An Efficient Software Protection Scheme

Antonio Maña, Ernesto Pimentel
Computer Science Department
University of Málaga
29071 – Málaga SPAIN
e-mail: amg@lcc.uma.es

Key words: Software protection, smart cards, cryptography, information commerce.

Abstract: Software piracy has been considered one of the biggest problems of this industry since computers became popular. Solutions for this problem based in tamperproof hardware tokens have been introduced in the literature. All these solutions depend on two premises: (a) the physical security of the tamperproof device and (b) the difficulty to analyze and modify the software in order to bypass the check of the presence of the token. The experience demonstrates that the first premise is reasonable (and inevitable). The second one, however, is not realistic because the analysis of the executable code is always possible. Moreover, the techniques used to obstruct the analysis are not helpful to discourage an attacker with usual resources. This paper presents a robust software protection scheme based in the use of smart cards and cryptographic techniques. The security of this new scheme is only dependent on the first premise because code analysis and modification are not useful to break this scheme.

1. INTRODUCTION

Software protection is a complex problem; consequently there are several fields of research concerning different aspects of the problem. Some of the most important goals related to software protection are:

- *Intellectual property protection*. The objective is to link the software with information about it's author. Among the

techniques used for this purpose the most popular is watermarking [CoTh99].

- *Protection against function analysis in mobile environments*. The objective in this case is to prevent a malicious host from discovering the purpose of a software agent and modify its behavior. Techniques like code obfuscation or function hiding [LoMo99] are used, sometimes complemented by the use of hardware tokens [Fünf99].
- *Protection against illegal copy and use of software*. The objective is to guarantee that only authorized users can run the software. Our work is mainly aimed to solve this problem.

Every year software industry has to face a cost of several billion dollars due to software piracy. In 1999, the global piracy rate for PC business software applications was 36 percent with an estimate cost of $12 billion. As soon as computers started to became popular unauthorized copying of software started to be considered an important problem [Kent80]. Development of computer communications brought the growth of BBS services distributing pirated software. Today, other circumstances like the advances in code analysis tools and the popularity of Internet creates new opportunities to steal software. Some of the money lost because of the software piracy is included in the cost of legal software and therefore pirate copies are partially paid by the legal users.

Most of the software that is produced today has either weak protection mechanisms (serial numbers, user/password, etc.) or no protection mechanisms at all. This lack of protection is essentially derived from the user resistance to accept protection mechanisms that are inconvenient and inefficient. In Bruce Schneier words: “The problem with bad security is that it looks just like good security”. Many commercial software protection tools claim to achieve total security with software techniques. Most of these tools are *snake oil*[1]. Theoretic approaches to the formalization of the problem have demonstrated that a solution that is exclusively based in software is unfeasible [Gold97].

On the other side, legal protection tools like *trade secrets*, *copyright*, *patents* and *trademaks*, are not adapted for the protection of software. Some authors have proposed the creation of new specific legal protection means for software products [Samu95].

[1] Taken from the Snake-Oil FAQ: The term is used in many fields to denote something sold without consideration of its quality or its ability to fulfil its vendor's claims. This term originally applied to elixirs sold in travelling medicine shows. The salesmen would claim their elixir would cure just about any ailment that a potential customer could have. Listening to the claims made by some crypto vendors, “snake oil”' is a surprisingly apt name.

An important related aspect is license management, that has to be capable of covering a wide range of situations and conditions while being easy and convenient for the final user.

Based on some advances of the general information security technology, we have developed a low cost software protection and license management scheme that is secure, flexible and convenient for the users. This scheme, avoids two of the most common attacks to software protection mechanisms: multiple installation from a single legal license and production of unprotected (pirated) copies of the software.

The rest of the paper is organized as follows. Section 2 reviews the most relevant related work. Section 3 introduces the new scheme. In section 4 we analyze implementation details. Other applications of this scheme are presented in section 5 and finally, section 6 summarizes the conclusions and presents ongoing research and future work.

2. RELATED WORK

In this section we will briefly review some proposals for software protection and license management, considering aspects like security, convenience and practical applicability.

