Tamper-Resistant Ubiquitous Data Management

Luc Bouganim, François Dang Ngoc, Philippe Pucheral

Tamper-Resistant Ubiquitous Data Management

Luc Bouganim François Dang Ngoc Philippe Pucheral

INRIA Rocquencourt

78035 Le Chesnay - France

<Firstname.Lastname>@inria.fr

Abstract: *Chip-Secured XML Access (C-SXA) is a versatile and tamper-resistant XML-based Access Right Controller embedded in a smart card. C-SXA can be used either to protect the privacy of on-board personal data or to control the flow of data extracted from an external source. Tamper-resistance is inherited from the smart card for on-board data or achieved using cryptographic techniques for external data. C-SXA can provide different views of the same on-board or external data depending on the user or application accessing them. Moreover, access control on external data can benefit from on-board storage to enforce powerful, context dependant access control policies. These two features allow C-SXA to address a wide range of applications such as secure portable folders, data sharing among a community of users, parental control and Digital Right Management, in a more secure and accurate way than existing technologies. This work relates the C-SXA experience. We first motivate the interest of the approach and describe different usage scenarios. We then present the internals of C-SXA and show how they tackle the smart card's hardware limitations. Finally, we demonstrate its viability showing how our smart card engine can be integrated in a distributed architecture including the smart card, the server and the user terminal, making the complete chain from the user to the data secure*

Key words: *ubiquitous data management, data confidentiality, XML access control, XML data store, smart card.*

1 Introduction

The rapid growth of ubiquitous computing impels mobile users to store personal data on the Web in order to increase data availability and facilitate data sharing among partners. However, Database Service Providers (DSP) arouse user's suspicion because DSP's privacy policies have been frequently dishonored [AKS02]. In addition, no one can fully trust traditional server-based security mechanisms against more and more frequent and malicious attacks [FBI03]. While client-based security policies have been historically disregarded considering the vulnerability of client environments [Rus01], the emergence of hardware elements of trust in client devices drastically changes the situation [BoP02]. Secure tokens and smart cards plugged or embedded into different client devices are exploited today in a growing variety of applications (e.g., authentication, healthcare folders, digital right management).

Chip-Secured XML Access (C-SXA) belongs to this new category of PETs (Privacy Enhancing Technologies) taking their security from tamper-resistant hardware devices. C-SXA is a versatile XML-based Storage Manager and Access Right Controller embedded in a smart card. It evaluates the user's privileges on on-board or external encrypted XML data and delivers the authorized subset of these data. Combining storage and access control in the same secured architecture allows tackling accurately two important issues:

- The first issue is protecting the privacy of personal data embedded in a mobile client device. Indeed, the rapid growth of smart card [DH04, Gem04] and MOPASS card [Mop04] stable storage capacity (from kilobytes to hundreds megabytes) makes the management of secure on-board data realistic and more and more attractive for a wide range of applications.

- The second issue is controlling the access to external resources through a mobile client device. Indeed, the erosion of trust put in traditional database servers and DSP, the growing interest for different forms of data dissemination and the concern for protecting children from suspicious Internet content are different factors that lead to move the access control from servers to secured clients. In that case, the external data are stored externally in an encrypted form on a server and are delivered to the client in a streaming fashion.

Combining on-board XML storage and access control in a smart card brings important benefits. First, providing access control management on on-board data allows defining different views of the same on-board data for different users or software willing to access these data. This functionality is required by applications like secure portable folders (different users may query, modify and create data in the holder's folder) or virtual home environment (different software have a partial access to the holder's environment). Conversely, access control on external data can benefit from secured on-board storage to enforce powerful access right policies, in the spirit of advanced access right management languages like XrML [XrM]. XrML allows defining rules based on historical data (e.g., a user is granted access to given data provided she did, or did not, some actions in the past). Past actions have therefore to be recorded securely to avoid any tampering[1]. This combination of features allows C-SXA to address important classes of applications, like secured portable folders, data sharing among a community of users (family members, relatives or business partners), parental control, teacher control in e-learning

[1] Historical data should be stored securely for two reasons: (i) a malicious user may tamper the historical data to inhibit the access control; and (ii) tampering may disclose personal information, thereby hurting privacy.

environments as well as Digital Right Management (DRM), in a more secure and accurate way than existing technologies.

