J. Lopez, A. Mana, and M. I. Yague, "XML-Based Distributed Access Control System", *Third International Conference on E-Commerce and Web Technologies (ECWeb02)*, LNCS vol. 2455, pp. 203-213, 2002.
NICS Lab. Publications: https://www.nics.uma.es/publications

XML-based Distributed Access Control System

Javier López, Antonio Maña, Mariemma I. Yagüe *

Computer Science Department
University of Málaga. Spain.
{jlm, amg, yague}@lcc.uma.es

Abstract. The use of attribute certificates and the concept of mobile policies have been proposed to overcome some of the limitations of the role based access control (RBAC) paradigm and to implement security requirements such as the "originator controlled" (ORCON) policy. Mobile policies are attached to the data that they control and enforced by their execution in trusted servers. In this paper we extend this idea to allow the execution of the policies in untrusted systems. Our extension allows that policies are bound to the data but not attached to it. By this modification security administrators are able to change policies dynamically and transparently. Additionally, we introduce X-ACS, an XML-based language designed to express policies in a simple and unambiguous way overcoming the limitations of other approaches. Important features of X-ACS are that it can be used by processors with limited capabilities such as smart cards while allowing the automated validation of policies.

1 Introduction

Despite of the popularization of distributed systems in most computing disciplines, systems for access control to information still rely on centralized security administration. Centralized control has important disadvantages [1] and does not facilitate the deployment of originator retained control mechanisms.

On the other hand, solutions proposed for distributed access control do not provide the flexibility and manageability required. An interesting approach based on the concept of mobile policies [2] has been proposed to solve some of the limitations of RBAC [3]. This system introduces the remote execution of the access control policies, addressing a solution for some of the problems of centralized access control, but requires that access control policies are executed in trusted computers (data servers in this case) which, in practice, represents just a small improvement over the single server model. Once access to a data object is granted, this data is sent to the client computer where it has no protection. Furthermore, because data and policy are compiled in a package, a change in the policy that controls a data object requires that the data-policy package is recompiled and distributed to all trusted servers.

* Work partially supported by the E.U. through project IST 2001-32446

In this paper we present ACDACS (Attribute Certificate Distributed Access Control System). ACDACS extends the mobile policy concept by allowing the execution of mobile policies in untrusted systems and enabling that policies are linked to the data object but not integrated with it. Therefore, policies can be dynamically changed in a transparent manner. We also introduce X-ACS; an XML-based authorization language designed to support a wide range of authorization scenarios in a simple and unambiguous way, to be used by processors with limited capabilities such as smart cards and to facilitate policy validation.

The rest of the paper is organized as follows. Section 2 presents the motivations. Section 3 summarizes some related work. Section 4 describes the ACDACS system. Finally, section 5 summarizes the conclusions and presents ongoing and future work.

2 Motivation

In distributed computing environments, such as extranets or research networks that comprise several institutions, the access policy applicable to each resource (data object, service, etc.) must be defined by the owner of the resource. The access control system must guarantee that the policy is enforced before access is granted to the resource. Moreover, there are many different situations where it is desirable that the owner of each resource is able to retain the control over it and to change the access policy dynamically and transparently. In these systems traditional centralized access control mechanisms do not provide the necessary functionality and flexibility. The need of a central authority and repository is not always acceptable by the institutions sharing the network. Furthermore, centralized systems are unable to provide means to guarantee that originators retain control over their information.

Several access control models have been introduced to fit different access control scenarios and requirements. Some schemes have also tried to integrate different models in a unified framework [4]. These approaches represent significant advances over traditional single-policy systems but, unfortunately, are still constrained by the underlying models and do not provide the necessary flexibility.

Role based access control is commonly accepted as the most appropriate paradigm for the implementation of access control in complex scenarios. RBAC can be considered a mature and flexible technology. Numerous authors have discussed its access properties and have presented different languages and systems that apply this paradigm [5][6].

The main problem with RBAC is that the mechanisms are built on three predefined concepts: “user”, “role” and “group”. The definition of roles and the grouping of users can facilitate management, specially in corporation information systems, because roles and groups fit naturally in the organizational structures of the companies. However, when applied to some new and more general access control scenarios, these concepts are somewhat artificial.