One of the simplest and most popular protection mechanisms consists in a password or key check that enables installation of the software. If the check fails the software is not installed or it works in demo mode with restricted functionality. This mechanism is very popular in *shareware*. The password (or key) validation function is, evidently, included in the software. Therefore, it is possible to find it using reverse engineering. As a consequence it is frequent that key generation programs are produced by dishonest users and also that authentic passwords are published in certain Internet sites.

Sometimes the software is personalized to be used in one computer, for example, extracting information from some of the hardware devices (hard disk, network adapter, etc.) or from the operating system configuration. During its execution, the protected software checks that the computer is the one it was personalized for. This check, as the previous ones, can be bypassed. Also, this mechanism is inconvenient for the users because changes in the hardware or in the operating system may result in the need to get a new license and reinstall the software.

Self modifying code, and code obfuscation [CoTh00] are used in some software protection schemes. These techniques provide short term protection and can be used in situations where software life is short (for example for

agents and applets). Some of these techniques have been developed for a very special kind of software: virus [FHS97].

A very interesting approach is represented by function hiding techniques. In [SaTs98] the authors present an scheme that allows evaluation of encrypted functions. The idea is to establish an homomorphism between the space of cleartext data and the space of data enciphered by some cryptosystem. The objective is to evaluate some function on some data without revealing them. This process can be expressed this way: Let P be the domain of cleartext data and Q the domain of encrypted data. Let $f : P \rightarrow P$ be a function that the user wants to evaluate on some $x \in P$, and let $e : P \rightarrow Q$ and $d : Q \rightarrow P$ be respectively the encryption and decryption functions of some cryptosystem. Then, under certain conditions on the original function *f*, it is possible to find a function $f' : Q \rightarrow Q$ such that $\forall x \in P \quad f(e(x)) = e(f(x))$ or, using an alternative of the previous expression $\forall x \in P \quad d(f(e(x))) = f(x)$. This property is useful because it allows a piece of software to store $e(x)$ and implement f' in order to compute $f(x)$ from $e(x)$ without revealing *f*, *x* or *f(x)*. Unfortunately this property only holds for certain families of functions (polynomial functions in this case).

Among the proposed solutions that rely on some hardware component, one of the most popular consists in the use of hardware tokens that are difficult to duplicate, which are connected through some communications port to the computer running the software. The protected software checks the presence of the token and refuses to run if the check fails. Examples of this kind of systems are hardware keys or dongles. These systems usually have the problem of the incompatibility between tokens of different applications. When the tokens are smart cards, as it is expected that the computer will include just one card reader, the user must continuously change the card, a problem known as *card juggling* that represents a serious inconvenience.

The check of the presence can be done in different ways; the simplest is to read a value from the communications port, but, commonly, to avoid that the interception of the communication in that port allows the attacker to replicate the token, the software will send a value (called challenge) that the token has to process, the software can predict the result that the token must send back. In any case, whatever the check is, it is not hard to bypass this protection, as the access to the communications port or the reader are easily found in the executable code. The check can then be bypassed obtaining a completely functional copy of the software as figure 1 shows. This process can even be automated by specially designed programs called “patches”.

Sometimes the software is distributed encrypted and the token is used to decrypt it before it runs on the computer. The problem is that, when the software is decrypted, it is stored in the RAM memory of the user’s (and

potential pirate) computer. There are different techniques that the pirate can use then to recover the software (for example producing a core dump).

One of the first proposals to use smart cards for software protection is presented in [ScPi84]. Protective technologies commercializes a tool that is based in those ideas and that share certain similarities with the initial scheme presented in the introduction of the section 3.

More recently, Aura and Gollman presented in [AuGo99] an interesting scheme based on smart cards and digital certificates that solves the card juggling problem and provides mechanisms for license management and transfer. In addition, a compilation of countermeasures against attacks are reviewed. Unfortunately, as their proposal is focused on the check of the presence of the smart card, it is vulnerable to the code modification attacks described above.

