# Protected Computing vs. Trusted Computing*

Antonio Maña, Antonio Muñoz
Computer Science Department
University of Málaga, 29071 Málaga, Spain

***Abstract.*** **The Trusted Computing initiative represents an important effort towards more secure and dependable computers and therefore towards the security of all computer-supported infrastructures and processes. However, the Trusted Computing paradigm entails some risks and drawbacks that are by no means negligible. In this paper we introduce the concept of Protected Computing as an alternative to Trusted Computing and compare these approaches from the points of view of security, usability, applications and convenience for users. Finally we also show how Protected Computing can represent an interesting complement to Trusted Computing.**

## I. INTRODUCTION

The pervasiveness of communication infrastructures along with their ever-increasing bandwidth have fostered the current trend towards distributed computing, giving rise to new paradigms such as *grid computing* or *service-oriented computing*. On the other hand, the advances in microelectronics have originated not only important enhancements in the capabilities of existing devices such as mobile phones and PDAs, but also the embedding of small independent computers with advanced communication capabilities into every imaginable object. This second fact has, in turn, introduced the need for new computing paradigms designed to take advantage of their highly distributed nature. Among these paradigms, we can mention *pervasive computing* and *ambient intelligence*.

All these new paradigms are receiving an important attention from the scientific community, but there is a crucial issue that needs to be solved before they can be widely applied in practice: security.

It is commonly accepted that one of the main reasons that make the provision of real-world security for computing systems so hard is the lack of mechanisms for secure computing. In fact, the ability to perform computations as intended by the software producer and to hide the internal functioning of software has been identified as the key to other security solutions [1].

There are important problems in information security that have been proved to be impossible to solve with software-based solutions. Among these, we find problems such as copy protection, watermarking, obfuscation, production of digital signatures, etc. [2-5]. For other problems, such as auditability, anonymity, or fair exchange, existing cryptographic solutions are very complex and inefficient. However, those problems would have trivial and fast solutions if we could assume (i) that certain actions are protected in such a way that guarantees that they are executed as desired by the software producer and (ii) that the function performed can not be determined by an external entity.

Therefore, for all these problems, solutions must be based in the use of a "trusted element" that can perform the protected actions [1]. Tamperproof hardware devices and external entities (usually known as trusted third parties) are the most frequent "trusted elements".

Although required, the use of trusted elements is not sufficient to guarantee a satisfactory solution. In the optimal solution the amount of trust in these elements must be minimal, while the amount of protection is maximized. The problem is that, usually, these two criteria are conflicting and we need to find a balance among them.

In order to increase the level of trust that users are willing to place on these elements, it is possible to obtain independent certifications of the behavior of trusted hardware, especially in the case of simpler hardware. This certification is not possible in the case of trusted third parties (TTPs), which therefore require a higher amount of trust and are not able to provide high levels of protection. Consequently, our preferred alternative for this purpose is the use of tamperproof hardware elements. Additional reasons for this claim are:

1. TTPs are intrinsically not multipurpose (there are specific TTPs for specific problems). Therefore every user has to deal with multiple TTPs, which complicates the trust and certification schemes.
2. Many environments and applications require that the trusted element performs certain actions in representation of the user. TTPs require more trust because they could potentially use the knowledge of these actions and the data involved in them for illicit purposes.
3. TTPs do not provide better protection than tamperproof hardware because, after all, they must use computing systems, which are usually the easiest target for attacks.
4. TTPs require an online connection that introduces performance, security and availability problems. For many applications this requirement is a serious inconvenient. TTPs cannot be used in offline applications.

Among the schemes that use a tamperproof hardware element, the Trusted Computing[1] initiative represents an important effort towards more secure and dependable computers and therefore towards the security of all computer-supported infrastructures and processes. However, the Trusted Computing paradigm entails some risks and drawbacks that are by no means negligible.

