Chapter IX
A Hardware Approach for Trusted Access and Usage Control

Nicolas Anciaux
INRIA Rocquencourt, France

Luc Bouganim
INRIA Rocquencourt, France

Philippe Pucheral
University of Versailles, France

ABSTRACT

This chapter advocates the convergence between Access Control (AC) models, focusing on the granularity of sharing, and Digital Right Management (DRM) models focusing on conditional authorizations and obligations. The convergence is also expected in terms of control enforcement considering that both AC and DRM models must be equally protected against any form of tampering and piracy. We capitalize on the democratization of powerful secure chip platforms (e.g., smart cards, secure USB dongles) which can be plugged in a variety of client devices (PC, PDA, cell phones, consumer electronics) to design a new architecture of a trusted access and usage control system. The benefits of the proposed architecture are exemplified in two different contexts: a fair DRM scenario and a healthcare scenario.

INTRODUCTION

In computer systems, access control models are used to express who is granted privilege to execute which actions on which set of resources. Many works have been conducted on access control management in the database context, trying to provide the finest granularity of sharing. In relational database systems, privileges can be granted on virtual objects, called views, dynamically built by an SQL query (Melton et al., 1993). In XML databases, XPath expressions are usually used to delineate the objects or document parts targeted by an access control rule (Bertino et al., 2001;

Gabillon et al., 2001; Damiani et al., 2002). The reason for this granularity concern is that databases often contain sensitive information (e.g., personal, commercial, administrative, military data) shared by a large number of users playing different roles with different privileges. Digital Right Management (DRM) models are also used to regulate the access to resources. DRM models primarily target the protection of digital assets (e.g., videos and sounds). The granularity of the access control is of lesser concern here but the conditions (e.g., to pay a fee) and obligations (e.g., to increment a counter at each copy) related to how a privilege can be exercised become central (XrML; ODRL). Hence DRM models complement access control with usage control. Another major concern of DRM systems is the enforcement of the access and usage control rules to fight against a large scale piracy threatening the global multimedia content industry (IFPI).

As the information distributed to customers becomes more complex and structured (e.g., encyclopaedia, cultural collections, stock exchange databases) the need for finer granularity rules arise in the DRM context. Conversely, as database applications show an increasing concern for regulating the usage made of the information legally accessed, the need for access control rules integrating contextual conditions and obligations arises. This is particularly true for databases containing personal data (Agrawal et al., 2002). The management of Electronic Health Records illustrates this well. For example, permissive access control rules should apply to a medical folder in specific contexts like an emergency situation. Obligations like registering all accesses to a medical folder in a log are also required to allow auditing the system.

In this chapter, we advocate the convergence between the access control and DRM worlds, encompassing the expression of fine grain access control and accurate usage control. This convergence is also expected in terms of control enforcement. In database environments, the access control is usually enforced by the database server, under the assumption that the server is trusted. Unfortunately, even the most defended servers (including those of Pentagon, FBI and NASA) have been successfully attacked, and database systems are identified as the primary target of computer criminality (Computer Security Institute, 2007). This motivated the design of new architectures where the server role amounts to deliver raw content to smart clients implementing the access control (Bouganim et al., 2004; Hacigumus et al., 2002). The question becomes how to enforce access and usage control on the client side. This question is not new in the DRM context, though no satisfactory solutions have been proposed yet. Indeed, today's DRM methods are so coercive that they do nothing but exasperating consumers and legitimize piracy (Champeau, 2004). The question is newer in the database world, people slowly becoming aware of the value of personal data and starting considering that protecting privacy is at least as important as protecting digital assets.

This chapter suggests solutions to enforce access and usage control on the client side thanks to secure hardware. Secure chips appear today in various form factors (smart cards, USB secure tokens, TPM, etc). They can be used to implement a Secure Operating Environment (i.e., a tamper-resistant storage and execution environment) in any device they are plugged in. This Secure Operating Environment is the required building block to design future trusted access and usage control systems. The suggested approach is expected to pave the way for new fair DRM models and for privacy conscious models of exchanging personal data.

This chapter is organized as follows. Section 2 presents background material on access control models, DRM models and secure hardware and sketches in this light an architecture for future trusted access and usage control systems. Section 3 summarizes two previous works related to the enforcement of XML access control policies in secure chip and the management of data embed-

ded on chip. These works can be seen as building blocks for the aforementioned architecture. Section 4 illustrates the benefit of the suggested architecture in two different contexts: a fair DRM scenario and a healthcare scenario. Section 5 concludes discussing future trends.

BACKGROUND AND PROPOSED APPROACH

Access Control & DRM Models

Access Control Models

An access control model is a framework by which security administrators grant or revoke the right to access some data or perform some action in a system. The set of authorizations regulating the use of all resources of a system is called an access control policy. Authorizations can take different forms depending on the underlying data model. For example, an authorization over a relational database is usually expressed as granting the permission to execute a given action (e.g., Select) on a relational table or view (i.e., a virtual table computed by an SQL query) (Melton et al., 1993). An authorization over a XML document is usually expressed as a composition of positive (resp. negative) rules selecting authorized (resp. forbidden) sub-trees in the document thanks to XPath expressions (Bertino et al., 2001; Gabillon et al., 2001; Damiani et al., 2002). Another dimension of the access control model is the way by which authorizations are administered, following a Discretionary (DAC) (Harrison et al., 1976), a Role-Based (RBAC) (Sandhu et al., 1996) or a Mandatory (MAC) approach (Bell et al., 1976). In this chapter, we do no restrictive assumption on the way authorizations are actually expressed and administered, except for illustrative purposes. We also consider both the relational and XML contexts even though a large part of this chapter focuses on the XML data model. Indeed, XML allows illustrating access control rules in a simple and intuitive way and is the preferred syntax for most existing DRM languages. We thus briefly detail below the XML and XPath syntax before presenting examples of XML access control rules.

XML has become a de-facto standard for the presentation, exchange and management of any information. Figure 1 shows a sample of XML metadata describing an MPEG-21 video. Roughly speaking, an XML document can be seen as a tree of elements, each one demarcated by an opening and closing tag (e.g., <Seq> and </Seq> for the Seq element). Attributes may be attached to elements (e.g., attribute value of the analysis element). Terminal elements (at the leaves of the tree) are represented by text (e.g., Closer).

Queries can be expressed over an XML document using the XPath language. An XPath expression allows to navigate in the document through the parent axis (denoted by /) and the descendant axis (denoted by //) and to apply predicates on elements and attributes. The result of an XPath expression is an element (or a group of elements) along with its (their) subtree(s). For example, the XPath expression /Video/Film/Seq/Key selects all the decryption keys of the sequences of the film while //Seq[SexRating>3] selects any sequence (anywhere in the document) having a direct child SexRating whose value is greater than 3.

Different authorization models have been proposed to regulate access to XML documents, tackling different facets of the problem. A particular attention has been paid on the granularity of the access control (from DTD to attribute instances) (Bertino et al., 2000; Damiani et al., 2002; Gabillon et al., 2001), on the performance of the algorithms implementing this control (Cho et al., 2002; Murata et al., 2003; Wang et al., 2001), on the distribution channel used to expose the information (pull, push, selective dissemination) (Bertino et al., 2002; Birget et al., 2001; OASIS Standard) and on the tamper-resistance of the access control (Bouganim et al., 2004; Miklau et al., 2002; Ray et al., 2002).

Figure 1. An MPEG 21 XML document

```
<Video>
 <   Title> Closer </Title>
 <   Film>
       <Seq>
     <      Desc> …… </Desc>
     <      SexRating> 3 </SexRating>
     <      Key> xxxxxxxxxx </Key>
       </Seq>
       <Seq>….</Seq>
 <   /Film>
 <   Bonus>
       <Seq>….</Seq>
       ….
 <   /Bonus>
 <   Analysis Value = "Technical">
       <Seq>….</Seq>
 <   /Analysis>
</Video>
```

Roughly speaking, an access control policy is composed of a set of positive (resp. negative) authorization rules granting (resp. denying) a given subject access to some nodes of the document. These nodes are usually selected thanks to XPath expressions. The descendant relationship among nodes is simply exploited as a mean to propagate authorization rules down through the XML hierarchy. There are substantial differences among the models in the way conflicts among – potentially propagated – positive and negative rules are tackled. In (Bertino et al., 2000; Gabillon et al., 2001), the complete subtree rooted at a forbidden node is forbidden. This constraint is relaxed in (Damiani et al., 2002)], allowing exceptions to a negative rule to be expressed. However, this leads to make visible the label (i.e., tag) of forbidden ancestor(s) in the path from the root to an authorized node. Replacing the node label by a dummy value has been proposed in (Fan et al., 2004; Gabillon, 2004) to reduce information disclosure in such situation.