We believe that a more general approach is needed in order to be used in these new environments. For example, in the referred situations, groups are an artificial substitute for a more general tool: the attribute. In fact, the definition of groups is usually based on the values of some specific attributes (employer, position, ...). Some attributes are even built into most of the access control models. This is the case of the user element; the identity is just one of the most useful attributes, but it is not necessary in all scenarios and, therefore, it should not be a built-in component of a general model.

In actual access control models, the structure of groups is defined by the security administrator and is usually static. Although the grouping of users can suffice in many different situations, it is not flexible enough to cope with the requirements of more dynamic systems where the structure of groups can not be anticipated by the administrators of the access control system. In these scenarios new resources are incorporated to the system continuously and each resource may possibly need a different group structure and access control policy. Furthermore, the policy for a given resource may change frequently.

Our work is focused on the solution of the originator retained control issue providing fair distributed access control management and enforcement. The basic goal is to be able to express, validate and enforce access control policies without assuming the trust in the rest of the computers of the network. Finally, because the creation and maintenance of access control policies is a difficult and error prone activity, we have developed a language to express those policies and validate them to find contradictions or ambiguities.

3 Related Work

Access control is one of the most mature disciplines in computer security. Nevertheless, new models and functionalities in access control systems are required to fullfil the needs of new Internet applications. An interesting system for policy-based management of networks and distributed systems is presented in [7]. The separation of the policy specification from the access control implementation has been proposed in [8]. This separation follows the "network-centric" approach of Röscheisen and Winograd [9] and allows the policy to be modified dynamically, without changing its underlying implementation [10]. Other access control languages have been developed in the security community to support different access control approaches. Jajodia et al. present in [11] a logical language which allows users to specify the policy according to what access control decisions are to be made as well as the authorizations.

Several proposals have been introduced for access control to distributed heterogeneous resources from multiple sources. The Akenti Project [1] proposes an access control system designed to address the issues raised in allowing restricted access to distributed resources controlled by multiple stakeholders. The requirement for the stakeholders to trust the rest of the servers in the network, the assumption of the existence of a supporting identity PKI (public key infrastruc-

ture) and some security vulnerabilities related to the existence of positive and negative use-conditions are the main drawbacks of the Akenti system.

The PERMIS Project [12] objective is to set up an integrated infrastructure to solve identification and authorization problems. The PERMIS group has examined various policy languages – such as Ponder [7] – concluding that XML is the most appropriate candidate for a policy specification language. Because the PERMIS system is based on the RBAC model it shares its limitations. Also the requirement of a supporting PKI is hard to fulfil and it is not necessary in many authorization scenarios.

Since its inception, XML has been used for defining specific vocabularies to represent different human endeavor. In our context, the eXtensible rights Markup Language (XrML) [13] and eXtensible Access Control Markup Language (XACML) [14] are two proposals for standard XML extensions for digital rights management and access control. While XrML can be considered a mature specification, its complexity makes it not appropriate for our application scenarios where simplicity is essential. XACML is a very recent (still in the development process) and promising competitor in the field of access control languages. Both, XACML and our X-ACS language are based on XML Schema. The main differences are that X-ACS is designed to be used by processors with limited storage and processing capabilities such as smart cards and oriented to the validation of the policies.

Also based on XML, the Security Assertion Markup Language (SAML) [15] is an assertion language and messaging protocol for describing, requesting and sending authentication and authorization data between security domains. Its basic goal is to promote the interoperability between disparate security systems, providing the framework for secure e-business transactions across company boundaries.

Another interesting work, presented in [16], uses XML for defining a fine-grained access control system for XML documents. This approach differs from ours in that it is completely "server-side". Authorizations can be specified at document or instance level (in XML documents), or alternatively at schema level (in DTDs). Authorizations specified in a DTD are applicable to all XML documents that conform to it. Our proposal is based on a higher level language, the XML Schema language which presents an XML syntax and object oriented features. XML Schema can be extended with business rules expressed in Schematron. The expressive power of both languages allows the definition of advanced integrity constraints in a database-like style [17].

4 The ACDACS System

Considering the basic objective of providing means to implement the ORCON policy, the different scenarios considered and the analysis of previous proposals, our main goals for the ACDACS distributed access control system are:

- *Originator retained control.* Originators should be able to retain control over the resources they own even after access is granted to users.