On the one hand, the hardware security features of the smart card guarantee a high tamper-resistance both for the on-board data and for the access control management engine. On the other hand, its severe hardware limitations (very slow writes in stable storage, tiny RAM, constrained stable memory, etc.) makes traditional storage and access control technologies irrelevant. In a recent paper, we proposed a tamper-resistant streaming evaluator of access control rules for regulating access to XML documents [BDP04a]. The proposed solution relies on a non-deterministic automata engine embedded in a smart card and on a dedicated streaming index structure allowing skipping the irrelevant (i.e., forbidden) parts of the input document. Based on [BDP04a], we built two JavaCard prototypes running on real smart card platforms. The first one implements a hard-coded collaborative agenda application demonstrating the potential of a smart client-based access control approach. This prototype focuses more on the application's aspects than on the underlying technology. This work, despite limited, has been rewarded with the silver award of the e-gate'04 contest[2], showing the growing interests of industrials to enforce access control with hardware solutions on client devices. In the second prototype we made a faithful implementation of the principles proposed in [BDP04a] and add support for on-board data and for XrML like access control rules. Our objective was threefold: (i) to validate the solution proposed in [BDP04a] on a real smart card platform; (ii) to show how our smart card engine can be integrated in a distributed infrastructure including the smart card, the server and the user terminal, making the complete chain from the user to the data secure; and (iii) to illustrate the generality of the approach through representative applications exhibiting different profiles wrt the data origin (on-board data, external data or both), the way the information is accessed (pull vs. push), the type of this information (textual vs. video) and the response time requirements (user patience vs. real time).

This paper relates the C-SXA experience and details the three aforementioned objectives and the technological means to reach them. To meet these objectives constitutes a rather significant – and complementary – contribution compared to [BDP04a].

The sequel of this paper is organized as follows. Section 2 presents the main application

[2] This international contest is organized yearly by Sun, Axalto and ST Microelectronics and rewards innovative smart card applications.

domains targeted by C-SXA and discusses the current state of the technology in these domains. Section 3 introduces the architecture of C-SXA and illustrates its use through various scenarios. Section 4 gives the internal algorithms embedded in the smart card. Section 5 gives a technical description of the global infrastructure. Finally, Section 6 concludes.

2 Target applications and alternatives

2.1 Storage and protection of on-board data

The value of smart cards to secure and share in a controlled way personal information has been recognized in several domains like education (scholastic folders), commerce (loyalties), telecommunication (address book) or mobile computing (user's profiles containing licenses, passwords, bookmarks, etc) [PBV01] (see Figure 1). MasterCard published recently the *MasterCard Open Data Storage (MODS)* Application Programming Interface "*to meet the desire expressed by customers to better control how much information they are willing to share with whom*" [Mas02]. MODS allows retailers, banks and other organizations to access and store data on users' smart cards with an enhanced security for the smart card's holder. The IBM-Harris report on consumer privacy survey strongly highlights this same requirement [IBM]. While the need for on-board data management and sharing facilities is clearly established, few technical solutions have been proposed yet.