For the sake of clarity and conciseness, we consider in this chapter a simplified access control model for XML, inspired by Bertino's model (Bertino et al., 2001) because several models share this same foundation. Subtleties of this model are ignored. In this simplified model, access control rules take the form of a 3-uple <subject, sign, object>. Subject denotes a user or a group of users. Sign denotes either a permission (positive rule) or a prohibition (negative rule) for the read operation (updates are not considered). Object corresponds to elements or subtrees in the XML document, identified by an XPath expression. The expressive power of the access control model, and then the granularity of sharing, is directly bounded by the supported subset of the XPath language. We consider in this chapter a robust subset of XPath denoted by $XP^{\{[],*,//\}}$ (Miklau et al., 2002). This subset, widely used in practice, consists of node tests, the child axis (/), the descendant axis (//), wildcards (*) and predicates or branches […]. Attributes are handled in the model similarly to elements and are not further discussed.

For simplification purpose again, we consider that the cascading propagation of rules is implicit in the model, meaning that a rule propagates from an object to all its descendants in the XML hierarchy. Due to the rule propagation along the hierarchy and to the multiplicity of rules for a same user, a conflict resolution principle is required. Conflicts are resolved using two policies: 1) Denial-Takes-Precedence, which states that if two rules of opposite signs apply on the same object, then the negative one prevails and 2) Most-Specific-Object-Takes-Precedence, which states that a rule which applies directly to an object takes precedence over a propagated rule. Finally, if a subject is granted access to an object, it is also assumed to be granted access to the path from the document root to this object (As in (Fan et al., 2004; Gabillon, 2004), names of denied elements in this path can be replaced by a dummy value). This *Structural* rule keeps the document structure consistent with respect to the original one.

DRM Languages

Several initiatives, e.g., XrML (XRML), MPEG-REL (ContentGuard, 2004), ODRL (ODRL),

XACML (OASIS Standard), XMCL (XMCL, 2001), demonstrate the need for expressive and extensible DRM languages capable of implementing a large variety of business models. Some of these initiatives are gaining a wide acceptance. For example, XrML from ContentGuard is used by Microsoft in its DRM implementations. XrML also formed the basis for MPEG REL, the Rights Expression Language of MPEG-21. The Open Digital Rights Language has been adopted by the Open Mobile Alliance (OMA) for its DRM standard. While different in their syntax and usages, the DRM languages mentioned above share strong commonalities. To illustrate this, let us consider XrML as a reference language. The constituents of an XrML grant (the central part of an XrML license) are:

- The *principal* to whom the grant is issued.
- The *right* that the grant specifies.
- The *resource* that is the direct object of the "right" verb.
- The *condition* that specifies the terms, conditions and obligations under which the right can be exercised.

Principal, right and resource are respectively named party, right and asset in ODRL and subject, action and resource in XACML with a similar meaning. ODRL integrates conditions within the right statement while XACML distinguishes between conditions and obligations. XACML supports also denials (i.e., negative authorizations) in addition to grants. The way a right is actually exercised is implementation dependent and may differ depending on the DRM infrastructure, on the application and on the type of content to be protected.

Access & Usage Control Rules

As the two preceding sections make clear, there are strong commonalities between access control and DRM control. In both cases, the control policy grants privileges on a set of objects to a set of subjects. There are also distinctions between access control and DRM. As stated earlier, (database) access control models are highly concerned by the sharing granularity, leading to the use of assertional languages (either SQL or XPath) to identify the objects, and the subjects, targeted by a rule. Conversely, DRM is primarily concerned by the conditions and obligations related to how a privilege can be exercised on a digital asset. This lead to integrate two new concepts in the rule definition, namely *context* and *obligation*. Context is used in DRM scenarios to assess a contextual predicate (e.g., "the number of films produced by Gaumont watched in the last 7 days is more than 3") conditioning the activation of a given privilege (e.g., "to watch a free of charge film"). Obligations are mandatory actions associated to the exercise of a privilege (e.g., "append a new record to the audit trail").

For the reasons given in the introduction, we advocate in this chapter the convergence between the access control and DRM worlds, encompassing fine grain access control and accurate usage control. To this end, we consider in the sequel general Access and Usage Control (AUC) rules expressed by <subject[Q], object[Q], sign, action, context[Q], obligation>, where:

- *subject[Q]* defines the set of grantees for that rule, expressed by a qualification Q over the domain of subjects and roles;
- *object[Q]* defines the set of objects targeted by the rule, expressed by a qualification Q over the object domain;
- *sign* denotes either a permission (+) or a prohibition (-);
- *action* is the operation the subject is authorized/forbidden to exercise on the object;
- *context[Q]* defines the conditions which must be met to make the rule activable, expressed by a qualification over the set of objects materializing the subject context (history of subject's actions, audit trail, etc).

- *obligation* is the procedure the system is mandated to achieve when the subject exercises a granted action; we assume here that this procedure sums up to perform updates in the subject context (e.g., logging an action).

Secure Enforcement of Access & Usage Control

Enforcement of Access & Usage Control

In database environments, the access control is usually enforced by the database server and sometimes by the application server. In both cases, the underlying assumption is that the server is trusted. As stated in the introduction, the server security has been shown weaker than expected in a number of situations. This motivated the design of new architectures where the server role amounts to deliver raw content to smart clients implementing the access control (Bouganim et al., 2004; Hacigumus et al., 2002). In addition, important use cases (e.g., selective dissemination, parental control) require performing the control on client devices. The question becomes how to enforce AUC rules on the client side. Except in very specific situations where the client device can be trusted (e.g., an ATM terminal), the challenge is to build an incorruptible AUC rule evaluator on untrusted client devices.

This section reviews the main approaches tackling this challenge and evaluates them in the light of the following criteria: (1) tamper-resistance, (2) ease and cost of deployment, (3) platform agnosticisms (i.e., whether a subject can exercise her privileges whatever the device she is using at a particular time), (4) expressive power of the AUC model and (4) privacy preservation.

Secure software. Software-based enforcement systems consist of a secure code executed on the client device to evaluate AUC rules and render regulated contents. Representative examples are the couples *Fairplay* / *itunes* from *Apple* and *Windows Media Rights Manager* / *Windows Media Player* from *Microsoft* which both control access and usage of music files. The unquestionable advantages of software-based solutions are their low cost and ease of deployment (e.g., over the internet) and their powerfulness (there is no particular limitation of the AUC model which can be supported). The counterpart is manifold. First, the security of the enforcement process is weak, as any software is vulnerable by nature. In particular, the integrity of the secure software code (a prerequisite to a correct execution) cannot be definitely assessed in an untrusted environment (Hauser et al., 2003). This weakness is important considering that (i) any customer is a potential attacker, (ii) the attacks can be conducted with impunity from the private sphere, and (iii) a single pirate can break the security of the whole system by distributing the crack on the internet. In addition, device and software agnosticisms are difficult to achieve. The former requires developing and embedding the secure software in any rendering devices (common computers but also mp3 player, car music players, etc.) and the latter binds the consumer to the use of an exclusive software. Finally, AUC rules involving context (e.g., based on past subject actions) are either impossible to evaluate (unless all actions are performed from the same device) or require centralizing the context information on a server thereby hurting subject's privacy.

Secure hardware. Hardware solutions can be seen as trusted black boxes receiving an encrypted content and delivering the clear text of its authorized subset according to the AUC rules. For example, set-top-boxes used for decoding Pay TV in Europe are equipped with secure microcontrollers like the STi5202 decoder chip. The Microsoft NGSCB initiative[1] relies on a secure chip, called the Trusted Computing Modules (TPM), to enforce DRM on common computers). Today, standards are emerging for defining hardware-based content protection for home network

devices, e.g., SVP Secure Video processor (SVP). Hardware-based solutions provide an unequalled level of security. Indeed, hardware is much more difficult to tamper than software, security protection is formally proven (and Evaluation Assurance Level can be assigned given completed Common Criteria (Common Criteria)) and large scale piracy is difficult to organize in case a chip is broken. If the secure chip comes in a portable and pluggable form factor (e.g., a USB secure token), it provides a nice answer to the platform agnosticisms (at least for devices equipped with a USB port) and privacy preservation requirements (the subject context is hardware protected and never centralized). On the other hand, hardware based solutions are more costly to deploy, linking manufacturers to higher investments and longer time to market (Grimen et al., 2006). More, current solutions largely underexploit the storage and computing capabilities of secure chips, thereby supporting only basic DRM models (raw decryption of multimedia flows).