The basic idea behind the concept of Trusted Computing is the creation of a chain of trust between all elements in the

* Work partially supported by the E.U. through project IST-506926

[1] The Trusted Computing Group (TCG) is the new name of the Trusted Computing Platform Alliance (TCPA), which is related to products such as LaGrande, TrustZone, Presidio, NGSCB, Longhorn, Palladium, etc.

0-7803-9575-1/06/$20.00 ©2006 IEEE

computing system, starting from the most basic ones. Therefore, the chain starts with a tamperproof hardware device, known as Trusted Platform Module (TPM), which analyses the BIOS of the computer and, in case it is recognized as trusted, passes control to it. This process is repeated for the master boot record, the OS loader, the OS, the hardware devices and finally the applications. In a Trusted Computing scenario a trusted application runs exclusively on top of protected and pre-approved supporting software and hardware.

Trusted Computing has been publicized as the only possible solution for the protection of computing platforms, but this claim is not substantiated, as demonstrated by the existence of the Protected Computing concept. Protected Computing represents a new alternative and, at the same time, an important complement to Trusted Computing. The most relevant feature of Protected Computing is that it makes possible that a piece of software is securely executed by directly interacting with a tamperproof hardware device. In a Protected Computing scenario, a piece of software can be protected without the need to protect the underlying software or hardware. On the technical side, Protected Computing is based on the partitioning of the software elements into two parts. One of them is executed in a secure processor, while the other part can be executed in a normal processor. For this approach to be realized in real world a secure tamperproof coprocessor capable of executing code "on the fly" is required. This coprocessor must be fast enough (both in processing and communication) and must have some specific features that have already been identified. The results of our experiments using USB-enabled smart cards as secure coprocessors show the practical usability of the scheme [6].

In this paper we introduce the concept of Protected Computing as an alternative and a complement to Trusted Computing. Technically, both schemes are independent, compatible and complementary. Protected computing can take advantage of the existence of the Trusted Platform Module, which is part of the Protected Computing infrastructure, in order to enable features that are not supported by the Trusted Computing scheme, such as enhanced user and application control or dynamic attestation.

The rest of the paper is organized as follows. Section II provides a background on different software protection approaches. Sections III and IV describe the Trusted Computing and the Protected Computing schemes respectively. Section IV presents a comparison of both schemes. Section V describes some possible ways of combining the two schemes. Finally Section VI presents conclusions and describes ongoing and future work.

## II. SOFTWARE PROTECTION BACKGROUND

Several mechanisms for secure code execution, and their properties have been proposed in the literature. A classification of the different approaches to the software protection problem is presented in this section. We focus on security, convenience and practical applicability. More extensive reviews of the state of the art in software protection can be found in [1, 7].

Some protection mechanisms are oriented to the protection of the computer system against malicious software. Among these, SandBoxing is a popular technique to create a secure execution environment that should be used to run non-trusted code. A sandbox is a container that limits, or reduces, the level of access its applications have, and controls the interaction between them. SandBoxing has been an important technique in research since Butler Lampson in his paper entitled "Protection", back in 1971, defined the antecedents of the SandBoxing technique [8].

Proof Carrying Code, refers to a general mechanism to verify that a software element can be executed in a host system in a secure way. This strategy requires that every code fragment is associated to a detailed description on how security policy is satisfied. The host just needs to verify that the description is correct and the code fits properly. This strategy shares some similarities with Constraint Programming. Both are based on a code that is able to perform a limited set of operations. Furthermore both Proof Carrying Code and Constraint Programming have several problems mainly due to the process of determining the set of operations permitted. Furthermore, in many cases, it is difficult to determine restrictions that preserve the semantic integrity. There are variants of this strategy, such as Proof-referencing-code that do not carry code proofs explicitly [9].

Other mechanisms are oriented to defend software against malicious servers. A example of such mechanism is the concept of Sanctuary [10] for agents. A Sanctuary is a site where a mobile agent can be securely executed. An important precedent, although not directly related with software are IBM's Cryptolopes [11, 12] and Intertust's Rights System Platform [13].

