamper evident mechanism). There must be instead a read-write (RW, in 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 all such files remain valid and used by P as long as there is a valid message authentication code associated to them. Moreover, erasing such data or trying to restore previous data will have no effect since P is assumed to periodically send through S/MIME [**?**] encrypted and signed backups of those files 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 dishonest for the remaining 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. Finally, we assume that each O is honest only with respect to his SCs.

3.2 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 cards so that they can also be used to access 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 by recent laws that mandatory require the use of PINs for accessing smart cards. Moreover, after a prescribed number of PIN log on failures a 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's 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.

Virtual PINs. 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. Indeed, remember that even though P uses RBAC policies, a corrupted U can try to remove a card from the system while it is off-line and then knowledge of the PIN would allow him to obtain illegitimate signatures. 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 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 translate 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 of architecture. Indeed, the above table should be stored somewhere in the permanent memory of P and would include the real PIN. A physical attack to P when it is off-line, with the aim of reading the content of this disk, would give to the adversary knowledge of both virtual and real PINs. The problem is that it can directly use the smart cards through the real PINs in case he will be able to physically extract them from the reader.

Encrypted virtual PINs. The need of protecting the system from the above attack implies the use of an encryption mechanism so the a virtual PIN sent by the client is then translated into a real PIN without storing such PINs in the permanent memory of P. P could therefore store the above table in an encrypted format. The decryption key would usually be in the memory of P and could be loaded either automatically during the boot or by an activation procedure where the key is inserted manually. Notice that the former case requires the secret key to be stored on the permanent memory and thus does not prevent the physical attack discussed above. The latter approach instead could be managed by using the special player SM previously mentioned in the description of the our system. However, notice that in case SM is corrupt still the table with all real pins would be exposed, cards can be removed and used somewhere else. We have therefore developed a more advanced scheme that goes beyond such limitations.

3.2.1 PIN Encryption Scheme.

We show now our solution for managing the translation of a virtual PIN into a real PIN without significantly decreasing the security and the flexibility of the system, thus improving the benefits of the mechanisms discussed above.

We use 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 the user accesses remotely the smart card, 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 smart card. 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 thus preventing the above physical attacks when P is off-line.

However, there still use a subtle security problem. Indeed, since PINs are elements of a relatively small space (e.g., 10^8) one could mount a brute force attack against the access control table in order to decrypt the real PIN. Indeed, using a standard encryption scheme as AES256, the decryption under a key k of a 256-bit ciphertext will result in a 256-bit plaintext that very likely will be greater than 10^8 when k is the wrong key and instead will be less than 10^8 when k is the correct key. Therefore, with a feasible (just 10^8 iterations) brute force attack, one can still obtain the real PIN (assuming that by physically accessing Pthe adversary also obtains the table).

It is therefore critical that each ciphertext when decrypted with any possible virtual PIN, still produces a well-formed PIN that the adversary should try in the log on procedure. In this case, after 3 wrong trials the smart card becomes unusable and the adversary is stuck.

We therefore propose the following scheme: the virtual PIN is first extended to 256-bit long string by means of SHA256. Then the real PIN is extended to a 256 bit by randomly padding the remaining $(256-27)$ bits (10^8 requires 27 bits). Then AES256 is computed using the extended virtual PIN as key and the extended real PIN as message. In this case, a brute force attack always produces 256-bit long plaintexts that can be reduced considering the last 27 bits to a candidate PIN. Notice that if the first bit of those 27 bits is 0, then the resulting number is always a valid candidate PIN. Instead, if the first bit is 1 the value can go beyond 10^8, and in this case the adversary could discard this value. Summing up, trying the 10^8 possible virtual PINs, the adversary will have at least 2^{26} candidate PINs, which is currently considered a satisfying space with respect to the only 3 chances that the adversary could have.

Finally, we remark that still in case U knows a virtual PIN and has physical access to P, it can obtain the real PIN. Therefore, we suggest to use the above PIN encryption scheme along with the use of a special activation key provided by SM in order to decrypt the table. More formally, each entry of the table will be encrypted twice, once with the key given by SM during the boot of P (we stress again that SM can be removed when the system does not need to be on-line when Os are away) and once with the virtual PIN. As long as there is no coalition between SM and U the system will result to be secure. We therefore require Os not to fail twice giving privileges (i.e., activation keys to SM, delegation of SCs to Us) to corrupted players.