The objective of this chapter is precisely to go one step further and show that, despite limited hardware resources, secure chip solutions can be devised to support trusted, powerful and privacy conscious AUC models. To understand how far we can go in this direction, it is mandatory to have a closer look at the current secure chip hardware constraints and at their forecast evolution.

Background on Secure Chips

The term secure chip refers to a monolithic chip providing strong anti-tampering features, whatever its actual form factor (i.e. physical sizes and shapes) ranging from the well known smart card to chips embedded in smart phones, USB keys and other forms of pluggable smart tokens. Note that powerful server-based secure coprocessors like the IBM 4758 (Dyer et al., 2001) fall outside this definition and are not discussed in this chapter. Secure chips share strong hardware commonalities and differ mainly in their interface to the host they connect to (Vogt et al., 2003).

Today's secure chips typically embed on a single silicium die: a 32 bit RISC processor (clocked at about 50 MHz), memory modules composed of ROM (about 100 KB), static RAM (some KB) and electronic stable storage (hundreds of KB of EEPROM or FLASH), and security modules enforcing physical security. The ROM is used to store the operating system, fixed data and standard routines. The RAM is used as working memory (heap and stack). Electronic stable storage is used to store persistent information, and holds data and downloaded programs. In the following, we analyze the main hardware trends for secure chips highlighting its very unique internal resource balance.

CPU resource: during the last ten years, embedded processors improved from the first 8-bit generation clocked at 2 MHz to the current 32 bit generation clocked at 50 MHz with an added cryptographic co-processor. At least two factors advocate for a continuous growth of CPU power. First, the rapid evolution of secure chip communication throughput (e.g., high delivery contactless cards, USB cards) allows secure chip applications to evolve towards CPU intensive data flow processing using cryptographic capabilities (e.g., DRM). Second, many efforts from secure chip manufacturers focus on multi-applications and multi-threaded operating systems demanding even more CPU power. In addition, note that increasing the CPU power as little impact on the silicon die size, an important parameter in terms of tamper-resistance and production cost on large scale markets.

RAM resource: secure chips hold today a few KB of static RAM, almost entirely consumed by the operating system (and the Java virtual machine in Java enabled chips). The gap between the RAM left available to the applications on one side and the CPU and stable storage resources on the other side will certainly keep on increasing. First, manufacturers tend to reduce the hardware resources to their minimum to save production costs. The relative cell size of static RAM (16

times less compact than ROM and FLASH, 4 times less compact than EEPROM) makes it a critical component, which leads to calibrate the RAM to its minimum (Anciaux et al., 2003). Second, minimizing the die size increases the tamper-resistance of the chip, thus making physical attacks trickier and more costly. RAM competing with stable memory on the same die, secure chip manufacturers favour the latter against the former to increase the chip storage capacity, thus enlarging their application scope.

Stable storage resource: the market pressure generated by emerging applications leads to a rapid increase of the storage capacity. Taking advantage of a 0.18 micron technology allows doubling the storage capacity of existing EEPROM memories. At the same time, manufacturers are integrating denser memory on chip, like NOR and NAND FLASH. NOR FLASH is well suited for code due to its "execute-in-place" property but its extremely high update cost makes it poorly adapted to data storage. NAND FLASH is a better candidate for data storage though its usage is hard. Reads and writes are made at a page granularity and any rewrite must be preceded by the erasure of a complete block (holding commonly 64 pages). Researchers in memory technologies aim at developing highly compact non-volatile memories providing fast read/write operations with fine grain access (e.g., MEMS, PCM, etc.) but this must be considered only as a long term perspective. Note that mass storage smart cards appear today in the market place, combining in a USB key form factor a smart card-like secure microcontroller connected by a bus to a Gigabyte-sized external (and then unprotected) NAND Flash module (Anciaux et al., 2007).

To conclude, secure chips appear as rather unusual computing environments and can be summarized by the following properties: (1) High processing power wrt the amount of RAM and on-chip data; (2) Tiny RAM wrt the amount of on-chip data; (3) Fast reads but slow and sometimes complex writes/rewrites in stable storage.

Trusted AUC Architecture

From the above discussion, we argue that secure chips can be a cornerstone of future trusted, powerful and privacy preserving AUC systems. This section describes such an architecture, centred on the secure chip and being agnostic with respect to the other components of a DRM system (client device and software, content server, licence server, etc). Figure 2 pictures this architecture and illustrates the software components and data which must be embedded on chip to enforce AUC rules of the form <subject[Q], object[Q], sign, action, context[Q], obligation>.

Being able to evaluate subject[Q] requires first authenticating the subject (the secure chip holder), leading to embed an authentication module and authentication data (credentials). The subject authenticates himself to the secure chip thanks to a PIN code and the secure chip authenticates itself to the other components of the system thanks to a cryptographic protocol, thereby implementing a strong authentication mechanism. Then, predicate Q can be evaluated over the subject credentials (e.g., subject.age > 18) to check whether the rule can become active.

To deliver the plaintext authorized result, the AUC engine must determine first which AUC rules are concerned by the subject request (may be more than one). Then, for each of these rules, it must (i) evaluate object[Q] and context[Q] thanks to the query processor and (ii) execute the obligation if the rule turns out to be active. The effect of this last step is in turn to update the context data (e.g., adding a record in an action trail).

Rules, objects and context data can be stored in plaintext in the secure chip (since embedded data is physically protected by the chip), encrypted on an external medium (e.g., transmitted from a remote server or stored on an insecure external memory), or both. For instance, in a DRM scenario, contextual data could be maintained on chip to protect the subject privacy, AUC rules (i.e., licenses) could be downloaded from a li-

cense server, and finally objects (i.e., the digital content) could be stored on a DVD medium or be downloaded from a content server.

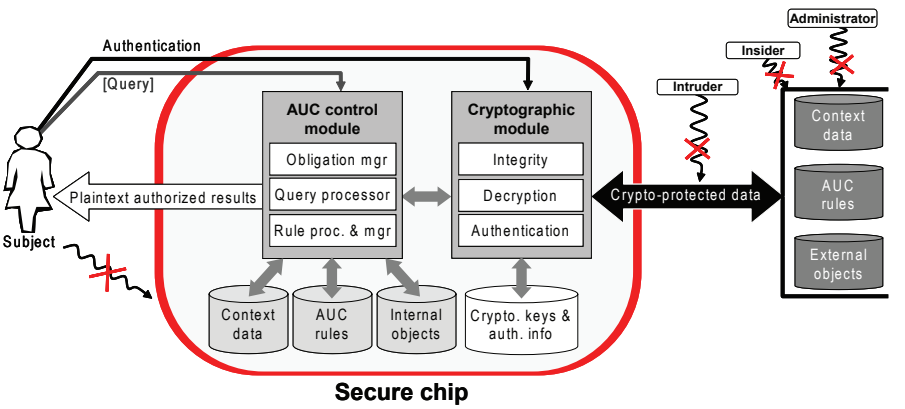
Elements stored on an external medium must be cryptographically protected to protect them against any forms of attacks which could be conducted by intruders, insiders, administrators or even by the subject himself (the secure chip holder). Basically external data must be encrypted to resist to *snooping* attacks, must contain secure checksums (e.g., hashes) to prevent from *spoofing* and *splicing* attacks, and finally must include timestamps to avoid *replaying* attacks (Anciaux et al., 2006). To this end, cryptographic modules and associated keys have to be embedded on the chip. Note that embedded data (e.g., the AUC rules) can be dynamically updated through a secure channel established with a remote source.

The AUC software embedded in the secure chip acts as an incorruptible mediator between the content provider and the content consumer and protects equitably the interest of both parties. The AUC system suggested above provides the content provider with a tangible (i.e., hardware based) guarantee against piracy. Indeed, the secure chip holder himself has no access to the embedded data (except those permitted by the AUC rules) and has no way to tamper it. In commercial DRM scenarios, privacy is usually a lesser concern and the consumer is asked to trust the server to protect data that can be gathered about him and his activity. The architecture sketched above re-establishes equity by providing the consumer with the same guarantees about the protection of his own data. To this respect, the roles of consumer and producer of information are inverted.