Several techniques can be applied to software in order to verify self-integrity. Anti-tamper techniques, such as encryption, checksumming, anti-debugging, anti-emulation and some others [7, 14] are in this category. Some schemes are based on self-modifying code, and code obfuscation [15]. A related approach is represented by software watermarking techniques [16, 8]. In this case the purpose of protection is not to avoid analysis but to detect whether the software has been copied or modified. The relation between these techniques is strong. In fact, it has been demonstrated that neither perfect obfuscation nor perfect watermark exists [17]. All of the latter techniques provide short-term protection; therefore, they are not applicable for our purposes.

Many protection systems are based on checks. In these systems the software includes "checks" to test whether certain conditions are met. We can distinguish solutions based exclusively on software, and others that require some hardware component. However, in both types of schemes, the validation function is included into software. Therefore, reverse engineering and other techniques can be used to discover it. Theoretic approaches to the formalization of the problem have demonstrated that a solution exclusively based on software is unfeasible [18]. By extension, all autonomous protection techniques are also insecure.

In some scenarios, such as agent-based ones, the protection required is limited to some parts of the software (code or data). In this way, the function performed by the software, or the data processed, are hidden from the host where the software is running. An external offline processing step may

be necessary to obtain the desired results. Among these schemes, the most interesting approach is represented by function hiding techniques. Reference [19] describes a scheme that allows evaluation of encrypted functions. The fundamental idea is to establish an homomorphism between the spaces of plaintext and encrypted data, with the objective of evaluating a certain function on some data without revealing them.

The case of online collaboration schemes is also interesting. In these schemes, part of the functionality of the software is executed in one or more external computers. The security of this approach depends on the impossibility for each part to identify the function performed by the others. This approach can be appropriate for distributed computing architectures such as agent-based systems or grid computing, but presents the important disadvantage of the impossibility of application to off-line environments

Finally, there are techniques that provide protection for both the platform and the applications, such as the Trusted Computing Platform. Therefore, this approach consists of an enhancement to the platform, which includes a trusted component, frequently built-in hardware, which is used to create a foundation of trust for software processes [20]. An early member of this category is the ABYSS architecture [21]. In ABYS, some processes of the software to be protected are substituted by functionally equivalent processes, which run inside a secure coprocessor. Processes are encrypted while outside of the secure coprocessor. Finally, the SmartProt mechanism is the foundation of the Protected Computing paradigm and is based on the division of an application´s code between a trusted and an untrusted processor in such a way that is not possible to run the application without the collaboration of the trusted processor [6].

## III. TRUSTED COMPUTING

The idea behind the Trusted Computing paradigm was introduced in 1997 by Arbaugh, Farber and Smith [22]. The technology currently known as Trusted Computing has been developed by the Trusted Computing Group (TCG) on the basis of the specifications developed by the Trusted Computing Platform Alliance (TCPA). According to their documentation [23] "The distinguishing feature of TCG technology is arguably the incorporation of 'roots of trust' into computer platforms." At first sight it may appear strange that the concept of "root of trust" in TCG systems refers to components that must be trusted because misbehavior might not be detected. This is easy to understand by noting that trusted does not mean trustworthy.

It is possible to have several roots of trust, and in fact the TCG specifications distinguish three of them in a typical trusted platform; a root of trust for measurement (RTM), root of trust for storage (RTS) and root of trust for reporting (RTR). The most relevant one is the RTM, which is a computing engine capable of making inherently reliable integrity measurements. Usually, the RTM is implemented by the normal platform computing engine, controlled by the core root of trust for measurement (CRTM).

Users should trust each root to function correctly without external supervision. Trusting "roots of trust" (determining whether they are trustworthy) can be achieved for instance by means of hardware certification procedures such as those currently used for security-sensitive systems (FIPS 140-2, ISO 15408, etc.).

The main goal of trusted computing is protecting the computing platform from software-based attacks. It does so by representing the different configurations that are considered safe, which are stored securely in the Trusted Platform Module.