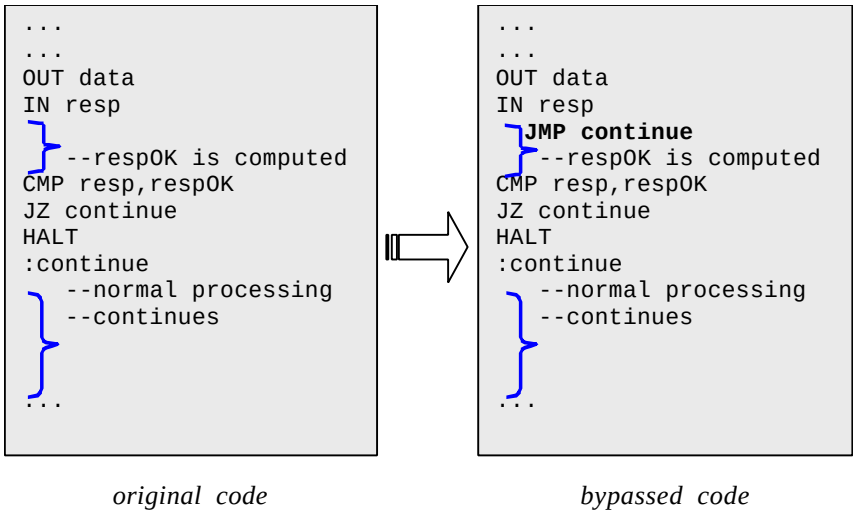

Fig. 1. Code modification to bypass the check of the presence of the token.

From the study of the problem it is concluded that to obtain a provable secure protection scheme we must have a tamperproof processor that contains and executes the protected software [HePi87]. A variation of this scheme is the distribution of encrypted code that the tamperproof processor decrypts and executes [Be94].

3. DESCRIPTION OF THE NEW SOFTWARE PROTECTION SCHEME

As it is usual in other fields of information security, in software protection there are no completely secure solutions. The objective of a software protection scheme is to make the attack to the scheme difficult enough to discourage dishonest users.

The new scheme is based, as others, in a tamperproof processor. The popularization of smart cards and their evolution in storage and processing capacity have lead us to consider them the most appropriate choice for our scheme. However, our design does not depend on this technology and, consequently, our solution can be implemented using any similar hardware token (for example, some hardware keys and some tokens that integrate smart card and reader functionalities).

A secure software protection scheme can be designed using just smart card technology. In this scheme some sections of the software to be protected can be substituted by functionally equivalent sections to be processed in the smart card. In this way, the protected software is divided and will not work unless it cooperates with the right card. Code modification attacks will not succeed in this case. In fact, the only possible attack is to analyze the data transmitted to and from the card trying to guess the functions that the card performs. If we include enough functions, with enough importance in the main code, and enough complexity, the attack described could become impractical.

This scheme needs one card per application and the quantity and complexity of the protected functions are limited by the capacity of the card. Moreover, this scheme does not allow the distribution of the protected software using Internet because the cards must be distributed with the software. With the purpose of avoiding the aforementioned problems we will introduce the cryptography as the second building block of our software protection scheme.

3.1 Fundamentals of the new scheme

Figure 2 shows the first scheme that we elaborated. We will use it to illustrate the final scheme. The figure shows that several sections of the original code are substituted by their equivalent for the card during the production phase. These new sections are encrypted with the public key of the card using an asymmetric cryptosystem [RSA78] during the personalization phase and are kept encrypted so only the card that has the matching private key will be able to decrypt and execute those protected sections. The cards now have to store a key pair, but the protected software

sections do not reside on the cards. The key pair must be generated in the card and the private key must never be transmitted outside the card. The original code sections are substituted by calls to a function that transmits their equivalent protected sections (e.g. **"B"**), including code and data, to the card, where they are decrypted and executed. When finished, the card sends back the results.

Assuming that the encryption algorithm is secure, the attack to the system must be based in the analysis of the input and output data (and possibly the running time) of the card functions. However, we must emphasize that now the card only stores one function at a time and therefore we can use more complex functions because all the capacity of the card is now available for each single function. Moreover, this scheme allows the card to execute any number of protected functions. The dishonest user will need to discover all of the protected functions to be able to break this protection scheme.