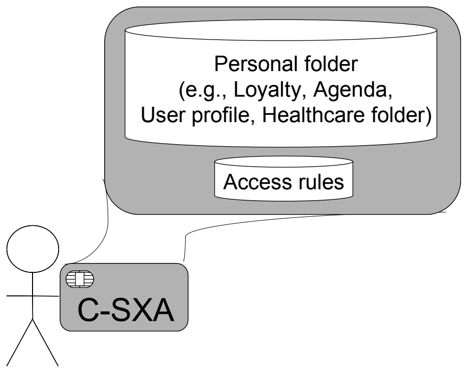


Figure 1: *Storage and protection of on-board data*

Light versions of popular DBMS like Sybase SQL Anywhere Studio [Syb00], IBM DB2 Everyplace [KLL+01], Oracle Lite [Ora02] and Microsoft SQL Server for Windows CE [Ses00] have been designed, targeting the growing number of mobile phones, PDAs, and other portable consumer devices. These systems have been designed with software portability and footprint reduction in mind. However, they do not address the more severe limitations of smart cards. Indeed, smart cards have a very specific hardware architecture (Section 3 will highlight this point), entailing a thorough re-thinking of database techniques.

The first attempts towards a smart card DBMS were ISOL's SQLJava Machine DBMS [Car99] and the ISO standard for smart card database language, SCQL [ISO99]. Both were addressing generation of smart cards endowed with 8 kilobytes of stable memory. While their design was limited to mono-relation queries, they exemplify the strong interest for dedicated smart card DBMS. PicoDBMS [PBV01, ABB01], designed by our team, was the first full-fledged relational database system embedded in a smart card, supporting a robust subset of the SQL standard (and then encompassing SCQL). PicoDBMS is however a complex technology primarily designed to manage efficiently huge and well structured embedded folders. Finally, MODS [Mas02] is based on flat files and crude access rights (i.e., file level access rights).

The versatility and wide acceptance of the XML standard [W3C] makes it the best candidate today to describe, organize, store and share the variety of data that appear in the above applications. Thus, there is a strong need for a native XML data store embedded on chip. To the best of our knowledge, C-SXA is the first attempt in this direction.

2.2 Protection of external data

Different requirements motivate a secured access to external data from a smart card (see Figure 2): (1) the management of personal folders (as above) whose size exceeds the smart card storage capacity; (2) the sharing of personal or professional data (e.g., agenda, address book, bookmarks, etc) among a community of users (family, friends, colleagues, partners); (3) the consumption of information disseminated through a license-based distribution channel or (4) accessed freely through the Internet.

While the internal data are protected by the tamper-resistance of the chip, external data need be protected by encryption. The role of encryption differs depending on the source of threatening. In cases (1) and (2), encryption is required to preserve the confidentiality of the data hosted in untrusted servers, considering the increasing suspicion towards traditional database servers [FBI03] and DSP [HIL02]. In case (3), the role of encryption is protecting digital assets from illegal access and copying [Sma]. Finally, case (4) refers to the ever-increasing concern of parents and teachers to protect children by controlling and filtering out what they access on the

Internet [PIC]. To meet this last requirement, encryption can take place in the Web server, in the ISP or in the client device communication card while the access control and decryption remains confined in the smart card.

Usually, the data are kept encrypted at the server and a client is granted access to fragments of them according to the decryption keys in its possession. Variations of this basic model have been designed in different context, such as encrypted backups for personal data [Sky], encrypted data hosted by untrusted DSP [HIL02], encrypted relational databases [Ora04, HeW01], for-profit as well as non-profit publishing [MiS03, BCF01, Sma]. Despite their respective merit, these models have in common a static way of sharing data. Indeed, access control policies are all precompiled by the encryption, so that changing these policies may incur a partial re-encryption of the dataset and/or a potential redistribution of keys.