The Trusted Computing technology is based on the concept of transitive trust (a.k.a. inductive trust), a process where a root of trust composed of a specific group of functions is used to determine whether trust should be placed in a second group of functions. If the trust level of the second group of functions is acceptable, the trust boundary is extended from the original root of trust to include the second group of functions and so on and so forth. Figure 1 illustrates this process.

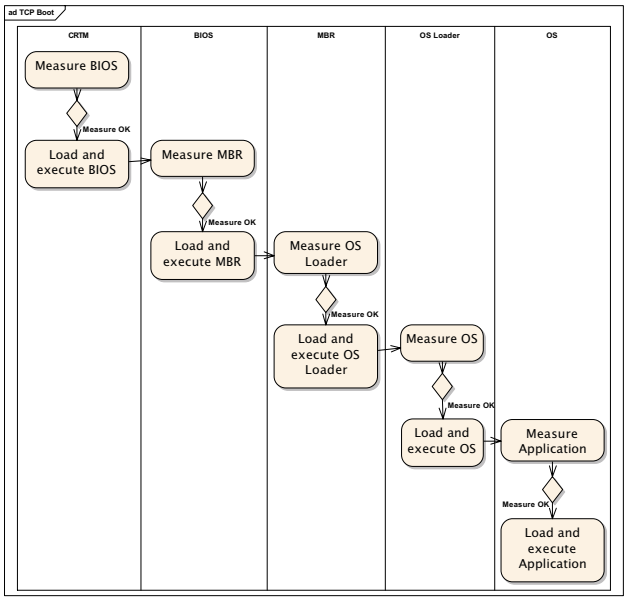


Fig. 1. Modified boot sequence in TC

Consequently, platform boot processes are modified to allow the TPM to measure each of the components in the system (both hardware and software) and securely store the results of the measurements in Platform Configuration Registers (PCR) within the TPM. This mechanism is used to extend the root of trust to the different elements in the computing platform.

Trusted Platforms should provide at least three basic features: *protected capabilities*, *integrity measurement* and *integrity reporting.*

The concept of *protected capabilities* refers to a set of commands with exclusive permission to access shielded locations such as memory, registers, etc., where it is safe to operate on sensitive data. Shielded locations can be accessed only by protected capabilities. TPMs must implement protected capabilities and shielded-locations in order to protect and report integrity measurements. Furthermore,

TPMs can provide additional protected capabilities such as random number generation or cryptographic key management.

*Remote Attestation* is one of the most controversial and debated upon features. In the typical scenario where Alice and Bob are communicating, Alice can take advantage of the remote attestation feature in order to determine whether the current configuration of Bob's platform is safe. This is possible for Alice because the TC technology provides mechanisms for her to measure (obtain a cryptographic hash) of the configuration of Bob's platform. If this configuration is altered or modified, a new hash value must be generated and sent to Alice in a certificate. These certificates attest the current state of Bob's platform and allow her to accept or reject the communication. One important disadvantage of this approach is the lack of flexibility and generality because it requires that all possible secure states are predefined. This remote attestation mechanism has some other important drawbacks [24]:

- It says nothing about program behavior;
- it is static. Static validation is not sufficient to stop attacks to software [25];
- upgrades and patches are hard to deal with;
- it is incompatible with highly heterogeneous computing environments; and
- the lack of efficient revocation mechanisms introduces problems.

*Integrity measurement* is the process of obtaining metrics of platform characteristics that affect its integrity; storing those metrics; and putting digests of those metrics in shielded locations. *Integrity storage* stores integrity metrics in a log and stores a digest of those metrics in PCRs. *Integrity reporting* is the process of attesting to the contents of integrity storage. These features are designed to allow platforms to enter any state, including those not identified as secure, but to prevent that a platform can lie about states that is was or was not in. An independent process may evaluate the integrity state(s) and determine an appropriate response. Obviously, these features have the potential to be used against the user of the platform.