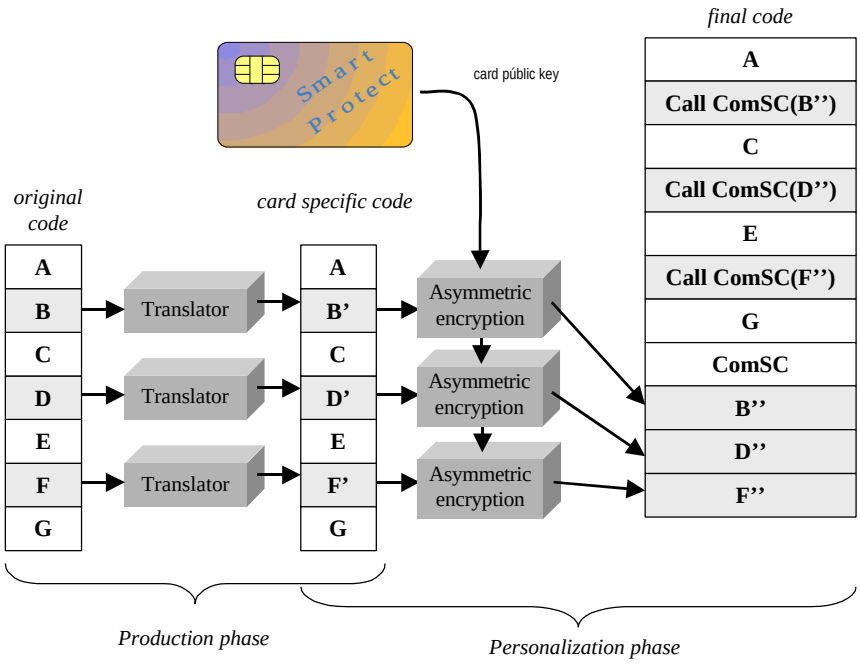

Fig. 2. Code transform in our first software protection scheme.

An alternative attack could consist in the substitution of some of the authentic protected sections by other fake sections produced by the dishonest user (for example such a false section could try to send back the contents of the card). This attack can be considered a kind of "Trojan horse". To avoid these attacks we must authenticate the code before its execution [DDB89].

To summarize, this first scheme allows a single card to be used to protect many applications, increases the complexity of the protected functions, allows the card to execute any number of those functions and enables the distribution of the software through Internet.

But, in spite of the advantages mentioned, some aspects like efficiency and robustness of the scheme need to be improved. The use of an asymmetric cryptosystem introduces a high computational cost. Also the lack of a code authentication mechanism opens a dangerous attack line. On the other hand, this first scheme does not take into account some desirable features like license transfer or expressive authorization. Also, the need to include a personalization phase is not adequate for some distribution models. We want the software to be freely distributed, although to run it the user will need to get a license.

The final scheme is shown in figure 3. In this case the production phase includes the encryption of the protected sections (wich include code and data) with a symmetric cryptosystem.

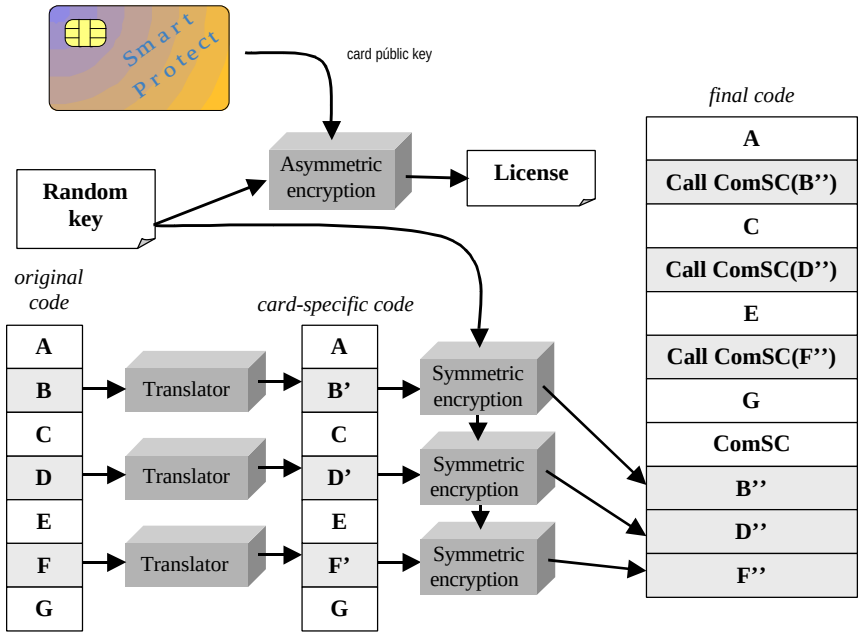

Fig. 3. Code transform and license production.