As section 4 will exemplify, this architecture is not tight to commercial DRM scenarios and can apply to any situation where sensitive data is exchanged between an information producer (could be a patient) and a consumer of this information (could be a physician querying the patient medical folder). The following roles can be distinguished, independently of the scenario:

- *information producer*: he is in charge of (1) protecting the storage of his data and (2) defining the adequate AUC policy to regulate the access and usage of his data. Whether obligations are integrated in the AUC policy, evidences of the fulfilment of these obligations must be given back to the producer. Data storage protection can rely on traditional server security (e.g., content servers) or on secure chip (e.g., embedded personal folders).

Figure 2. Functional architecture of a trusted AUC system. ©2008 Bouganim. Used with permission.



- *information consumer*: he can query/use the produced data according to his own privileges as defined in the AUC policy.
- *trusted AUC engine*: It is in charge of (1) enforcing the AUC rules, (2) checking the integrity of all incoming data, (3) securing the storage of embedded data if any, (4) supporting dynamic updates of embedded data. Whether obligations are integrated in the AUC policy, the AUC engine must build evidences of the fulfilment of these obligations. This means securing the storage of the subject context (the target of the obligations) and making this context queryable by the AUC engine, either to evaluate context dependent AUC rules or to allow the information producer auditing the AUC policy.

HARDWARE BASED SECURITY ELEMENTS

The trusted AUC architecture sketched in Section 2 needs further research efforts to become a reality. This section presents two previous works which can contribute to the definition of this architecture. The first work, presented in section 3.1, deals with the enforcement of XML access control policies over streaming documents thanks to a secure chip. The second work, presented in section 3.2, deals with the management of data embedded on chip.

Hardware Enforcement of XML Access Control Rules

When they do not rely on secure hardware, client-based access control solutions rely on data encryption. The data are kept encrypted at the server and a client is granted access to subparts of them according to the decryption keys in its possession. Sophisticated variations of this basic model have been designed in different contexts, such as DSP (Hacigumus et al., 2002), database server security (He et al., 2001), non-profit and for-profit publishing (Miklau et al., 2003; Bertino et al., 2001; Microsoft Inc.) and hierarchical access control (Akl et al., 1983; Birget et al., 2001; Ray et al., 2002). By compiling the access control policies into the data encryption, these solutions minimize the trust required on the client at the price of a rather static way of sharing data. Indeed, whatever the granularity of sharing is, the dataset is split in subsets reflecting a current sharing situation, each encrypted with a different key, or composition of keys. Thus, access control rules intersections are precompiled by the encryption. Once the dataset is encrypted, changes in the access control rules definition may impact the subset boundaries, hence incurring a partial re-encryption of the dataset and a potential redistribution of keys.

To circumvent the drawbacks mentioned above, we designed a solution taking advantage of secure hardware on the client side (Bouganim et al., 2004). This solution is a client-based access control manager capable of evaluating dynamic access control rules on a ciphered XML document with the benefit of dissociating the access control management from encryption. The problem addressed can be stated as follows: (i) *to propose an efficient access control rules evaluator coping with the hardware constraints of a secure chip:* first, the limited amount of secured memory precludes any technique based on materialization (e.g., building a DOM (W3C DOM) representation of the document); second, the limited communication bandwidth lead to minimize the amount of data to be downloaded in the secure chip; (ii) *to guarantee that prohibited information is never disclosed:* the access control being realized on the client device, no clear-text data but the authorized ones must be made accessible to the untrusted part of this client device; and (iii) *to protect the input document from any form of tampering:* under the assumption that the chip is secure, the only way to mislead the access control rule evaluator

is to tamper the input document, for example by substituting or modifying encrypted blocks.

To tackle this problem, we made the following contributions (Bouganim et al., 2007):

1. *Accurate streaming access control rules evaluator:* We proposed a streaming evaluator of XML access control rules, supporting $XP^{\{[],*,//\}}$, a robust subset of the XPath language (Miklau et al., 2002). The choice of a streaming evaluator allowed coping with the secure chip memory constraint. Streaming is also mandatory to cope with target applications consuming streaming documents. At first glance, one may consider that evaluating a set of XPath-based access control rules and a set of XPath queries over a streaming document are equivalent problems (Diao et al., 2003; Green et al., 2004; Chan et al., 2002). However, access control rules are not independent. They may generate conflicts or become redundant on given parts of the document. The proposed evaluator detects these situations accurately and exploits them to stop as soon as possible rules becoming irrelevant.
2. *Skip Index:* We designed a streaming and compact index structure allowing to quickly converge towards the authorized parts of the input document, while skipping the others, and to compute the intersection with a potential query expressed on this document (in a pull context). Indexing is of utmost importance considering the two limiting factors of the target architecture: the cost of decryption in the secure chip and the cost of communication between the chip, the client and the server.
3. *Pending predicates management:* Pending predicates (i.e., a predicate P conditioning the delivery of a subtree S but encountered after S in the document) are difficult to manage. We proposed a strategy to detect eagerly the pending parts of the document, to skip them at parsing time (whenever possible) and to reassemble afterwards the relevant pending parts at the right place in the final result.
4. *Integrity checking with random accesses:* We combined hashing and encryption techniques to make the integrity of the document verifiable despite the forward and backward random accesses generated by the *Skip Index* and by the support of pending predicates.
5. *Dynamic access control policy management:* The dynamicity of access control policies requires refreshing the access control rule definitions on the secure chip. We proposed a solution to ensure the confidentiality and integrity of this refreshing mechanism as well as to guarantee the consistency of the rule updates with respect to the processed document in order to avoid any unauthorized access.

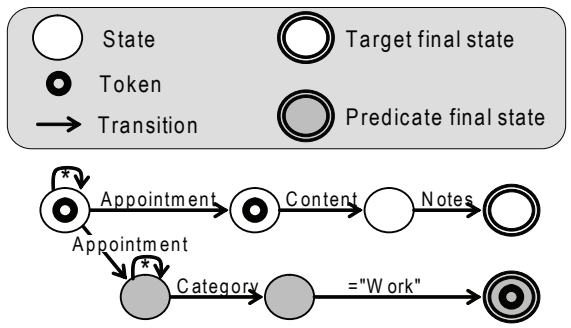
The remaining part of this section summarizes the streaming evaluation of access control rules, the integrity checking and the dynamic access control policies management. A complete description of the solution can be found in (Bouganim et al., 2007).

Streaming Evaluation of Access Control Rules

For each element of the input document, the access control rule evaluator must be capable of determining the set of rules that applies to it to determine the outcome for that element. The access control rule evaluator is fed by an event-based parser (SAX) raising *open*, *value* and *close* events respectively for each opening, text and closing tag encountered in the input document.

Each access control rule (i.e., XPath expression) is represented by a non-deterministic finite automaton (Hopcroft et al., 1979), named Access control rule Automaton (ARA for short). An ARA is made of states connected by transitions (see Figure 3). Tokens traverse the ARA while

Figure 3. Access control rule automaton. ©2008 Bouganim. Used with permission.



transitions are triggered, at document parsing time. An ARA has one target final state (representing the element targeted by the access control rule) and may have zero, one or more predicate final states (one for each predicate involved in the access control rule). When all final states of an ARA have been reached by a token, the corresponding access control rule becomes active, meaning that it applies to the forthcoming elements. Figure 3 shows the token positions in the ARA corresponding to a rule *R1* at the time the parser analyses the leftmost element of an XML document representing an agenda.

The following data structures are maintained in the secure chip to manage the set of ARA representing a given access control policy:

- *Token Stack:* The *Token Stack* memorizes the progress of tokens in all ARA and allows backtracking in the ARA.
- *Authorization Stack:* The *Authorization Stack* registers the rules having reached their target final state and is used to solve conflicts between rules. The status of a rule present in the stack can be: *positive-active* (⊕: forthcoming elements will be delivered), *positive-pending* (⊕?: the delivery of the forthcoming elements is conditioned by a predicate not yet evaluated), *negative-active* (⊖: forthcoming elements can be skipped), *negative-pending* (⊖?: the skip of the forthcoming elements is conditioned by a predicate not yet evaluated).
- *Predicate Set:* this set memorizes the predicates already evaluated.

The outcome of the current element can be easily determined from the information kept in these data structures, thanks to the conflict resolution algorithm presented in Figure 4. In the algorithm, AS denotes the Authorization Stack and AS[i]. RuleStatus denotes the set of status of all rules registered at level *i* in this stack. In the first call of this recursive algorithm, depth corresponds to the top of AS.