– *Distributed access control management.* Administrators should be able to manage the resources they control regardless of the location of that resource.
– *Distributed access control enforcement.* Access control enforcement mechanisms must be distributed to avoid bottlenecks in request processing.
– *Flexibility.* The system should be applicable in different scenarios.
– *Independence.* The system should not depend on underlying infrastructures or authentication systems.
– *Dynamism.* There should be a fast and secure mechanism to change policies.
– *Ease of management.* The distributed approach should not introduce complexity of management. Supporting tools should be provided.
– *Efficiency.* Access control management and enforcement should be efficient.
– *Security.* The distributed access control mechanism must ensure the same level of security as a centralized one.

4.1 ACDACS System Architecture

The ACDACS system is based on the following idea: the security requirements of the processes related to the transmission and access to information are feasible if we can have a trusted software running in the client computer. Therefore, the system is based on the creation of mobile software elements that transport the protected content and enforce the access control policies. ACDACS is based on the SmartProt software protection system [18]. SmartProt partitions the software into functions that are executed by two collaborating processors. One of those processors is a trusted computing device that enforces the correct execution of the functions and avoids that these functions are identified or reverse engineered. We use smart cards as secure coprocessors although special online access servers can also be used as secure coprocessors for this scheme.

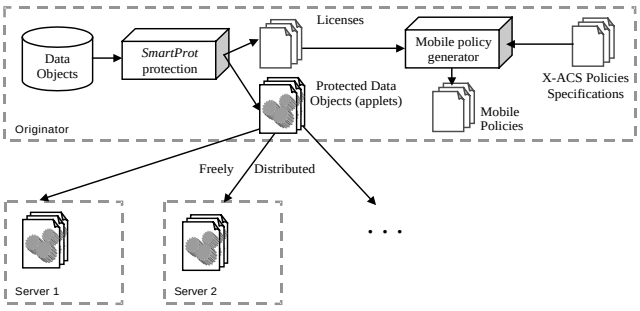

Fig. 1. The ACDACS Architecture

Figure 1 shows the architecture of the system. The unprotected data objects in the originator computer are transformed into PDOs (protected data objects). PDOs are Java applets that protect the data and enforce the access control mechanism. Policies are not included in the PDO. Instead, each PDO is linked

to the applicable policy by a mobile policy that is produced specifically for the client when requested. PDOs can be freely distributed to untrusted servers.

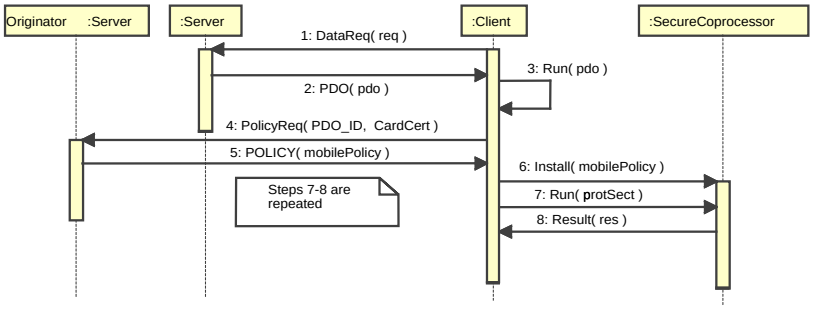

Fig. 2. ACDACS Functioning

Figure 2 depicts the dynamic functioning of the system. When the client requests some data object from a server it receives the PDO containing it. This PDO runs in the client computer. Before the PDO can execute the protected sections of its code it has to retrieve the corresponding mobile policy (which includes the license that allows the decryption and execution of the protected sections of the PDO). To do this the client sends a request containing the certificate of the public key of the secure coprocessor (the smart card or the access server). In case the PDO is retrieved from its originator, a mobile policy for that PDO is produced. Otherwise the server just forwards this request to the PDO originator.

In scenarios where the number of attributes and attribute certification authorities, known as SOAs (source of authorizations), are high it might be desirable to avoid that clients have to verify all the certificates directly. In this case a temporary authorization mechanism is used to map several attribute certificates from different SOAs to a single temporary authorization. This temporary authorization is a special attribute certificate signed by one SOA which simplifies the verification performed by the PDOs. This solution is especially useful when the client is accessing several PDOs from the same originator.