In the authorization phase (equivalent to the personalization phase of the previous scheme), a new license is produced containing the random symmetric key used to encrypt the protected sections, information about conditions of use (i.e. time limits, number of executions, etc.), the

identification of the software (ID, version number, etc.) and finally the identification of the license. All this information is encrypted with the card public key. When the license is received by the client it is stored in the card.

The functionality of the previous scheme is maintained in this new one, but the efficiency is improved because decryption of the protected sections is now much faster. The definition of the license structure permits a high degree of flexibility. Furthermore, as each application has its own key, we can manage them individually.

We previously mentioned the necessity to authenticate the code to be executed by the card to avoid certain attacks. In this scheme, because the protected sections are encrypted using a symmetric key that is kept inside the card, it is impossible for a dishonest user to produce false sections. However, if the license was to be transmitted using an insecure channel, a man-in-the-middle attack could be carried out, but as we will show in the next section, the software producer will require a certificate of the card public key that the dishonest user will not be able to forge.

3.2 License management

3.2.1 Sale

Because the license for the user (containing the key to decrypt the protected sections) is encrypted with the card public key, it is essential to avoid that the corresponding private key is known outside the card. To achieve this objective the most practical solution is to use special smart cards produced for this purpose. These cards will contain a key pair and some support software. A certificate of the public key of the card is signed by the card manufacturer to guarantee the authenticity of the keys.

To buy a protected application, the client sends a request containing the certificate of the public key of his card and a random number to the software producer. The producer verifies the validity of the certificate and, in case the validation succeeds, produces a new license, encrypts the license and the random number using the public key received and sends it to the client card. The card verifies that the license matches the request (i.e. the random number is correct) and stores it. The producer also stores all the licenses in a database to be able to generate new licenses for the client when needed (theft, destruction of the card, etc.). If a request for an already generated license is received, the producer will prepare a new license for the client with no extra cost. This new license will include a different serial number (this number is part of the identification of the license). The software application is distributed and copied freely, with no additional protection.

3.2.2 Transfer

One of the features that we have considered important (introduced in [AuGo99]) is license transfer. License transfer could be used to delegate the right to use some software application to another user or simply to store your license in a new card. Our scheme introduces the possibility of selective license transfer.

Our license transfer scheme has been designed to avoid using certificate chains because of the overhead in communication, storage and processing they introduce. Another goal was to avoid storing public keys of external entities in the smart cards.

The protocol to transfer a license is divided in two phases: *delegation* (steps 1 to 3) and *recover* (steps 4 to 6). We call this protocol *direct transfer* opposed to the *scheduled transfer* which is used mainly for recovery purposes. The protocol is as follows:

1. The user selects which license (or licenses) are going to be transferred from the source card. Notice that, opposite to other systems, our scheme does not oblige the user to transfer all the licenses in the source card (which we consider to be a serious limitation). In the rest of this protocol we will assume that we are transferring one specific license.
2. The public key certificate of the destination card is sent to the source card.
3. The source card creates a certificate delegating the license to the public key of the destination card, destroys its own license and finally sends the delegation certificate to the destination card.
4. The destination card requests a new license to the software producer. This request includes the delegation certificate received from the source card and the destination card public key certificate.
5. The software producer verifies both certificates and generates a new license for the destination card if the verification succeeds. The license database is updated accordingly.
6. The destination card decrypts and stores the new license.

Suppose now, that the protocol described above is interrupted (accidentally or intentionally to attack the protection scheme). For instance, if the protocol is aborted after step 3, the destination card would possess the delegation certificate but not the new license. The source card has already destroyed its license but it can request a new copy from the software producer and get a new valid license. Afterwards, using the delegation certificate that has stored, the destination card can also get a new license. This attack could be used to replicate any number of licenses.