The final goal of TC is to provide a computing platform on which users are not allowed to tamper with the application software, and where these applications can communicate securely with their authors and with each other. The original motivation is to be found in the field of digital rights management (DRM). In the first versions of TC, pirated software could be detected and deleted remotely. The mechanisms that are currently proposed are more subtle, though. TC will protect application software registration mechanisms, so that unlicensed software will not be installed.

Furthermore, TC apps will work better with other TC apps, so people will get less value from old non-TC apps (including pirate apps, but also legal pre-TC applications as well as homebrew ones). Additionally, some TC apps may reject data from old apps whose serial numbers have been blacklisted.

There are many other possibilities. Governments will be able to arrange things so that all documents created on civil servants' PCs are 'born classified' and can't be leaked electronically to journalists. Auction sites might insist that you use trusted proxy software for bidding, so that you can't bid tactically at the auction. Cheating at computer gaming could be made more difficult, etc.

Unfortunately, trusted computing has the potential to create important problems. For instance, TC can support remote censorship based on a business model called *traitor tracing*. In general, digital objects created using TC systems remain under the control of their creators, rather than under the control of the person who owns the machine on which they happen to be stored (as at present). This feature may allow for instance that a dictator can oblige the author of a writing that criticizes the dictatorship to censor it. Furthermore, the dictator can impose restrictions on the software that users can install so that the "approved" word processor can be ordered to do the deletion against the author's will. Other important drawback, especially for businesses, is that popular software suppliers can make it much harder for users to switch to their competitors' products.

In our opinion one of the main limitations of the Trusted Computing technology is the lack of support for applications to control their security settings and the lack of flexibility of the attestation mechanisms. These shortcomings result in applications not being able to control their security settings and not being able to adapt to dynamic situations. Finally, we believe that the overall approach is not fair, since protecting the applications (which we might consider a right of the application creator) requires controlling the configuration of the user machine (which should be a right of the user).

## IV. PROTECTED COMPUTING

The precedents of the Protected Computing approach were first introduced in 1984 [5]. However, the lack of adequate hardware and software support made it unfeasible in practice. However, the current concept of Protected Computing has its foundations on more recent work [1, 6]. The Protected Computing approach is based on the partitioning of the software elements into two or more parts. Some of them are executed in a secure processor, while others are executed in a normal (untrusted) processor. The basic idea is to divide the application code into two mutually dependent parts in such a way that:

- the public part cannot be used to gain knowledge about the protected part; and
- the communication trace between the parts cannot be used to gain knowledge about the protected part

In a Protected Computing setting, different secure coprocessors can be used (even simultaneously). On one hand, it is possible to use powerful cryptographic processors such as the IBM 4758 PCI and other similar products from nCipher, or even TPMs as secure coprocesors. On the other hand, it is also possible to use mobile and replaceable devices such as Smart Cards.

Protected Computing involves three types of actors: software producers, coprocessor manufacturers and clients (who should possess a coprocessor). For the sake of the description we will assume the simplest certification scheme where coprocessor manufacturers certify the public keys of their coprocessors and those of the software producers. More sophisticated schemes are considered in [1].

Protected Computing requires tamperproof coprocessors

that have cryptographic capabilities, contain a key pair generated inside the coprocessor and ensure that the private key never leaves it. The coprocessors must also contain a specific symmetric key, the public key of the coprocessor manufacturer and some support software.

Regarding the functional capabilities, the most relevant requirement is that coprocessors must be capable of executing code "on the fly". In particular, the coprocessors must contain an interpreter for the protected code sections and a runtime manager. Optionally, they can also contain more elements, such as a license manager or an electronic purse. Inside the trusted coprocessor, each application runs in a sandbox isolated from others. The runtime manager is used to keep separate memory spaces for each application. It is worth noting that the contents of the memory used by one application remains between calls to the coprocessor.

The basic idea is to use the coprocessor to enforce the correct execution of the protected parts of the program. These parts must be carefully selected in order to obtain the best protection.