To allow owners of the information to be able to dynamically change the access control policy we must separate the policy from the PDO. When the PDO is executed it retrieves the policy from the originator and enforces it. The reasons to require the request of the mobile policy at access time and from the originator are that a high degree of flexibility is allowed, and the originator is given more control over the application of the policies. Nevertheless, for efficiency, originators can define certain validity constraints for each policy (based on time, number of accesses, etc. depending on the smart card features). Therefore policies can be cached by clients and used directly while they remain valid. Also the generation of the mobile policy is a reasonably fast process while the generation of PDOs is

slower. Furthermore, PDOs are much more stable than policies. Finally, opposed to PDOs, each mobile policy is specific for a smart card (or access server).

Policies are specified using X-ACS and are later translated into a more compact format to be included in the mobile policy. The link between PDO and the corresponding mobile policy is established by cryptographic means. The structure of the mobile policy is defined as follows:

$$MP ::= Policy, Encrypt_{CardPublicKey}(PDOkey, validity, H(Policy))$$

where *Policy* is the compact representation of the X-ACS policy, *CardPublicKey* is the public key of the smart card that will access the PDO, *PDOkey* is the random symmetric key used to encrypt the protected sections of the PDO, *validity* represents the limits of use of the MP and H is a collision-resistant one way hash function.

The mobile policy includes the key required by the smart card to decrypt and run the protected sections of the PDO. This key is encrypted for a specific smart card and will only be in the clear inside that card. As the PDO key is only known by the originator it is impossible for dishonest users to alter mobile policies or produce false ones.

We will now describe the main building blocks of ACDACS: (i) an infrastructure for software protection and (ii) a policy specification language and its associated validation mechanisms.

4.2 Software Protection

The ability to protect software that runs in the client computer in order to guarantee that it performs the intended function opens a way to solve the originator retained control problem. Our solution for this problem is based on the protection of the mobile software that we use to convey the information. The mobile software (PDO) contains some protected sections that must be executed by a secure coprocessor.

The SmartProt system includes three actors: the information provider, the card manufacturer and the client (who possess a smart card). The card manufacturer certifies the public keys of the smart cards. SmartProt requires smart cards that have cryptographic capabilities, contain a key pair generated inside the card and ensure that the private key never leave the card. Cards also contain the public key of the card manufacturer and some support software. Particularly, cards contain an interpreter for the protected code sections, a license manager, a runtime manager and, optionally, an electronic purse. Each protected software application runs in a sandbox isolated from others. The contents of the memory used by each application remain between calls to the card.

The process is divided into two main phases: *production* and *authorization*. The first step of the production phase consists in the translation of some specific sections of the original application code by functionally equivalent sections of card-specific code. The translation process also identifies the dependencies between these protected sections, reorganizes the code and introduces fake code to

confuse the attacker. These sections are then encrypted with a fresh key using a symmetric cryptosystem. The last step substitutes the original code sections by calls to a function that transmits the respective equivalent protected sections, including code and data, to the card. Some additional support functions are also included. The protected mobile software application generated in the production phase can be distributed and copied freely. In the case of the ACDACS system, the protected application (PDO) is a Java applet responsible for the transport of the information. Therefore, the protected mobile software includes the information to be accessed (which is encrypted), the access control enforcement mechanism and a cryptographic link to the access policy. The production phase is independent of the client card and will be performed just once for each piece of mobile software.

A license (specifically created for the smart card of the user) stating conditions (e.g. validity) is required to run the protected software. In the ACDACS system the license includes the access policy and is called Mobile Policy (MP). In the authorization phase, the new MP is produced linking the policy and the PDO. The MP contains validity constraints that are set by the security administrator according to the volatility of the policies.

MPs can be cached and used in the client computer while they remain valid. Just in the case that no valid MP is available, a new MP has to be requested and produced. The MP is obtained and loaded in the card as part of the PDO. When the MP is received by the client smart card, it is decrypted, verified and stored inside the card until it expires or the user decides to extract it.

Once installed, the MP allows the execution of the protected sections of the PDO. These protected sections do not reside in the cards. Instead, during the execution of the protected program, these sections are transmitted as necessary to the card where they are decrypted using the installed MP and executed. When finished, the card may send back some results. Some other partial results will be kept in the card in order to obtain a better protection against function analysis and other attacks.