To prevent this attack, a serial number, different for each new copy of the license produced, is included in the license (see section 3.2.1). In the scenario depicted above, when the source card requests the new copy after aborting the protocol, the software producer generates a new license (with a different serial number) that is sent to the card and stored in the database. Later, when the destination card attempts to use the delegation certificate to get a new license, the request will be denied.

If the protocol is aborted during the step 3 (for instance, extracting the card from the reader) it may occur that the source card have destroyed its license and the delegation certificate has not been sent to the destination card. In this situation the source card can still request a new license.

The inclusion of the software producer in the transfer protocol may seem inconvenient but if the producer is not included, the source card would need to verify the public key certificate of the destination card which, in turn, would increase the complexity of the protocol and also would introduce weaknesses in the protection scheme.

3.2.3 Recovery

Providing efficient and convenient solutions to the problems that the protection scheme may introduce is considered very important for user acceptance. In any scheme that uses some kind of hardware components it is essential to prevent the consequences of failure in those components. In our scheme licenses are linked to smart cards based on the fact that the private key is not known outside the card. Consequently, in case of card failure it will be impossible to run the software. For this contingency, the user must take some prevention measures. As the price of the cards is small, it seems reasonable to prepare a replacement card to be used in case of failure of the main card. The preventive process requires the execution of the delegation phase of the scheduled transfer protocol for all the licenses in the card. In case of failure of the main card, the protocol would continue on the recover phase. At the end of the protocol the replacement card will possess the same licenses as the main card.

The difference between the direct transfer protocol and the scheduled transfer protocol is the inclusion of the date (or other parameter like number of executions) when the transfer will take place. This date is included in the delegation certificate. Steps 3 and 4 of the direct transfer protocol are replaced by this sequence in the scheduled transfer protocol:

3'. The source card creates a certificate delegating the license to the public key of the destination card on date *D* and sends it to the destination card. The source card will not delegate that license again to any other card until date *D*.

4'. Later two different situations can arise:
- If the user wants to keep using the main card the replacement card must destroy the delegation certificate and send a new scheduled transfer request before date *D*. In this case the source card will accept the request.
- Otherwise, on date *D*:
  - Source card will destroy its own license.
  - As in the direct transfer case, both cards can request a new license to the software producer but only the first will be accepted.

3.2.4 License expiration

The licenses are always kept protected because they are either encrypted or stored in the smart card. Therefore, the card software, which is trustworthy, can destroy licenses when they expire (we can use different parameters like number of executions, time of use, etc.), the software can even warn the user when the expiration is about to happen. One of the parameters most used in software licenses is the expiration date. To include this parameter it would be interesting to have an internal clock in the cards. Some manufacturers have announced cards including this feature.

4. IMPLEMENTATION DETAILS

Today, smart card technology offers features that not so many years ago corresponded to personal computers [CDHP00]. However, compared to the processing power of the host computers, each access to the smart card introduces important delays. As our scheme requires the transmission of a considerable amount of code and data to and from the card, it is important to take into consideration the efficiency of the protection scheme.

The amount of data and code transmitted determines the magnitude of the delay introduced. On the other side, since the main attack to the protection scheme is based in the analysis of the functions performed by the card, the protection scheme will be more secure as the functions grow in size and complexity.

Consequently, it is necessary to find a balance between security and speed. Fortunately, in this case, this balance is possible and it is not difficult to obtain security and speed measures that satisfy both the software producer and the client. A detailed description and study of the efficiency of the protection scheme is included in [LMP00].

The scheme has been designed and the tests carried out using smart cards with symmetric and asymmetric cryptographic capabilities. An implementation that uses smart cards that have only symmetric cryptographic capabilities is possible, but the changes that need to be introduced in the scheme, together with the low prices of the cards with both types of cryptosystems, do not justify the use of cheaper cards.

4.1 Functions executed by the smart cards

This is an essential characteristic because the security of the system is based on the difficulty to guess the functions that the card executes from the analysis of the input and output data and the execution time [Hohl98].