Figure 2 depicts the transformation of the application code. The sections that are selected to be protected are encrypted with a symmetric key. Therefore coprocessors need to have access to this key in order to be capable of executing a protected application. An encrypted license containing the key is the usual way of providing the key to the coprocessor.

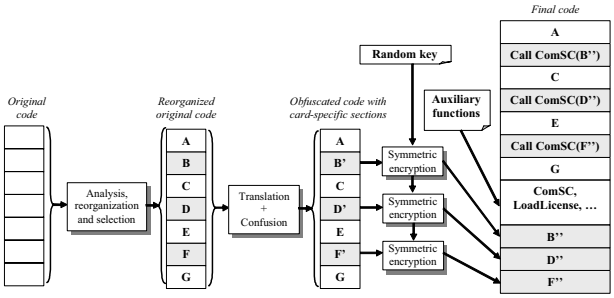


Fig. 2. Code transformation in Protected Computing

During the execution of the protected software the public part of the software sends the encrypted sections, along with any necessary data, to the secure coprocessor, where they are decrypted, loaded and executed. Some results are sent back to the unprotected part, while others are securely stored inside the secure coprocessor in order to be used in future computations. This technique, named call-chaining, is useful to avoid attacks based on sharing the coprocessor.

An important advantage of this scheme is the fact that different types of coprocessors can be used for different applications. For instance, it is possible to apply the scheme in order to protect mobile agents in a multi-agent setting, where several agents are sent to different (untrusted) agencies in order to perform some collaborative task. Because agents run in potentially malicious agencies, the main goal in this scenario is to protect agents from the malicious agencies.

The Protected Computing scheme can be applied in order to fulfill this goal by making every agent collaborate with one or more remote agents running in different agencies. These agents act as secure coprocessors for the first one. Likewise, these agents are in turn protected by other agents as shown in Figure 3. In this setting, an attack requires the cooperation of all agencies. For this specific example, it is possible that the protected parts of an agent are directly included in other agents. This increases the performance by avoiding the transmission of the protected code sections over the network.

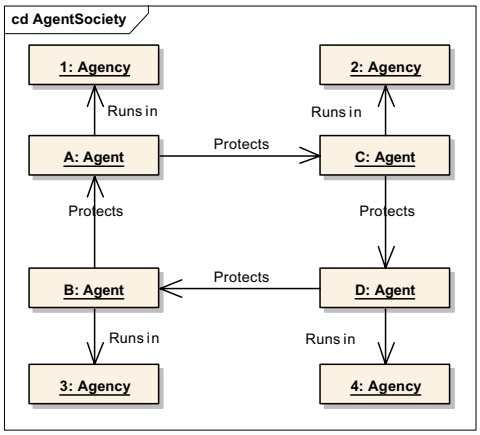


Fig. 3. A mutually-protected agent society

Protected Computing makes it possible to securely execute independent applications, without the need to protect all underlying hardware and software. Applications and computer owners have complete control over their security settings. In our opinion the fairness of this approach is higher compared to the Trusted Computing, since the applications can establish their security settings without imposing a specific configuration of the user machine, which remains under control of the user.

## V. PROTECTED COMPUTING AS AN ALTERNATIVE TO TRUSTED COMPUTING

An analysis of the two approaches leads us to the conclusion that both approaches are independent, compatible and complementary. On the one hand, Trusted Computing is more oriented to the protection of the platform against software-based attacks. Protected Computing, on the other hand, is oriented to the protection of applications both against malicious platforms and against software-based attacks.

The two approaches are based on very different philosophies. Trusted Computing aims at providing security by creating a trusted computing infrastructure starting from the hardware up to the OS and the applications. Alternatively, Protected Computing aims at securing independent applications without altering the rest of the computing elements.

On the applicability side Trusted Computing is oriented towards the local protection because it requires that the trusted element (the TPM) is physically connected to the standard computing platform in a secure way. In particular, no remote element can be used to play the role of the TPM. The approach followed by Protected Computing is to protect applications on the basis of their collaboration with a secure coprocessor. It is important to emphasize that the

communication trace between the parts cannot be used to gain knowledge about the protected part. Thanks to this feature it is possible to use remote elements as trusted elements.