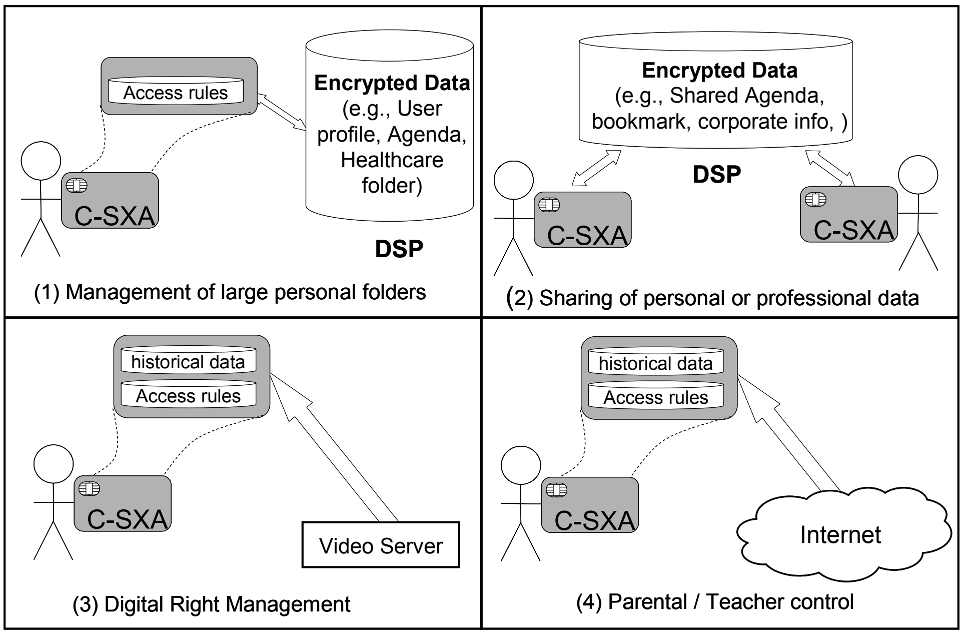

Access rules
Encrypted Data (e.g., User profile, Agenda, Healthcare folder)
C-SXA
DSP
(1) Management of large personal folders
Encrypted Data (e.g., Shared Agenda, bookmark, corporate info,)
DSP
C-SXA
C-SXA
(2) Sharing of personal or professional data
historical data
Access rules
C-SXA
Video Server
(3) Digital Right Management
historical data
Access rules
C-SXA
Internet
(4) Parental / Teacher control

Figure 2: *Protection of external data*

Unfortunately, there are many situations where access control rules are user specific, dynamic and then difficult to predict. In medical folders, the rules protecting the patient's privacy may suffer exceptions in particular situations (e.g., in case of emergency) [ABM03], may evolve over time (e.g., depending on the patient's treatment) and may be subject to provisional authorizations [KmS00]. In a data-sharing scenario, the sharing policies change as the initial situation evolves (new relationship between users, new partners or friends, new projects with diverging interest, etc.). Access control languages like XrML [XrM] or ODRL [ODR] demonstrates the need for more expressiveness and flexibility in DRM applications (e.g., Alice may listen freely to a piece of music provided she has listened a given amount of commercial before). Finally, neither Web site nor Internet Service Provider can predict the diversity of access control rules that parents/teachers with different sensibility are willing to enforce (e.g., rules may depends on religious beliefs, political opinions, lecture topics, etc.).

The goal pursued in our study is being able to evaluate dynamic and personalized access control rules on a ciphered input document, with the benefit of dissociating access rights from encryption by embedding the access control in a secured chip. Dynamicity can be obtained by downloading access control policies in the smart card on demand. In addition, the combination of on-board data storage and access control on external data allows for powerful access control models based on historical data, in the spirit of XrML and ODRL.

3 Functional Architecture and Scenarios

After giving preliminaries on XML access control models, this section details the C-SXA architecture and illustrates its utilization through three scenarios, addressing respectively the management of external data, the management of on-board data and a mix of both. Finally, it discusses briefly how the limitations of the smart card can be tackled to make the approach viable.

3.1 XML access control background

Roughly speaking, an XML document can be seen as a tree of *elements*, each one demarcated by an *opening* and *closing tag*. *Attributes* may be attached to elements. Terminal elements are represented by *text*. Simple queries can be expressed over an XML document using the XPath language. Basically, 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 subtree.

Several authorization models have been recently proposed for regulating access to XML documents. We introduce below a simplified access control model for XML, inspired by Bertino's model [BCF01] and Samarati's model [DDP02] that roughly share the same foundation. Subtleties of these models are ignored for the sake of simplicity.