3.3 Implementation Details

In this section we provide the reader with the main implementation details of the prototype of the PSCS we just presented. First of all we implemented our PSCS using a Client/Server schema, respectively between the PKCS#11 local component and the PPKCS#11 engine. The first one exposes a standard PKCS#11 interface to the U local application, but when the applications invoke its functions, the module remotely calls the corresponding engine function on P. Invocations are encapsulated in a proprietary format and sent using the HTTP protocol through a secure IP channel (HTTPS) with mutual authentication based on X.509 certificates exchange. The server engine forwards the requests to the plugged SCs and returning 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 the SC. Also in the case of our library some operations, for instance AES symmetric encryption or the hash operations, are executed locally by the client module and other, for instance sign/decrypt, by the SC on Pthrough the engine component. We stress that this mechanism is transparent to Us and requires only the availability of U certificate installed in his machine.

On the server side, to speed up the prototype realization we used a standard high level application server called Twisted [**?**] coding all the P modules using the language python [**?**] while the client component has been written in C language. Free BSD 4.4 has been chosen as the host operating system including some security features for certificate management and the secure network layer.

As showed in the next picture, the overall architecture of PSCS is set up by 4 modules: Policy Manager, Log Service, *RequestHandler* and the *CardHandler*. The functions implmented by the first two are straightforward: the former module implements the RBAC policies management while the latter module provides high level API to log all system activities along with the certification of the log file integrity.

The module *RequestHandler* listens to U's requests. Its main tasks consists parsing the request, verifying U's credentials according to the RBAC policies and, maintaining the sessions information for each Us. Whenever Uowns all the privileges required by the operation he asked, the request is forwarded to the corresponding CardHandler, pushing it into its requests' queue. Obviously RequestHandler maps the requests trough a dictionary with the running configuration that is the list of the available objects, updating the structure each time a new SC is inserted or removed. The PSCS administrator through a web interface manages the possible options changing the configuration of available cards (e.g., choosing if they should be used in SRSCor PRSC operating mode).

For each physical card reader available to the PSCS there is an instance of the module *CardHandler*, which runs as a separate process, that handles the SC. Then it monitors card status changes (insertion, removal or failure) and, if the SC is ready, it picks up the next operation from its queue and invokes the corresponding low level card primitive. Card operations are synchronous with respect to the source queue, but may be executed concurrently with respect to the involved cards. Whenever during the processing phase an error is raised, the error messages is forwarded to the requestor (client PKCS#11 module) in order to let the application to report the error message. When the result is available, the RequestHandler is notified and it is supposed to format a reply message that is sent back to the client.

4. ANALYSIS OF PSCS

We now show that the above system satisfies all the security and critical requirements discussed in previous Sections.

4.1 Security Analysis

Verifiability. In our system a signature is generated by using the private key of O. Therefore the canonical verification procedure of the signature scheme can successfully assess the validity of the signature.

Strong unforgeability. Assume that an unauthorized signature is obtained by an adversary. Notice that since the signing keys are in the tamper proof area of the smart card, such a signature must have been produced by the smart card after a successful log on procedure, which requires the use of a correct PIN. Since in our security model we assume that

the system is physically unbreakable while it is on-line, we can therefore distinguish two cases: 1) the adversary has obtained the signature accessing the system remotely while it is on-line; 2) the adversary has obtained the signature while the system was off-line. The former case, has negligible probability to happen since P follows the RBAC policies and therefore gives back a signature computed by SC only if U has the corresponding rights. Therefore it must be the case that the adversary impersonated U. This however can not happen as long as we U has not been corrupted and thus his smart card and computing resources have not been violated. The use of TLS and PKIX prevents further impersonations. The case in which U has been corrupted instead does not correspond to an unauthorized signature, since U becomes the adversary and thus it obtained a legitimate signature. The latter case, consists instead in physically attacking P. Notice that the adversary can extract SC from the reader but still he needs a valid PIN to access to it. By also tampering P he can read the RBAC database and obtain the encryptions of the PIN but still he is not able to decrypt them, since virtual PINs are needed for the decryption. Obtaining a virtual PIN requires either hacking the TLS protocol (since virtual PIN are always sent by a legitimate U to P in a TLS-encrypted channel) and corrupting SM or corrupting U and SM. While the former is assumed to be infeasible by the security of the TLS protocol, the latter is assumed to be impossible in our security model as a coalition of SM and U is assumed not to be possible. We stress that such a coalition does not make any sense when SM corresponds to O, moreover when SM is another entity, his power has been directly obtained from O to make services available when O has not access to P.