The definition of the license (MP) structure permits a high degree of flexibility. Furthermore, licenses can be individually managed because each application has its own key. This is not possible in other software protection proposals where the protected sections of all applications share the same key (usually the protected processor key). In our scheme, because the protected sections are encrypted using a symmetric key kept inside the cards, and therefore known only by the PDO producer, dishonest users can not produce false sections.

4.3 Authorization Language

The XML data model can represent semantic information through descriptive tags. Complemented by related technologies certain types of database-like schemes can also be used [17]. XML Schema presents a rich set of data types, allowing user-defined data types, mechanisms such as inheritance, etc. XML Schema allows us to represent the semantics of the policies. Although some constraints such as, "if the value of `<Rights>` is `Update` then the value of `<Actions>`

should be `Notify`" are not expressible using XML Schema, our Policy Validator application is able to check, among others, this kind of constraints.

X-ACS policies are based on an XML Schema template[1] that facilitates their creation and syntactic validation. X-ACS has been developed taking into account that policies must be evaluated by smart cards. Therefore, other related languages are not well suited for the ACDACS system.

In our language, a policy consists of a set of `access_Rule` elements. Each one of these elements defines all the combinations of attribute certificates that allow the user to gain the access established by the `Rights` attribute. Therefore, it is composed as a series of `attribute_Set` required to gain access and the `Rights` obtained over the data in case access is granted. Each `attribute_Set` defines a particular attribute certificate combination associated with an optional `Action` (that has to be performed before access is granted). Attribute certificates will be used to provide evidence of the possession of each attribute. Therefore, attribute certificates are described stating their name (`attribute_Name`), value (`attribute_Value`) and the signer of the certificate (`SOA_ID`). Optionally, policies include `parameter` elements that are instantiated using the metadata available for the requested object.

Suppose our administrator states the policy shown in figure 3 in order to grant authorization to access the marks of a subject to lecturers and deny it to students. It is possible for lecturers to register as students of courses that they do not teach. In such a case, those lecturers would get access to their own marks.

```
<?xml version="1.0" encoding="UTF-8"?>
<policy xmlns="http://www.uma.es/ecWeb02"
xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.uma.es/ecWeb02 PolicyTemplate.xsd"
policy_Description="GRANT write_access TO marks IF Position='Lecturer'">
 <access_Rules>
   <access_Rule Rights="update">
     <attribute_Set>
       <attribute>
         <attribute_Name>Position</attribute_name>
         <attribute_Value>Lecturer</attribute_value>
         <SOA_ID>LCC_ADM</SOA_ID>
       </attribute>
     </attribute_Set>
   </access_Rule>
 </access_Rules>
</policy>
```

Fig. 3. WrongPolicy.xml

X-ACS policies are verified in several ways. Policies are verified syntactically using an XML Schema validator. Semantic verification is made in an automatic way by the Policy Validator, an XML application based on the DOM API. This validator also allows policies to be verified in the context where they will

[1] available as supplementary material

be applied. Policy context verification is based on the definition of a set of global rules establishing a series of facts about the environment of the system. This semantic information allows the detection of possible inconsistencies in the declared policy. Test cases can be defined by the administrator. Parameters are instantiated by the Policy Validator based on metadata expressed in test cases. The X-ACS global rules established about the context enable the detection of semantically incomplete policies. With the aid of the context validation the administrator would have detected the error in the previous example and stated the policy as shown in figure 4.

```
<?xml version="1.0" encoding="UTF-8"?>
<policy xmlns="http://www.uma.es/ecWeb02"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.uma.es/ecWeb02 PolicyTemplate.xsd"
policy_Description="GRANT write_access TO marks (metadata: Subject)
                    IF Position='Lecturer' AND Teaches=Subject">
  <parameter>Subject</parameter>
  <access_Rules>
    <access_Rule Rights="update">
      <attribute_Set>
        <attribute>
          <attribute_Name>Position</attribute_name>
          <attribute_Value>Lecturer</attribute_value>
          <SOA_ID>LCC_ADM</SOA_ID>
        </attribute>
        <attribute>
          <attribute_Name>Teaches</attribute_name>
          <attribute_Value>*Subject</attribute_value>
          <SOA_ID>LCC_ADM</SOA_ID>
        </attribute>
      </attribute_Set>
    </access_Rule>
  </access_Rules>
</policy>
```