In this simplified model, access control rules take the form of a 3-uple <sign, subject, object>.

Sign denotes either a permission (positive rule) or a prohibition (negative rule) for the read operation. Subject denotes a user or a group of users. 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 paper a rather robust subset of XPath denoted by XP{[],*,//} [MiS02]. 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.

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 this propagation mechanism 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, the path from the document root to this object is granted too (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.

The set of rules attached to a given subject on a given document is called an *access control policy*. This policy defines an authorized view of this document and, depending on the application context, this view may be queried. We consider that queries are expressed with the same XPath fragment as access control rules, namely $XP^{\{[],*,//\}}$. Semantically, the result of a query is computed from the authorized view of the queried document (e.g., predicates cannot be expressed on denied elements even if these elements do not appear in the query result). However, access control rules predicates can apply on any part of the initial document.

3.2 C-SXA architecture

As pictured in Figure 3, the smart card contains the core engine of C-SXA (written in JavaCard), which is composed of four modules: the XML local store, the access right controller, the query evaluator and the security module. On-board data (like for instance a loyalty profile or a personal folder) and the related access control rules are also stored in the smart card stable storage. Applications interact with C-SXA thanks to a C-SXA proxy located on the client device (externally to the smart card). This proxy provides a standard XML API to dialog with the engine in the card, independently of the underlying communication protocol (APDU). External data and their associated access control policies are stored in an encrypted form on an untrusted server (e.g., managed by a DSP).

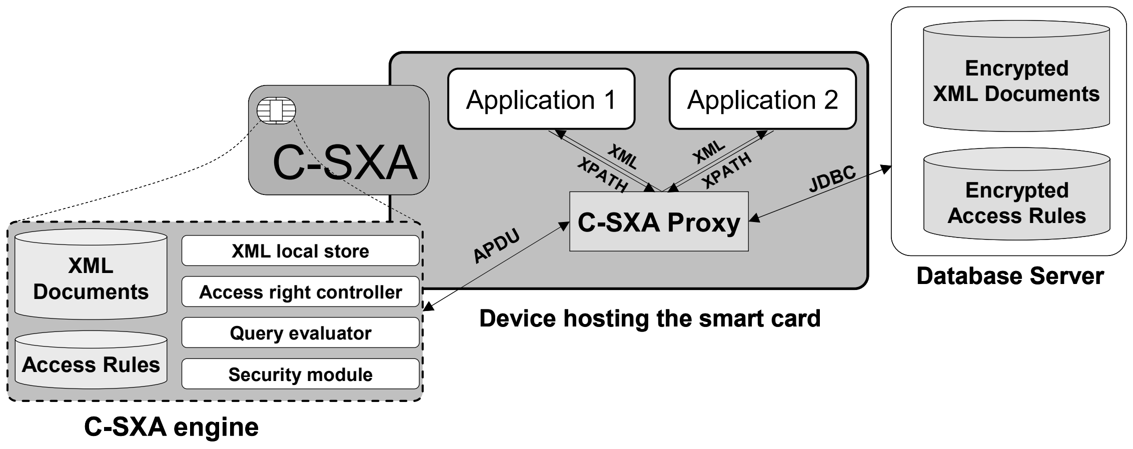


Figure 3: *Target architecture*

Let us consider an application willing to access local user's data. This application will first authenticate to the smart card (via the proxy) and will express its need by a simple XPath query. The C-SXA engine processes this query, computes on the fly the authorized part of the requested user's data, and delivers it to the application (via the proxy). Let us now consider an application accessing external data. The communication protocol between the application and the C-SXA proxy remains unchanged, making the data localization transparent. Since the C-SXA proxy uses the JDBC (Java Data Base Connectivity) protocol, the C-SXA server can be any JDBC enabled database system. When the application expresses a query, C-SXA downloads first the access control rules related to the queried data and decrypts them. Then, it downloads the data themselves in a streaming fashion, decrypts them and processes the query in a way similar to the querying of on-board data.