Integrity Checking with Random Accesses

Encryption and hashing are required to guarantee respectively the confidentiality and the integrity of the input document. Unfortunately, standard integrity checking methods are badly adapted to our context for two important reasons. First, the memory limitation of the secure chip imposes a streaming integrity checking. Second, the integrity checking must tackle the forward and backward random accesses to the document incurred by the Skip index and by the reassembling of pending document fragments. In this section, we sketch the solutions we proposed to face potential attacks on an input document.

In a client-based context, the attacker is the user himself. For instance, a user being granted access to a medical folder X may try to extract unauthorized information from a medical folder Y. Let us assume that the document is encrypted with a classic block cipher algorithm (e.g., DES or triple-DES) and that blocks are encrypted independently (e.g., following the ECB mode (Schneier, 1996)), identical plaintext blocks will generate

Figure 4. Conflict resolution algorithm

DecideElement(depth) → Decision ∈ {⊕, ⊖,?}
1: If depth = 0 then return '⊖'
2: elseif '⊖' ∈ AS[depth].RuleStatus then return '⊖'
3: elseif '⊕' ∈ AS[depth].RuleStatus and
4: '$\ominus^{?}$' ∉ AS[depth].RuleStatus then return '⊕'
5: elseif DecideNode(depth -1) = '⊖' and
6: '$\oplus^{?}$' ∉ AS[depth].RuleStatus then return '⊖'
7: elseif DecideNode(depth -1) = '⊕' and
8: '$\ominus^{?}$' ∉ AS[depth] RuleStatus then return '⊕'
9: else return '?'

identical ciphered values. In that case, the attacker can conduct different attacks: substituting some blocks of folders X and Y to mislead the access control manager and decrypt part of Y; building a dictionary of known plaintext/ciphertext pairs from authorized information (e.g., folder X) and using it to derive unauthorized information from ciphertext (e.g., folder Y); making statistical inference on ciphertext. Additionally, if no integrity checking occurs, the attacker can randomly modify some blocks, inducing a dysfunction of the rule processor (e.g., Bob is authorized to access folders of patients older than 80 and he randomly alters the ciphertext storing the age).

To face these attacks, we exploit two techniques. Regarding encryption, the objective is to generate different ciphertexts for different instances of a same value. This property could be obtained by using a Cipher Bloc Chaining (CBC) mode in place of ECB, meaning that the encryption of a block depends on the preceding block (Schneier, 1996). This however would introduce an important overhead at decryption time if random accesses are performed in the document. As an alternative, we merge the position of a value with the value itself at encryption time. Regarding integrity checking, the document is split into chunks whose size is determined by the memory capacity of the secure chip. Each chunk contains an encrypted *ChunkDigest* computed using a technique adapted from the Merkle hash Tree (Merkle, 1989). This technique gracefully combines encryption and hashing functions to allow random accesses to any part of the document with an 8 bytes alignment. The most original part of the proposed strategy is that integrity is checked in cooperation with the untrusted terminal at the price of decrypting one digest per visited chunk in the worst case (i.e., when the chunks accessed are not contiguous). As a conclusion, the document is protected against tampering and confidentiality attacks while remaining agnostic regarding the encryption algorithm used to cipher the elementary data. Unlike (Hacigumus et al., 2002; Bouganim et al., 2002), we do no assumption on any particular way of encrypting data that could facilitate the query execution at the price of a weaker robustness against cryptanalysis attacks.

Dynamic Access Control Policy Management

Dynamicity of the access control policies is a mandatory feature for a number of applications. This led us to design a secure mechanism to refresh the access control rules on the secure chip. Depending on the application scenarios, access control rule updates may be done pro-actively, requiring updates systematically before accessing the document, reactively when rule updates are detected, or even be disseminated jointly with the data. The update protocol must ensure three complementary properties: (i) *confidentiality* since access control rules definition may disclose unauthorized information; (ii) *integrity* since rule modification may mislead the rule processor; (iii) *consistency,* meaning that the set of access control rules stored on the secure chip must be up-to-date with respect to the processed document.

Access control rules confidentiality and integrity are enforced thanks to the encryption and hashing mechanisms presented above. Ensuring consistency is more difficult. Inconsistency between the set of access control rules and the

document may appear as a result of a malicious user whom may, for instance, filter the update flow, replay the document or the update flow. In any case, the secure chip must detect it and not deliver any data. Inconsistencies appears when a new access control policy is applied to an old document, thus potentially revealing unauthorized outdated data, or conversely, when an outdated access control policy is applied to a recent version of the document, thus revealing unauthorized up to date data. Both problems are solved using a crypto-protected cross reference versioning between the document and the access control rules.

Protection of On-Board Data

Databases on Chip: Existing Approaches and Systems

The value of secure chips to manage personal folders has been recognized in several domains like healthcare (medical folder), commerce (loyalties), telecommunication (address book) or mobile computing (user's profiles containing licenses, passwords, bookmarks, etc). In this spirit, MasterCard published the MasterCard Open Data Storage (MODS) API (Mastercard Inc.), allowing retailers, banks and other organizations to access and store data on customers' smart cards with an enhanced security for the holder. This motivated the design of data management techniques dedicated to secure chips.

Historically, the first attempt towards a DBMS embedded in a secure chip is ISOL's SQLJava Machine (Carrasco, 1999)] and the ISO standard for a smart card database language, SCQL (International Standardization Organisation, 1999). Both are addressing generation of secure chips endowed with 8 kilobytes of stable memory, explaining a design limited to low data volumes and simple database techniques. Since then, more elaborated secure chip DBMS have been developed. PicoDBMS (Pucheral et al., 2001; Anciaux et al., 2001) was the first full-fledged relational DBMS embedded in a smart card. PicoDBMS supports a robust subset of the SQL standard (full relational algebra) encompassing SCQL. The PicoDBMS kernel acts as a doorkeeper that authenticates the users and solely delivers the data corresponding to their privileges. In the relational context, the powerfulness of the access control is directly determined by the complexity of the (relational) views that can be built. To provide fine grain privileges, PicoDBMS supports complex query processing including select, project, join and aggregate operations. While PicoDBMS was exclusively designed to store stable data in EEPROM, other works consider secure chips endowed with FLASH memory (Bolchini et al., 2003).

Recently, a benchmark called Data management in Secure Chip (DiSC) has been proposed (Anciaux et al., in press) to help designing database techniques for secure chips. In particular, this benchmark enables to (1) compare different database techniques embedded on secure chips, (2) predict the limits of on-chip applications, and (3) provide co-design hints.

Challenges Related to Data Storage and Indexation

Designing appropriate data management techniques for secure chips is not straightforward. Light versions of popular DBMS like *Sybase SQL Anywhere Studio* (Giguère, 2001), *IBM DB2 Everyplace* (Karlsson et al., 2001), *Oracle Lite* (Oracle Corp., 2002) and *Microsoft SQL Server for Windows CE* (Seshadri et al., 1999) have been developed for lightweight devices, but they do not address the more severe limitations of secure chips (see section 2.2.2).A thorough re-thinking of all database techniques is mandatory to tackle accurately the secure chip hardware constraints. We summarize below the main database challenges incurred by secure chip data management in terms of storage, indexation and query processing.

Storage and indexing challenge. Traditional (disk-based) database servers are designed to fit fast sequential and slow random disk accesses, hence data locality plays an important role. The tradeoffs introduced by electronic stable memories (mainly FLASH or EEPROM) are totally different (cf Section 2.2.2). For example, EEPROM shares commonality with RAM in terms of access granularity (a memory word) and read performance (60-100 ns/word), but suffers from a dramatically slow write time (about 10 ms per word). Locality is no longer an issue, making disk-oriented structures irrelevant. Moreover, space consumption is an important concern in an embedded context. The challenge is then to devise compact structures for both data and indexes, adapted to the particularities of the access properties inherent to electronic memory.

Query execution challenge. Traditional query processing techniques typically resorts to materialisation (of intermediate results, hash tables, etc.) in the main memory and/or in swapping areas on disks. On the contrary, on-chip query execution strategies preclude materialisation. Within a secure chip, RAM is a scarce resource (Anciaux et al., 2003), and electronic memory is less appropriate for swapping (e.g., EEPROM exhibits a very slow write time, FLASH read/write costs exhibit a strong asymmetry and erasure is expensive). The challenge is then to devise RAM conscious query execution strategies, optimally avoiding swapping. Note that the two challenges are not independent since a lack of main memory for query processing may be compensated by a massive indexing.