If we know that the function performed by the card represents a straight line then we just need to run the function two times with different input data to discover it. In contrast, functions like one-way hashes [Pren00] or digital signatures [RSA78] are not vulnerable to these attacks. In most software applications this type of functions is not used frequently, but the functions that appear in most software applications have an advantage: they have more input and output data.

To make it difficult for the pirate to analyze the functions we include false (dummy) input and output data that are not used for the computation of the function, although it is transformed to confuse the attacker. Another technique that is very effective to obstruct the analysis is to mix the processing of several functions with the intention that the result of each call to the card depends on the input data of the previous calls and even on results of previous calls that have not been send back as results but stored in the card memory.

4.2 Card readers

One of the most common kind of software piracy takes place inside the organization of a legal client of the software by the use of multiple copies of a legally acquired software application. In our scheme this attack could be carried out making several computers share a card reader.

This problem has been considered in previous schemes, but the most common solution is to make the software have direct access to the card reader. This solution introduces countless problems and computational costs in the protected software because it must manage different situations and hardware features that are usually managed by the operating system.

In our scheme, to prevent this attack we have designed a solution based on the last technique described in section 4.1. The system “chains” the calls

to the card so any incorrect sequence of calls (produced if several computers share a card reader) will result in the software producing erroneous results.

5. OTHER APPLICATIONS

The scheme introduced can be useful in other environments, in fact it was devised from a previous work on information commerce over Internet [Mana00]. As an example of the different possibilities of this scheme we will explain briefly how it can be used for information commerce in applications like online newspapers [Const97] or digital libraries [KLK97].

For this application each user must possess a special smart card (with a key pair and our base software), a card reader and a web browser that can access the card (i.e. with a special *plug-in*).

To gain access to some information the client sends a request to the information provider, including the public key certificate of the client's card. This step might implicate some negotiation of the conditions of the trade. The information provider, using the applet generator described in [Mana00] generates a specific applet to fulfill the request and a license for the client's card. This applet includes protected sections that have to be executed by the card using the license. Because the card software is trustworthy we are able to control aspects like number of executions and, what is more, we can include an electronic purse to pay for the information accessed.

6. CONCLUSIONS AND FURTHER WORK

We have described a robust software protection scheme based in the use of smart cards and cryptographic techniques. Related schemes based in tamperproof hardware tokens that have been proposed in the literature have been analyzed concluding that all of them are based in the check of the presence of the token and are therefore vulnerable to code modification attacks. Considering that the new scheme is not based in that check, code modification is not a potential attack. We have shown the different protocols for the management of licenses and analyzed the security of the scheme and the importance of the implementation details. Finally, we have also introduced possible alternative applications of the scheme. Hence, we can conclude that the advantages of the presented scheme are robustness against different attacks (bypassing the check, code substitution and attacks to the license management protocols), confidence for the user, efficient use of the computational resources of the smart cards, free distribution and copy of the

software, selective license transfer, control of the expiration of the licenses and applicability in distributed computing environments.

Tools to produce protected software automatically from unprotected executable programs, applet protection and payment integration are under development. We are studying the possibilities that the combination of function hiding techniques with our scheme could open.

Finally we are studying the security achieved by the different families of functions that can be executed in the cards to obtain a measure of the protection achieved in some particular software application.

REFERENCES

[AuGo99] Aura, T.; Gollman, D. *Software License Management with Smart Cards*. Proceedings of the Usenix Workshop on Smartcard Technology (Smartcard'99), pp. 75-86. 1999.

[Be94] Bennet S. Yee. *Using Secure Coprocessors*. PhD thesis CMU-CS-94-149, Carnegie Mellon University, 1994.

[CDHP00] Castellá-Roca, J.; Domingo-Ferrer, J.; Herrera-Joancomartí, J.; Planes, J. *A Performance Comparison of Java Cards for Micropayment Implementation*. Proccedings of CARDIS'2000, pp 19-38. Kluwer Academic Publishers. 2000.