At this point, let us remark that the C-SXA engine, the Proxy and the Server are totally application independent.

3.3 Scenarios

External data scenario

This first scenario deals with collaborative works among a community of users. The objective is to organize a confidential data sharing space via an untrusted DSP to exchange textual XML data like agendas, address books, profiles, working drafts, etc. C-SXA permits to define powerful access control rules while handling rule dynamicity. Indeed, access control rules are likely to evolve while new partners join or leave the community and ad-hoc rules may be defined for particular data (e.g., sensitive appointments in an agenda or financial sections in a working document). Models compiling access control policies in the data encryption cannot tackle these situations accurately.

In the following, we will consider a collaborative agenda for its representativeness of this class of applications. The agenda is actually stored in an encrypted form on a remote server to allow its sharing among partners while protecting the confidentiality of its content. Figures 4 pictures an XML agenda sample in its usual hierarchical representation.

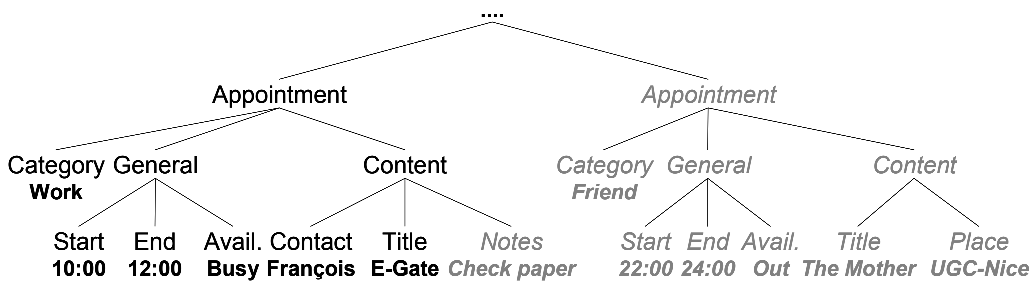


Figure 4: *Hirerarchical representation of an XML agenda*

Let us now consider two users, Alice and Bob, willing to share their remote respective agenda through a cellular phone equipped with a C-SXA-enabled SIM card. Alice will first fix Bob's access control policy thanks to a *rights management GUI*. These rules are translated in XPath expressions by the application and stored in an encrypted form in the remote server. Figure 5 gives an example of access control policy defined for regulating the access to the agenda presented above. This policy states that: by default, nobody is granted access to any appointment (rule R1); users belonging to the group Secretary can access all appointments of category "work" (rule R2), except the notes that appear in the content (rule R3); a colleague can see the appointments of category "work" in which he/she participates (rule R4)[3]; finally, Bob can access all appointments of category "friend" (rule R5). Figure 4 pictures in gray the agenda parts that remain hidden to the secretaries.

[3] Note that *context variables* can be used to increase the expressiveness of the access control

When Bob connects to the server and asks for Alice's agenda, C-SXA downloads Bob's access control policy through a secured channel in Bob's SIM card. Bob can then issue a query on Alice's agenda. C-SXA interprets it, downloads the relevant part of Alice's agenda in a streaming fashion, decrypts it, checks its integrity and evaluates Bob's access control rules to deliver the authorized final result. If Alice is willing to change Bob's access control policy, she uses again the access right management GUI, the new policy is sent to the server and will become active the next time Bob connects to Alice's agenda.

Rule R1:	<PUBLIC,	⊖,	// Appointment >
Rule R2:	<Secretary,	⊕,	// Appointment [Category = "Work"] >
Rule R3:	<Secretary,	⊖,	// Appointment [Category = "Work"] /Content/Notes >
Rule R4:	<Colleague,	⊕,	// Appointment [// Contact = CURRENT_USER] / Content >
Rule R5:	<Cathy,	⊕,	// Appointment [Category = "Friend"] >

Figure 5: *Access control policy rules*

Many different scenarios dealing with external data can be envisioned. Video distribution is an interesting example. A video can be split in sequences, each of which annotated with meta-data describing the sequence. In a parental control application, parents may forbid their child to watch certain sequences containing violent scenes. (e.g., ⊖, //movie//sequence/[type = "violence"]). Video distribution is representative of real-time applications dealing with pushed, huge multimedia data while collaborative agenda is representative of asynchronous applications dealing with pulled, light textual data.