Strong undeniability. In our system when U requests the signature of a message m under O's public key, it has to sign such a request using his personal smart card, and such a signature is sent to P which then verifies it before sending a signature request for m to the smart card corresponding to O. It turns out therefore that as long as U' local computing system is not under attack, all his signature request stored by P correspond actually to request sent by U.

Strong identifiability. The actual message that is signed by SC is a pair `msg`$= (m, id)$ where m is the message requested by U and id is the identity of U that has also been logged in P along with the signature request. Therefore each signed message uniquely and securely identifies U.

Correctness. The correctness of an encryption scheme is immediately translated in the correctness of the proxy system since SC will be able to decrypt a message upon receiving a request of P on behalf of a legitimate U.

Indistinguishability. The semantic security of the underlying encryption scheme ensures that an encrypted message does not disclose any information about the plaintext as long as the decryption key is not available to an adversary. However, the decryption is stored in the tamper proof area of the smart card, therefore in order to be able to decrypt message without a legitimate authorization, an adversary has to violate either the RBAC policies while the system is on-line or should physically attack the system while it is off-line. The analysis here continue almost verbatim as we have already discussed for strong unforgeability. Further details are therefore omitted.

4.2 Usability and Performance Analysis

Compatibility. Compatibility with standard software is a critical requirements to obtain a ready to use system. Therefore P must be addressed as a local device (SC reader) through a special PKCS#11 driver (library) which routes the application requests to the remote appliance through the local network connection. This should be transparent to any application designed to operate through PKCS#11 functions. No change or rebuild should be necessary to applications when using the appliance services. This is necessary to preserve the certification process of the applications. Moreover, signatures produced by Us are verifiable with standard procedures since they are actually computed by SC. Moreover, notice that source codes of such application are not commonly available, therefore we do not modify anything else including the user interface, (e.g., the PIN management) thus preserving the laws compliance of the application. More in details, to enforce legal compliance, especially with respect to the PIN management, the appliance should exhibit the same behavior of a local SC blocking the card (service for the user) after 3 PIN log on failures.

Robustness. Beyond what we have discussed so far there are several other requirements depending on the fact that the legitimate owner leaves her SC connected to the network appliance, to keep alive the proxy system. Indeed both honest and malicious Us could try to use the system, basically obtaining digital signatures/decryptions of documents/ciphertexts. Therefore, it is is necessary to enforce strong authentication and tracking mechanisms to identify users, to manage privileges and to meaningfully log all the operations performed by P with each SC for a given U. As for any network device, P could be attacked (even by authorized Us), and this should not be turned in a general DoS attack but only the specific SC should stop its activities for that user. Since SCs are remote with respect to Us, hardware failures should be considered with more attention. An operation on a SC object occasionally can fail due to several reasons (e.g., electrical failures, unusual temperature), but that SC could become again available. Therefore the network appliance should enforce a stronger policy on the failure management, introducing a retry mechanism with exponential delay up to some bound. SC are accessible only after user login, but actually client applications are connected through stateless sessions. This means that SCs sessions have to be managed by P considering that the login operation is time consuming, while the network connections with the client applications should be considered stateless and memoryless.

Flexibility. Through the RBAC policies Os can allow access to their SCs in very different way, such as, for a fixed time, only in the office time or according to the type of document to be signed/decrypted. The system allows O to configure the proxy signature service with or without the warranty about the identity of U. Moreover, adding a new SC should make automatically available the resource to all clients even if they are already connected to the appliance, and no reboot should be required. On the other hand, since client applications are stateless, removing a card at run time could be an issue to consider. In any case before unmounting a card the related queue should be empty. Finally the network appliance should be provided with an html-based management module which will enable the administrator to manage the SC set, the various configuration, logging, update, and to perform all the basic administrative tasks (e.g., upgrade, new module installation, power up, shutdown).

Efficiency. The system was designed to manage elevate rate of requests. Oblivious the bottleneck is represented by SCoperations that is very slow with respect to the request handling. In order to improve the system performance Os can use several SCs in the PRSC configuration so that several requests can be served in parallel. The system adopt a Round Robin scheme to dispatch the requests to the first available reader, providing the maximum concurrency degree with respect to the number of SC available (i.e., multithreading environment).

5. 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 system currently available in the literature. Given that our system is also cheap to set up and easy to install, we expect that our work will also give a chance for further extensions and improvements, thus generating follow up research on this topic.