Fig. 4. RightPolicy.xml

5 Conclusions and future work

We have presented the ACDACS distributed access control system to solve the originator retained control problem. We have described the underlying mechanisms that make possible this system: the SmartProt software protection scheme and the X-ACS language and tools. ACDACS is flexible, can be applied regardless of the attribute certification scheme, implements distributed access control management and enforcement mechanisms, does not depend on underlying infrastructures or authentication systems, allows the dynamic modification of policies in a transparent and efficient way and is secure. We have functional implementations of the SmartProt mechanism and the PDO (applet) generator. Regarding X-ACS, we have developed the XML Schema specification and the Policy Validator. Ongoing work is focused on the implementation of some of the components of the system such as the Policy Edition Assistant. We are working

on the definition of a generic mechanism to allow access control administrators to gain knowledge about the attributes certified by each SOA. This mechanism allows SOAs to define metadata about the attributes they certify. These metadata are also used by the policy creation and validation tools. We are currently applying the system to other scenarios such as digital libraries.

References

1. Thompson, M., et al.: Certificate-based Access Control for Widely Distributed Resources. In: Proc. of the Eighth USENIX Security Symposium (1999) 215-227
2. Fayad, A., Jajodia, S.: Going Beyond MAC and DAC Using Mobile Policies. In: Proc. of 16^{th} IFIP SEC. Kluwer Academic Publishers (2001)
3. McCollum, C.J., Messing, J.R., Notargiacomo, L.: Beyond the pale of MAC and DAC - Defining new forms of access control. In: Proc. of the IEEE Symposium on Security and Privacy (1990) 190-200
4. Jajodia, S., Samarati, P., Sapino, M.L., Subrahmanian,V.S.: Flexible support for multiple access control policies. ACM Transactions on Database Systems (2000)
5. Osborn, S., Sandhu, R., Munawer,Q.: Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies. In: ACM Transactions on Information and System Security, Vol.3(2) (2000) 85-106
6. Sandhu, R., Ferraiolo, D., Kuhn, R.: The NIST Model for Role-Based Access Control: Towards a Unified Standard. In: Proc. of the 5^{th} ACM Workshop on Role-based Access Control (2000) 47-63
7. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The Ponder Policy Specification Language. In: Proc. of Policy Worshop (2001)
8. Wedde, H.F., Lischka, M.: Modular Authorization. In: Proc. of the 6^{th} ACM Symposium on Access Control Models and Technologies (SACMAT) (2001)
9. Röscheisen, M., Winograd,T.: A Network-Centric Design for Relationship-based Security and Access Control. In: Journal of Computer Security, Special Issue on Security in the World-Wide Web (1997)
10. Sloman, M.S.: Policy Driven Management for Distributed Systems. Journal of Network and Systems Management, Vol. 2(4) (1994) 333-360
11. Jajodia, S., Samarati, P., Subrahmanian, V.S.: A Logical Language for Expressing Authorizations. In: Proc. of IEEE Symposium on Security and Privacy (1997) 31-42
12. Chadwick, D. W.: An X.509 Role-based Privilege Management Infrastructure. Business Briefing. In: Global Infosecurity (2002) http://www.permis.org/
13. ContentGuard, Inc.: eXtensible Rights Markup Language, XrML 2.0. (2001) http://www.xrml.org
14. Org. for the Advancement of Structured Information Standards.: eXtensible Access Control Markup Language. http://www.oasis-open.org/committees/xacml/
15. Org. for the Advancement of Structured Information Standards.: SAML 1.0 Specification Set (2002) http://www.oasis-open.org/committees/security/
16. Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., Samarati, P.: A fine-grained access control system for XML documents. In: ACM Transactions on Information and System Security (TISSEC), to appear.
17. Yagüe, M.I., Aldana, J.F., Gómez, C.A.: Integrity issues in the Web. In: Doorn, J. and L. Rivero (eds.): Database Integrity: Challenges and Solutions (2002) 293-321
18. Maña, A., Pimentel, E.: An Efficient Software Protection Scheme. In: Proc. of 16^{th} IFIP SEC. Kluwer Academic Publishers (2001)