On board data scenario

Secured medical folders have been the focus of many research works. In [ABB01], we proposed to manage on-board medical folders thanks to an embedded relational DBMS. PicoDBMS allows defining different views of the same folder depending on the privileges of the user logged on the smart card (e.g., patient, pharmacist, physician). This first study shown the benefit of smart folders (high level of confidentiality for highly sensitive data) and exacerbated the need for powerful access control models. However, PicoDBMS is dedicated to well-structured folders that can be stored into tables. While tables accommodate well the administrative data, XML turns to be more convenient to manage the medical data themselves due to their much higher irregularity. The need for powerful access control policies is however independent of the data model used to organize the data. This advocates for the design of an

rules, e.g. CURRENT_USER corresponds to the user currently connected to the card.

embedded XML store capable of storing huge and complex XML data and for an access right controller capable of evaluating complex access control policies on them. Unlike the preceding scenario, access control policies are fixed by a medical authority and are rather stable. Thus, they can be installed on the smart card at personalization time (i.e., before the card is actually delivered to the patient). However, access control rules may be contextual (e.g., some rules can be violated in case of emergency [ABM03, KmS00]) and such situations have to be recorded in the smart card stable storage for further controls.

Scenarios involving both on-board and external data

As outlined in the introduction, applications relying on XrML or ODRL like access control models may authorize users to get access to external data, depending on historical or profile information. Storing this information in the smart card is the right way to take advantage of these new powerful access control models while enforcing the privacy of this information and protecting it against tampering.

In a DRM context, a special offer may allow a customer to see some bonus sequences for free providing she previously completes a survey. In this case, as soon as the user fills in the survey, a new entry is appended to the on-board XML repository. The rights defined for the bonus sequences can be written as: (⊕, //Bonus[Card://DRM_RECORD='survey1']/Content), where Card://DRM_RECORD refers to an historical entry stored in the card. When requesting the video, the smart card will first evaluate the predicate on DRM_RECORD and perform the access using this rule.

In an e-learning scenario, high school students may have access to view lessons (videos, texts and audios) which (a) have been previously selected by their teachers and (b) are related to their major. Such rules, though complex, can be expressed in XPath: e.g., (a) ⊕, //lesson[contains(keywords, 'Paris')]; (b) ⊖, //lesson[topic!=%MAJOR%], where %MAJOR% refers to the on-board user profile.

3.4 Scenarios requirements vs. smart card resources

As the scenarios make clear, a powerful embedded XML store and access right controller is highly required. Thus, the question is whether existing smart card platforms are powerful enough to support the complexity of the embedded C-SXA engine. In other words, does the global approach make sense? Currently, our prototype runs on an *e-gate* platform equipped with a 32 bit CPU running at 30Mhz, 4KB of RAM, 32KB of EEPROM and USB communications at 16Kbps, provided by our industrial partner Axalto (the Schlumberger's smart card subsidiary). Although Axalto announces more powerful platforms (32 bit CPU running at 40Mhz, 8KB of RAM, 1MB of Flash and USB communications at 8Mbps) by the end of this year and that this evolution roughly follows the Moore's law, the smart card will remain the bottleneck of the architecture for a while. This bottleneck is mainly due to the decryption cost (of the streaming input document), to the low communication bandwidth (to download the document) and at a lower extent to the memory. The performance measures summarized in Section 5 show however that this bottleneck does not compromise the viability of the aforementioned scenarios, even with existing smart card platforms.