6. REFERENCES

[1] Awasthi, A. K., and Lal, S. Proxy blind signature scheme. Cryptology ePrint Archive, Report 2003/072, 2003. http://eprint.iacr.org/.

[2] Bakker, A., van Steen, M., and Tanenbaum, A. S. A law-abiding peer-to-peer network for free-software distribution. In *NCA* (2001), IEEE Computer Society, pp. 60–67.

[3] Fan, C., Zhou, S., and Li, F. Deniable proxy-anonymous signatures. In *ICYCS* (2008), IEEE Computer Society, pp. 2131–2136.

[4] Foster, I. T., Kesselman, C., Tsudik, G., and Tuecke, S. A security architecture for computational grids. In *ACM Conference on Computer and Communications Security* (1998), pp. 83–92.

[5] Fuchsbauer, G., and Pointcheval, D. Anonymous proxy signatures. In *SCN* (2008), pp. 201–217.

[6] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X509 public key infrastructure: Certificate and Certificate Revocation List (CRL) Profile. Network Working Group, RFC 3280, April 2002.

[7] Hu, C., and Li, D. A new type of proxy ring signature scheme with revocable anonymity. In *SNPD (1)* (2007), pp. 866–868.

[8] Hu, C., Liu, P., and Li, D. A new type of proxy ring signature scheme with revocable anonymity and no info leaked. In *MCAM* (2007), pp. 262–266.

[9] Kim, Y. S., and Chang, J. H. Provably secure proxy blind signature scheme. In *ISM* (2006), IEEE Computer Society, pp. 998–1003.

[10] Lee, B., Kim, H., and Kim, K. Secure mobile agent using strong non-designated proxy signature. In *ACISP* (2001), V. Varadharajan and Y. Mu, Eds., vol. 2119 of *Lecture Notes in Computer Science*, Springer, p. 474.

[11] Leiwo, J., Hänle, C., Homburg, P., and Tanenbaum, A. S. Disallowing unauthorized state changes of distributed shared objects. In *SEC* (2000), S. Qing and J. H. P. Eloff, Eds., vol. 175 of *IFIP Conference Proceedings*, Kluwer, pp. 381–390.

[12] Liu, W., Tong, F., Luo, Y., and Zhang, F. A proxy blind signature scheme based on elliptic curve with proxy revocation. In *SNPD (1)* (2007), pp. 99–104.

[13] Lu, R., Cao, Z., and Dong, X. Efficient id-based one-time proxy signature and its application in e-cheque. In *CANS* (2006), D. Pointcheval, Y. Mu, and K. Chen, Eds., vol. 4301 of *Lecture Notes in Computer Science*, Springer, pp. 153–167.

[14] Mambo, M., and Okamoto, E. Proxy cryptosystem: delegation of the power to decrypt ciphertexts. In *IEICE Trans. Fundamentals E80-A(1)* (1997), pp. 54–63.

[15] Mambo, M., Usuda, K., and Okamoto, E. Proxy signatures for delegating signing operation. In *ACM Conference on Computer and Communications Security* (1996), pp. 48–57.

[16] Park, H. U., and Lee, I. Y. A digital nominative proxy signature scheme for mobile communication. In *ICICS* (2001), S. Qing, T. Okamoto, and J. Zhou, Eds., vol. 2229 of *Lecture Notes in Computer Science*, Springer, pp. 451–455.

[17] Qin, Y., and Wu, X. Cryptanalysis and improvement of two blind proxy signature schemes. In *CSSE (3)* (2008), pp. 762–765.

[18] Shao, J., Cao, Z., and Lu, R. Improvement of yang et al.'s threshold proxy signature scheme. *Journal of Systems and Software 80*, 2 (2007), 172–177.

[19] Yumin, Y. A threshold proxy signature scheme with nonrepudiation and anonymity. In *ISCIS* (2006), A. Levi, E. Savas, H. Yenigün, S. Balcisoy, and Y. Saygin, Eds., vol. 4263 of *Lecture Notes in Computer Science*, Springer, pp. 1002–1010.

[20] Zhao, Z., Tang, X., Li, B., and Zhu, L. An id-based anonymous proxy signature from bilinear pairings. In *Security and Management* (2006), H. R. Arabnia and S. Aissi, Eds., CSREA Press, pp. 138–144.

[21] Zhou, X., and Wei, P. Anonymous proxy authorization signature scheme with forward security. In *CSSE (3)* (2008), pp. 872–875.