EEPROM and Flash-Based Solutions

We precisely addressed these challenges when designing PicoDBMS in the particular context of a relational database and of EEPROM based secure chips. We sketch below the PicoDBMS design as an illustration of how the data management techniques required by a trusted AUC engine could be implemented. Of course, considering a different hardware platform (e.g., Flash based) or a different database model (e.g., XML) would lead to a different design.

PicoDBMS design has been driven by the following rules (Pucheral et al., 2001): *Compactness rule* (minimize data, index and code footprint), *RAM rule* (minimize RAM consumption), *Write rule* (minimize writes in EEPROM), *Read rule* (take advantage of very fast reads in EEPROM), *Access rule* (take advantage of low granularity and direct reads in EEPROM), *CPU rule* (take advantage of the over-dimensioned CPU) and *Security rule* (never externalize private data, minimize code complexity). The resulting technical solutions in terms of storage, indexing and query execution are summarized below.

Storage and indexing model. PicoDBMS takes advantage of a compact pointer-based model to meet the *Compactness, Read* and *Access rules* altogether. This model exploits a combination of Flat storage (FS) where tuples are stored sequentially and attribute values are embedded in tuples, Domain Storage (DS) where values are grouped in domains and attribute values are replaced by pointers within tuples and Ring Storage (RS) where each domain value is linked with a ring of pointers to all tuples sharing this value. FS is adequate when the attribute does not present value redundancy. DS precludes any duplicate value to occur and then acts as a compression technique. RS plays the role of an index. It links together all tuples sharing the same attribute value through a circular chain of pointer headed by this value. This chain of pointer is stored again in place of the attribute values, providing a similar and compact implementation of both select and join indices.

Query execution strategy. PicoDBMS relies on so called *extreme right deep trees* to meet the *RAM* and *Write* rules. It executes all operators (including select, project, join, group by and aggregate computations) in a pure pipeline fashion to avoid any materialization. Pipelining join and aggregate computation is not easy. Natural joins

(equijoins on key, foreign key pairs) are computed efficiently without RAM thanks to the ring index (see previous paragraph). For aggregate computation, a Cartesian product is introduced between the operands of the *group by* clause at the bottom of the tree, such that (1) the tuples sharing a same grouping value are naturally produced together at the tree leaves, (2) aggregation is computed at the tree root, one grouping value at a time. In (Anciaux et al., 2003), we provided a more general framework for designing RAM-constrained query evaluators.

PicoDBMS has been developed on Java (Anciaux et al., 2001) and C (Anciaux, 2004), is running on different smart card platforms and it performance has been assessed using the DiSC Benchmark (Anciaux et al., in press), demonstrating that complex on chip data management techniques can be implemented today.

SCENARIOS

This section illustrates the benefits of the AUC architecture in two different contexts: a fair DRM scenario and a healthcare scenario. It reports on experiments conducted in the field, with real prototypes running on advanced smart card platforms. These prototypes are not strict implementations of the AUC architecture presented in Section 2 but rather adaptations of the main ideas.

A Fair DRM Scenario

Due to the poorness of their model and the inflexibility of their architecture, existing DRM systems badly adapt to new attractive usage scenarios. In addition, consumers are reluctant to use them for privacy preservation and fairness concerns. Indeed, exasperating coercive methods do nothing but legitimizing piracy (Champeau, 2004).

In (Bouganim et al., 2007(2)), we presented a new software and hardware infrastructure aiming at reconciling the content providers' and consumers' point of views by giving the ability to develop fair business models (i.e., that preserve the interest of both parties). The proposed infrastructure, named MobiDiQ (Mobile Digital Quietude), is an XML-based tamper-resistant right management engine embedded in a secure chip. It enforces licenses/contracts (i.e. access and usage control rules) depending both on the digital content accessed on the device (the objects) and on personal data (i.e., the context) stored securely on the chip. The MobiDiQ access right engine is embedded in the secure chip to prevent any tampering to occur, thereby giving strong anti-piracy guarantees to the content provider. Embedding personal data in the chip brings also strong guarantees about user's privacy preservation.

MobiDiQ is able to enforce versatile and powerful AUC policies required to develop fairer business models. For instance, commercial conditions can be negotiated between institutions and content providers to help some categories of citizen (e.g., students) to access valuable contents at a special rate. Parental control rules can also be set up to protect children against dangerous contents but also against a prohibitive use of legal commercial contents. Privacy preserving gifting and loaning scenarios can be supported as well (Axalto Simagine, 2005).

Let us illustrate the MobiDiQ behavior on a fair superdistribution scenario (a student accessing content negotiated by her University). Figure 1 in Section 2 showed a sample of the XML metadata attached to a given video. The video is divided into several tracks (film, bonus, analysis), each one subdivided in sequences that include descriptions, values indicated the rating in terms of violence, sex content, decryption keys, etc. Figure 5 shows the user's profile and the access control rules expressing the licenses downloaded by the user. The user's profile records the fact that this user is a master student of the University of Versailles by means of certificates (simplified in the figure).

Figure 5. Profile and licenses

<Profile>
<SIM_PrivateKey> xdxdxd </SIM_PrivateKey>
<UV_Student> xabc </UV_Student>
<UV_Master> shqdq </UV_Master>
<Group value = "John"> JohnGrpPrivateKey </Group>
….
</Profile>

Profile XML Data

Video License:
Require University_Versailles License
Rule R1: < UV_Member, play, ? , /Video/Film>
Rule R2: < UV_Member**,** play, ? , /Video/Bonus>
Rule R3: < UV_Member**,** play, ? , /Video/Analysis>

University_Versailles License
Rule R4: < [not /Profile/UVStudent], play, **Θ**, /Video/Bonus>
Rule R5: < [not /Profile/UVMaster], play, **Θ**, /Video/Analysis>
Rule R6: < ALL, play, **Θ**, //Seq[SexRating > 3]>

Licenses

In this example, MobiDiQ has to deal with two licenses. The first one is issued by the content provider and states that any member of the University of Versailles may have the right to play the Film (R1), Bonus (R2) and Analysis (R3) track of the video. The second license, delivered by the University of Versailles adds some restrictions, specifying that the Bonus track is restricted to students (R4) while the Analysis track is restricted to Master students (R5). Finally, the last rule expresses that any sequence rated with a degree greater than 3 for sex content should not be played. The Require statement stipulates that the second license is mandatory to enable the first one. Note that a required license always restricts the possibilities of the user (Rules R4, R5 and R6 have a negative sign). Indeed, the university cannot grant more right than the one delivered by the content provider itself.

Roughly speaking, the MobiDiQ engine can be seen as a DRM virtual machine with XPath access control rules as bytecode and is nothing but a simplified instance of the trusted AUC architecture presented in Section 2.3. More precisely, the example above illustrates the enforcement of AUC rules of the form <subject[Q], object[Q], sign, action> but little effort is required to extend it with context and obligation.

Healthcare Scenario

PlugDB[2] and DMSP[3] are respectively a research and an experimental project conducted in the context of a medical-social network providing care at home for elderly people. There common objective is to improve the healthcare coordination while giving the control back to the patient over the access and sharing of her medical and social data.

Today, the coordination among the participants of this network (e.g., doctors, social workers) is organized around a paper-based folder. This folder

stays at home and is consulted and updated by every participant. This solution suffers from two main drawbacks. First, the paper-based folder is the primary source of confidentiality breach since no access control can be settled but hiding the complete folder. Second, the folder cannot be accessed and filled remotely, precluding remote diagnosis and leading to incomplete content.

To solve these problems, a trusted AUC architecture is being developed and relies on a new hardware component, called Secure Portable Token (SPT). A SPT combines, in a USB key form factor, a smart card-like secure chip and an external (i.e., unprotected) Gigabyte-sized NAND Flash.

The basic idea is to place the patient in the centre of the scene, giving her the full ownership of her medical folders, as enacted by the legislation but seldom put into effect in practice. According to the AUC dialect, patients are considered as *information providers* and practitioners as *information consumers*. The peculiarity here is that the information is actually produced by the practitioners (e.g., medical prescriptions), but it remains the property of the patient who is in charge of protecting its storage and regulating access to it. Hence, in the simplest setting of the architecture, each patient is equipped with a personal SPT storing securely the patient folder (encrypted in the NAND Flash) and enforcing AUC rules (thanks to an AUC engine embedded in the secure chip), just like a trusted portable personal database server.