Figure 4 shows a comparison of different aspects of the two approaches.

| *Trusted Computing* | *Protected Computing* |
|---|---|
| Based on static analysis and bottom-up creation of a trusted computing base starting from a tamperproof hardware device | Based on secure collaboration with a tamperproof element (a hardware device is not the only possibility) |
| Each software layer trusts all software and hardware layers below | Each protected software element trusts a tamperproof collaborator |
| Fixed device | Mobile device |
| High complexity | Low complexity |
| Not replaceable | Replaceable |
| Same for all applications | Can be different for different applications |
| All-or-nothing scheme (need to protect all underlying software from the boot-up sequence) | Can be selectively used |
| Dedicated device | Multi-purpose device (e.g. can be used for user identification, payment, etc.) |
| Switching modes is slow and not transparent (requires rebooting) | Switching modes is fast and transparent |
| Applications do not control security settings | Applications can select security settings |
| Costly | Inexpensive |
| One-for-all (all devices are equal) | Different manufacturers (room for competition and market-driven security) |
| O.S. dependent | O.S. agnostic |
| HW platform dependent (cannot be used with existing hardware without important modifications) | HW platform agnostic (can be used with existing hardware) |

Fig. 4. Comparison between Trusted Computing and Protected Computing

## VI. PROTECTED COMPUTING AS A COMPLEMENT TO TRUSTED COMPUTING

While it is true that the philosophies of the two approaches are very different, it is also true that they present several complementary features that are clearly useful.

In the first place, one of the most important drawbacks of the Trusted Computing approach is the limited support for applications to control their security settings. In particular, the need to have predefined platform configurations represents a very inflexible solution unless all platforms are obliged to be very similar and to have a reduced number of possible configurations. Although this is possible in practice, it is a very high price to pay for the enhanced security.

Protected Computing represents an important complement to Trusted Computing because it can support the flexible and dynamic control demanded by applications while, at the same time, maintaining the users' control over their platforms.

The combination of both approaches can be based on the enforcement, by means of Trusted Computing, of a minimal configuration, which would be basically related to the hardware elements, and then using Protected Computing to support the application-level security settings by taking advantage of the secure coprocessor (TPM) available in the system and in particular of the cryptographic capabilities of the TPM.

It is worth noting that, in this case, the number of possible configurations of the hardware is much more limited. Additionally, the dynamic changes on this configuration are much more infrequent.

Another possible way of combining the approaches is to use the Trusted Computing approach in order to protect the platform (hardware and OS) from malicious applications and use a Protected Computing mechanism in order to protect applications against malicious platforms as well as against other malicious applications.

## VII. CONCLUSIONS

In this paper we have introduced the concept of Protected Computing as an alternative and a complement to Trusted Computing. In our opinion Trusted Computing can represent an important step towards more secure computing systems. However, in its current status it rises too many and too important concerns to just be accepted.

We have highlighted some of the problems (interoperability, limit to free competition, lack of owner control, lack of "assurance", etc.) of the Trusted Computing approach, which make it unacceptable as it is now. Our view is that none of these problems is impossible to solve. We strongly believe that the cooperation of the scientific community can help in solving these problems in the right way.

Finally, we hope that this paper has demonstrated the existence of alternatives and complements to Trusted Computing that the scientific community should explore in order to guarantee the best possible solution for all parts.

## REFERENCES


[1] Maña, A. Protección de Software Basada en Tarjetas Inteligentes. PhD Thesis. University of Málaga. 2003.

[2] Maña, A., Matamoros, S. Practical Mobile Digital Signatures. Proceedings of 2nd International Conference On Electronic Commerce and Web Technologies (Ec-Web'02). Springer-Verlag. LNCS 2455. pp. 224-234. 2002.

[3] Pagnia, H., Gartner, F. C. On the impossibility of fair exchange without a trusted third party. Darmstadt University of Technology, Department of Computer Science Tech. Rep. TUD-BS-1999-02. 1999.