[Cons97] Constantas, D. et al. *An Architecture for Electronic Document Commerce*. 4th CaberNet Radicals Workshop, 1997. Available online at http://www.newcastle.research.ec.org/cabernet/research/radicals/1997/papers/edc-constanta.html

[CoTh00] Collberg, C.; Thomborson, C. *Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection*. University of Auckland Technical Report #170. Available online at http://www.cs.auckland.ac.nz/~collberg/Research/Publications/CollbergThomborson2000a/index.html. 2000.

[CoTh99] Collberg, C.; Thomborson, C. *Software watermarking: Models and dynamic embeddings.* Proceedings of POPL'99 - 26th ACM Symposium on Principles of Programming Languages. 1999. Available online at http://www.cs.arizona.edu/~collberg/Research/Publications/CollbergThomborson99a/index.html. 1999.

[DDB89] Davida, G. I.; Desmedt, Y.; Blaze, M. J. *Defending Systems Against Viruses Through Cryptographic Authentication*. Proceedings of IEEE 1989 Symposium on Security and Privacy, pp 312-318. 1989.

[FHS97] Forrest, S.; Hofmeyr, S.; Somayaji, A. *Computer immunology.* Communications of the ACM, Vol. 40, No. 10, pp. 88-96. 1997.

[Fünf99] Fünfrocken, S. *Protecting Mobile Web-Commerce Agents with Smartcards* Proceedings of ASA/MA'99. 1999.

[Gold97] O. Goldreich, *Towards a theory of software protection*, Proc. 19th Ann. ACM Symp. on Theory of Computing, pp. 182-194. 1987.

[HePi87] Herzberg, A.; Pinter, S. S. *Public Protection of Software.* ACM Transactions on Computer Systems, 5(4)-87, pp. 371-393. 1987.

[Hohl98] Hohl F. *Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts*. in Giovanni Vigna (Ed.), Mobile Agent Security, LNCS 1420 Springer Verlag, pp 91-113. 1998.

[Kent80] Kent, S. *Protecting Externally Supplied Software in Small Computers*. PhD thesis, Massachusetts Institute of Technology, MIT/LCS/TR-255, MIT. 1980.

[KLK97] Kohl, U.; Lotspiech, J.; Kaplan M. A. *Safeguarding Digital library Contents and Users: Protecting Documents Rather Than Channels*. D-Lib Magazine, Sept-97 ISSN 1082-9873. 1997.

[LoMo99] Loureiro, S.; Molva, R. *Function hiding based on error correcting codes.* Proceedings of Cyptec'99 - International Workshop on Cryptographic techniques and Electronic Commerce. 1999.

[Mana00] Maña, A. *Una Solución Segura Basada en Java para la Comercialización de Contenidos Digitales*. (in spanish). Proceedings of the Sixth Spanish Conference on Cryptography and Information Security. Ra-Ma, isbn 84-7897-431-8, pp-243-252. 2000.

[LMP00] López, J.; Maña, A; Pimentel, P. *Un Esquema Eficiente de Protección de Software Basado en Tarjetas Inteligentes*. Technical Report 14/2000, Department of Computer Science, University of Malaga. 2000

[Pren00] Preneel, B. *El Estado de las Funciones Hash*. (in spanish). Proceedings of the Sixth Spanish Conference on Cryptography and Information Security. Ra-Ma, isbn 84-7897-431-8, pp-3-38. 2000.

[RSA78] Rivest, R. L.; Shamir, A.; Adleman, L. M. *A method for obtaining digital signatures and public-key cryptosystems.* Journal of the ACM, 21(2):120-126, February 1978.

[Samu95] Samuelson, P. *A Manifesto Concerning the Legal Protection of Computer Programs: Why Existing Laws Fail To Provide Adequate Protection*. Proceedings of KnowRight '95, pp 105-115. 1995.

[SaTs98] Sander, T.; Tschudin C.F. *On Software Protection via Function Hiding*. Proceedings of Information Hiding '98. Springer-Verlag. LNCS 1525. pp 111-123. 1998.

[ScPi84] Schaumüller-Bichl1, I.; Piller, E. *A Method of Software Protection Based on the Use of Smart Cards and Cryptographic Techniques*. Proceedings of Eurocrypt'84. Springer-Verlag. LNCS 0209, pp. 446-454. 1984.