A more complex setting is required (1) to guarantee the durability of the patient folder in case of crash or loss and (2) to allow remote diagnosis/updates by practitioners. To this end, a central server is used and provides a permanent internet connection to the practitioners and replicates part of the patient folders. As detailed in (Anciaux et al., 2008), the patient may choose different status for different parts of her folder, the two main interesting statuses in the context of this chapter being *secret data* and *confined data*. Secret data is exclusively stored on the patient SPT and can be accessed exclusively by the practitioner physically in front of the patient, with the restrictions imposed by the AUC policy. Confined data is data a patient may want to make durable or to share among a reduced circle of trusted persons (e.g., the family doctor), with the guarantee that nobody else can access it. To do this, the patient SPT encrypts the confined data, replicates the encrypted data on the central server and shares the encryption keys with the SPTs of people belonging to this trusted circle. This sharing scheme is possible under the assumption that every participant (including the practitioners) is equipped with a personal SPT. Again, note that the SPT holder has no free access to the SPT content and has no way to tamper it so that practitioners will still undergo the AUC control when accessing confined data.

According to Figure 2, the SPT plays a dual role. When the practitioner connects to a patient SPT and issues queries, the patient SPT acts as a trusted server guaranteeing a secure storage for the patient data and enforcing the AUC rules fixed by the patient. From the AUC engine point of view, AUC rules and context data is internal to the secure chip but patient data is stored encrypted within the SPT NAND Flash and is considered as external[4]. When the practitioner connects to the central server and issues queries over confined data, her own SPT plays the same role as above, except that (1) the AUC rules are external (even if they are cached internally, the original copy is in the patient SPT and a crypto-protected refreshing mechanism must take place through the central server) and (2) the patient data is external to the SPT, stored encrypted on the central server.

FUTURE TRENDS AND CONCLUSION

The scenarios presented above show that the trusted AUC architecture promoted in this chapter can pave the way for more powerful and versa-

tile DRM models. DRM is no longer synonym of basic conditional access to videos and sound assets. It can be understood in a broader sense encompassing every situation where (1) sensitive information has to be shared among partners having different privileges, (2) the expression of privileges includes access and usage control with contextual conditions and obligations, (3) the enforcement of the control must be guaranteed even in untrusted client devices.

Typically, the healthcare scenario opens up interesting perspectives towards a trusted AUC management of various personal information like scholarship or insurance folders and even information captured by our ambient intelligence surrounding (e.g., electronic surveillance of elderly people at home, employees monitored at work). Nothing justify that this information is less protected today than commercial digital assets. However, several technical challenges remains to be addressed before a general, robust and efficient trusted AUC architecture becomes a reality. Some of them are sketched below.

Limited storage capacity of secure chips: the stable storage of secure chips cannot accommodate a large volume of protected data. In new hardware architectures like SPT, the secure chip is connected by a bus to an external mass storage (within the same token) where protected data can overflow. However, external storage is not tamper resistant and must be protected against *snooping*, *spoofing*, *splicing* and *replaying* attacks. Designing cryptographic protections adapted to a database usage remains an open challenge for large datasets. Moreover, data can even overflow on a central server or be hosted on a server to increase availability (cf. healthcare scenario), adding a communication overhead to the encryption/decryption cost. This means that a new memory hierarchy should be considered (on-chip RAM, on-chip stable storage, off-chip token memory and server memory) along with ad-hoc cache management strategies adapted to data access patterns, object size, availability/connectivity considerations, etc.

Context ownership and resiliency: AUC rules may be based on the context of the consumer (e.g., a condition on her past actions) and may impose the obligation to fill in an audit trail accessible by the information provider. This raises a set of problems still open in our current design. First, the context of the consumer must be complete and up-to-date. This might be achieved by forcing her to access to any content through her own token (e.g., in the healthcare scenario, a practitioner accessing a patient folder would be forced to connect her own token to activate the AUC rule). Second, the context must be made resilient to avoid the consumer intentionally loose her secure chip and thereby potentially gaining more privileges by cleaning up her context. Third, the audit trail must be made accessible to the information provider in any situation. This introduces the problem of the audit trail location, protection and ownership and would probably lead to separate the audit trail from the context data.

Temporal AUC rules: temporal conditions have already been considered in commercial DRM models (XRML) and increase their expressive power and versatility. Typically, being able to express that a privilege can be exercised for a given duration (e.g., a one week trial), during a given time period (e.g., during working hours, during the week-end), possibly associated to a particular event (e.g., a diagnosis cannot be updated two days after an examination) is of general interest. The challenge here is that secure chips usually are not endowed with an internal clock and rely on the device they are plugged into to get time information. How to integrate a secure time server in a trusted AUC architecture is an interesting open issue.

REFERENCES

Agrawal, R., Kiernan, J., Srikant, R., & Xu, Y. (2002). Hippocratic Databases. *Int. Conf. on Very Large Data Bases*.

Anciaux, N., Bobineau, C., Bouganim, L., Pucheral, P., & Valduriez, P. (2001). PicoDBMS: Validation and Experience. *Int. Conf. on Very Large Data Bases*.

Anciaux, N., Benzine, M., Bouganim, L., Pucheral, P., & Shasha, D. (2007). GhostDB: Querying visible and hidden data without leaks. *ACM SIGMOD Int. Conf. on Management of Data*.

Anciaux, N., Benzine, M., Bouganim, L., Jacquemin, K., Pucheral, P., & Shaoyi Yin. (2008) Restoring the Patient Control over her Medical History. *IEEE Int. Symposium on Computer-Based Medical Systems*.

Anciaux, N., Bouganim, L., & Pucheral, P. (2003). Memory Requirements for Query Execution in Highly Constrained Devices. *Int. Conf. on Very Large Data Bases*.

Anciaux, N., Bouganim, L., & Pucheral, P. (2006). Data Confidentiality: to which extent cryptography and secured hardware can help. *Annals of Télécommunications*, *61*(3-4).

Anciaux, N., Bouganim, L., Pucheral, P., & Valduriez, P. (in press). DiSC: Benchmarking Secure Chip DBMS. *IEEE Transactions on Knowledge and Data Engineering*.

Anciaux N. (2004). *Database Systems on Chip*. Unpublished doctoral dissertation, University of Versailles, France.

Akl, S., & Taylor, P. (1983). Cryptographic solution to a problem of access-control in a hierarchy. *ACM Transactions on Computer Systems*, *1*(3), 239-248.

Axalto Simagine. (2005). Worldwide Mobile Communication and Java CardTM developer contest. 6th ed. held at 3GSM, Cannes, France. http://www.simagine.axalto.com.

Bell, D. E., & LaPadula, L. J. (1976). *Secure computer systems: Unified exposition and multics interpretation (Technical Report ESD-TR-73-306).* The MITRE Corporation.

Bertino, E., Castano, S., Ferrari, E., & Mesiti, M. (2000). Specifying and Enforcing Access Control Policies for XML Document Sources. *WWW Journal*, *3*(3).

Bertino E., Castano S., & Ferrari E. (2001). Securing XML documents with Author-X. *IEEE Internet Computing*, *5*(3).

Bertino, E., & Ferrari, E. (2002). Secure and Selective Dissemination of XML Documents. *ACM Transactions on Information and System Security*, *5*(2).

Birget, J., Zou, X., Noubir, G., & Ramamurthy, B. (2001). Hierarchy-Based Access Control in Distributed Environments. *IEEE International Conference on Communications*.

Bolchini, C., Salice, F., Schreiber, F. A., & Tanca, L. (2003). Logical and physical design issues for smart card databases. *ACM Transactions on Information Systems*, *21*(3).

Bouganim, L., & Pucheral, P. (2002). Chip-Secured Data Access: Confidential Data on Untrusted Servers. *Int. Conf. on Very Large Data Bases*.

Bouganim, L., Dang-Ngoc, F., & Pucheral, P. (2004). Client-Based Access Control Management for XML Documents. *Int. Conf. on Very Large Data Bases*.

Bouganim, L., Pucheral, P., & Dang-Ngoc, F. (2007). Dynamic Access-Control Policies on XML Encrypted Data. *ACM Transactions on Information and System Security*, *10*(4).

Bouganim, L., & Pucheral, P. (2007). Fairness concerns in digital right management models. *Int. Journal of Internet and Enterprise Management*, *5*(1).

Cho, S., Amer-Yahia, S., Lakshmanan, L., & Srivastava, D. (2002). Optimizing the secure evaluation of twig queries. *Int. Conf. on Very*

Large Data Bases.