[4] Spalka, A., Cremers. A.B., Langweg, H. Protecting the creation of digital signatures with trusted computing platform technology against attacks by Trojan Horse programs. Proceedings of the 16th International Conference on Information Security (IFIP/SEC 2001). Kluwer Academic Publishers. 2001.

[5] 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.

[6] Maña, A., López, J., Ortega, J., Pimentel, E., Troya, J.M. A Framework for Secure Execution of Software. International Journal of Information Security, Vol. 3, Issue 2, Springer-Verlag, 2004.

[7] Hachez, G. A Comparative Study of Software Protection Tools Suited for E-Commerce with Contributions to Software Watermarking and Smart Cards. PhD Thesis. Universite Catholique de Louvain. 2003.http://www.dice.ucl.ac.be/~hachez/thesis_gael_hachez.pdf

[8] Lampson, B. W. Protection. Proc. 5th Princeton Conf. on Information Sciences and Systems, Princeton, 1971.

[9] Gunter Carl A., Homeier Peter, Nettles Scott. Infrastructure for Proof-Referencing Code. In Proceedings, Workshop on Foundations of

Secure Mobile Code, March 1997.
[10] Bennet S. Yee,A Sanctuary for Mobile Agents. Secure Internet Programming 1999.
[11] IBM. Cryptolope Technology Homepage. http://www-3.ibm.com/software/security/cryptolope/
[12] Garcia-Molina, H., Ketchpel, S., Shivakumar, N. Safeguarding and Charging for Information on the Internet. Proceedings of the Intl. Conf. on Data Engineering. 1998.
[13] Intertrust Technologies. Intertrust Technologies Home Page. http://www.intertrust.com/
[14] Stern, J. P., Hachez, G., Koeune, F., Quisquater, J. J. Robust Object Watermarking: Application to Code. In Proceedings of Info Hiding '99, Springer-Verlag. LNCS 1768, pp. 368-378, 1999. http://www.dice.ucl.ac.be/crypto/publications/1999/ codemark.pdf
[15] Collberg, C., Thomborson, C. Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection. University of Auckland Technical Report #170. 2000. http://www.cs.auckland.ac.nz/~collberg/Research/Publications/CollbergThomborson2000a/
[16] Wayner, P. Dissapearing Cryptography. Information Hiding, Stenography and Watermarking. Morgan Kauffman. 2002.
[17] Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K. On the (Im)possibility of Obfuscating Programs. Proceedings of CRYPTO `01. Springer-Verlag. LNCS 2139. pp. 1-18. 2001.
[18] Goldreich, O. Towards a theory of software protection, Proc. 19th Ann. ACM Symp. on Theory of Computing, pp. 182-194. 1987.
[19] Sander, T., Tschudin C.F. On Software Protection via Function Hiding. Proceedings of Information Hiding '98. Springer-Verlag. LNCS 1525. pp 111-123. 1998.
[20] Pearson, S., et al. Trusted Computing Platforms: TCPA Technology in Context. Prentice Hall PTR. 2002.
[21] White, S., Commerford, L. ABYSS: An Architecture for Software Protection. IEEE Transactions on Software Engineering. Vol. 16, Nb. 6. June 1990.
[22] Arbaugh, B., Farber, D., Smith, J. A Secure and Reliable Bootstrap Architecture. IEEE Symposium on Security and Privacy. 1997.
[23] Trusted Computing Group Specifications, Available from https://www.trustedcomputinggroup.org/specs/
[24] Haldar, V., Franz, M. Symmetric Behavior-Based Trust: A New Paradigm for Internet Computing. New Security Paradigms Workshop 2004 (NSPW-2004), White Point, Nova Scotia. 2004.
[25] Davida, G. I., Desmedt, Y., Blaze, M. J. Defending Systems Against Viruses Through Cryptographic Authentication. Proceedings of the IEEE Symposium on Security and Privacy. pp 312-318. 1989.