Carrasco, L. C. (1999). RDBMS's for Java Cards ? What a Senseless Idea !. http://www.sqlmachine.com

Champeau, G. (2004). Fnacmusic.com: Le test complet sur Ratiatum.com - Le Peer-to-Peer (P2P) au delà du téléchargement. (In French). http://www.ratiatum.com/p2p.php?id_dossier=1708&page=1.

Chan, C., Felber, P., Garofalakis, M., & Rastogi, R. (2002). Efficient filtering of XML documents with XPath expressions. *Int. Conf. on Data Engineering*.

Common Criteria. The Common Criteria for Information Technology Security Evaluation. http://www.commoncriteriaportal.org/

ContentGuard. (2004). MPEG-21 Right Expression Language (MPEG-REL), ISO/IEC 21000-5:2004 standard, http://www.contentguard.com/MPEGREL_home.asp

Computer Security Institute. (2007). CSI/FBI Computer Crime and Security Survey. http://www.gocsi.com/

Diao, Y., & Franklin, M. (2003). High-performance XML filtering: An overview of filter. *Int. Conf. on Data Engineering*.

Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., & Samarati, P. (2002). A Fine-Grained Access Control System for XML Documents. *ACM Transactions on Information and System Security*, *5*(2).

Dyer, J., G., Lindemann, M., Perez, R., Sailer, R., van Doorn, L., Smith, S. W., & Weingart, S. (2001). Building the IBM 4758 Secure Coprocessor. *IEEE Computer*, *24*(10).

Fan, W., Chan, C. Y., & Garofalakis, M. (2004). Secure XML Querying with Security Views. *ACM SIGMOD Int. Conf. on Management of Data*.

Gabillon, A., & Bruno, E. (2001). Regulating access to XML documents. *IFIP Conf. on Database and Application Security*.

Gabillon, A. (2004). An Authorization Model for XML DataBases. *ACM Workshop on Secure Web Services*.

Giguère, E. (2001). Mobile Data Management: Challenges of Wireless and Offline Data Access. *Int. Conf. on Data Engineering*.

Green, T., Gupta, A., Miklau, G., Onizuka, M., & Suciu, D. (2004). Processing XML streams with deterministic automata and stream indexes. *ACM Transaction on Database Systems*, *29*(4).

Grimen, G., Mönch, C., & Midtstraum, R. (2006). Building secure software-based DRM systems. *Norsk informatikkonferanse*.

Hauser, T., & Wenz, C. (2003). DRM Under Attack: Weaknesses in Existing Systems. *Digital Rights Management - Technological, Economic, Legal and Political Aspects*.

Hacigumus, H., Iyer, B., Li, C., & Mehrotra, S. (2002). Executing SQL over encrypted data in the database-service-provider model. *ACM SIGMOD Int. Conf. on Management of Data*.

Harrison, M. A., Ruzzo, W. L., & Ullman, J. D. (1976). Protection in Operating Systems. *Communication of the ACM*, *19*(8).

He, J., & Wang, M. (2001). Cryptography and relational database management systems. *Int. Database Engineering and Applications Symposium*.

Hopcroft, J., & Ullman, J. (1979). *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Ed.

IFPI. International Federation of Phonographic Industry. http://www.ifpi.org/

International Standardization Organization. (1999). Integrated Circuit(s) Cards with Contacts

- Part 7, ISO/IEC 7816-7, 1999.

Karlsson, J. S., Lal, A., Leung, C., Pham, T. (2001). IBM DB2 Everyplace: A Small Footprint Relational Database System. *Int. Conf. on Data Engineering*.

Mastercard Inc. MasterCard Open Data Storage (MODS). https://hsm2stl101.mastercard.net/public/login/ebusiness/smart_cards/one_smart_card/biz_opportunity/mods

Merkle, R. (1989). A Certified Digital Signature. *Advances in Cryptology*.

Melton, J. A., & Simon, R. (1993). *Understanding the new SQL: A Complete Guide*. Morgan Kaufmann Ed.

Microsoft Inc. Windows Microsoft Media 9. http://www.microsoft.com/windows/windowsmedia/.

Miklau, G., & Suciu, D. (2002). Cryptographically Enforced Conditional Access for XML. *Int. Workshop on the Web and Databases*.

Miklau, G., & Suciu, D. (2003). Controlling access to published data using cryptography. *Int. Conf. on Very Large Data Bases*.

Murata, M., Tozawa, A., & Kudo, M. (2003). XML Access Control Using Static Analysis. *ACM Conference on Computer and Communications Security*.

OASIS standard, eXtensible Access Control Markup Language (XACML), http://www.oasis-open.org/committees/xacml.

ODRL. The Open Digital Rights Language Initiative. http://odrl.net/.

Open Mobile Alliance. http://www.openmobilealliance.org/

Oracle Corp. (2002). Oracle 9i Lite - Oracle Lite SQL Reference. Oracle Documentation.

Pucheral, P., Bouganim, L., Valduriez, P., & Bobineau, C. (2001). PicoDBMS: Scaling down database techniques for the smartcard. *VLDB Journal*, 10(2-3).

Ray, I., Ray, I., & Narasimhamurthi, N. (2002). A Cryptographic Solution to Implement Access Control in a Hierarchy and More. *ACM symposium on Access control models and technologies*.

Sandhu, R., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996). Role-based access control models. *IEEE Computer*, *29*(2).

SAX. A Simple API for XML. http://www.saxproject.org/.

Schneier, B. (1996). *Applied Cryptography, 2nd Edition*. John Wiley & Sons Ed.

Seshadri, P. (1999). Honey, I Shrunk the DBMS: Footprint, Mobility, and Beyond. *ACM SIGMOD Int. Conf. on Management of Data*.

SVP Alliance. Secure content anywhere, anytime. http://www.svpalliance.org/

Wang, Y., & Tan, K. L. (2001). A Scalable XML Access Control System. *Int. World Wide Web Conf.*

XMCL. (2001). XMCL - The eXtensible Media Commerce Language. http://www.xmcl.org/specification.html

XrML, eXtendible rights Markup Language. http://www.xrml.org/.

W3C DOM, DOM: Document Object Model. http://www.w3.org/DOM.

Vogt, H., Rohs, M., Kilian-Kehr, R. (2003). *Middleware for Communications, Chapter 16: Middleware for Smart Cards*. John Wiley and Sons Ed.

KEY TERMS

Access Control Models:

1. DAC: The Discretionary Access Control model (DAC) gives the creator of an object the

privilege to define the policy regulating access to this object, and granted privileges can be transmitted between users.

2. MAC: The Mandatory Access Control Model (MAC) attaches security level to objects and clearance level to users in a centralized way.

3. RBAC: The Role Bases Access Control Model (RBAC) introduces the concepts of Roles and Teams to improve the administration of access control policies for a large population of cooperating users.

Access Control Policy: Set of rules regulating the use of the resources of a system, each rule granting or revoking the right to access some data or perform some action in that system.

Attackers:

1. Intruder: a person with no database privilege, who infiltrates a computer system and tries to extract valuable information from the database footprint on disk.

2. Insider: a person properly identified by the database server (i.e., a registered user) who tries to get information exceeding her own privileges. The owned privileges give her more abilities than the intruder to tamper the system and to deduce valuable unauthorized content.

3. Administrator: a person who has enough (usually all) privileges to administer a computer system (System Administrator) or a DBMS (Database Administrator or DBA). These privileges give her the opportunity to access the database files and to spy on the DBMS behavior (e.g., main memory monitoring).

Attacks:

1. Data Snooping: an attacker examines the (potentially encrypted) data, on disk, in the memory or on the communication links and deduces unauthorized information.

2. Data Spoofing: an attacker deletes or modifies (even randomly) some data, thereby potentially corrupting the evaluation of access and usage rules and/or query evaluation.

3. Data Splicing: an attacker replaces a valid data by another valid data. This attack may lead to reveal unauthorized data, corrupt the evaluation of access and usage rules.

4. Data Replaying: an attacker replaces a valid data by one of its older version. For instance, replaying old access rules may lead to disclose unauthorized data.

Secure Operating Environment: A combination of hardware and software modules providing a tamper-resistant storage and execution environment protecting against any form of snooping and tampering attacks.

Usage Control: complements access control with *contextual predicates*, conditioning the activation of a given privilege, and *obligations*, i.e., mandatory actions associated to the exercise of a privilege.