In addition, while performance is an important issue, one must keep in mind that the data protection is the main issue here. As the two next sections will make clear, we devised various techniques to alleviate the performance problem without sacrificing tamper resistance. First we will consider pipeline algorithms based on non-deterministic automata to enforce the access control without consuming much memory and to reduce latency thus improving the communication speed. Moreover, in order to further reduce the communication flow and thus the volume of data to be decrypted, we rely on indexes allowing skipping irrelevant parts of the flow. Second, we focus on how the smart card engine can be integrated in the distributed architecture, including the server and the user terminal, providing further techniques to reduce again the communication flow by splitting the documents in smaller fragments and indexing them.

4 C-SXA engine

4.1 Queries and access control

While several access control models for XML have been proposed recently, few papers address the enforcement of these models and, to the best of our knowledge, no one considers access control in a streaming fashion. At first glance, streaming access control resembles the well-known problem of XPath processing on streaming documents. There is a large body of work on this latter problem in the context of XML filtering [DF03, GMO03, CFG02]. These studies consider a very large number of XPath expressions (typically tens of thousands). The primary goal here is to select the subset of queries matching a given document (the query result is not a concern) and the focus is on indexing and/or combining a large amount of queries. One of the first works addressing the precise evaluation of complex XPath expressions over streaming documents is due to [PfC03] which proposes a solution to deliver parts of a document matching a single XPath. While access control rules are expressed in XPath, the nature of our problem differs significantly from the preceding ones. Indeed, the rule propagation principle along with its associated conflict resolution policies (see Section 3) makes access control rules not independent. The interference between rules introduces two new important issues:

– *Access control rules evaluation:* for each node of the input document, the evaluator must be capable of determining the set of rules that applies to it and for each rule determining if it applies directly or is inherited. The nesting of the access control rules scopes determines the authorization outcome for that node.

– *Access control optimization:* the nesting of rule scopes associated with the conflict resolution policies inhibits the effect of some rules. The rule evaluator must take advantage of this inhibition to suspend the evaluation of these rules and even to suspend the evaluation of all rules if a global decision can be reached for a given subtree.

As streaming documents are considered, we make the assumption that the evaluator is fed by an event-based parser (e.g., SAX [SAX]) raising *open*, *value* and *close* events respectively for each opening, text and closing tag in the input document.

We represent each access control rule (i.e., XPath expression) by a non-deterministic finite automaton (NFA) [HoU79]. Figure 6.b pictures the Access control rules Automata (*ARA*) corresponding to two rather simple access control rules expressed on an abstract XML document. This abstract example, used in place of the motivating example introduced in Section 3, gives us the opportunity to study several situations (including the trickiest ones) on a simple document. In our ARA representation, a circle denotes a state and a double circle a final state, both identified by a unique *StateId*. Directed edges represent transitions, triggered by *open* events matching the edge label (either an element name or *). Thus, directed edges represent the child (/) XPath axis or a wildcard depending on the label. To model the descendant axis (//), we add a self-transition with a label * matched by any *open* event. An ARA includes one *navigational path* and optionally one or several *predicate paths* (in gray in the figure). To manage the set of ARA representing a given access control policy, we introduce the following data structures:

– *Tokens and Token Stack:* we distinguish between navigational tokens (NT) and predicate tokens (PT) depending on the ARA path they are involved in. To model the traversal of an ARA by a given token, we actually create a token proxy each time a transition is triggered and we label it with the destination StateId. The terms token and token proxy are used interchangeably in the rest of the paper. The navigation progress in all ARA is memorized thanks to a unique stack-based data structure called Token Stack. The top of the stack contains all active NT and PT tokens, i.e. tokens that can trigger a new transition at the next incoming event. Tokens created by a triggered transition are pushed in the stack. The stack is popped at each *close* event. The goal of Token Stack is twofold: allowing a straightforward backtracking in all ARA and reducing the number of tokens to be checked at each event (only the active ones, at the top of the stack, are considered).

– *Rule status and Authorization Stack:* Assume for the moment that access control rule expressions do not exploit the descendant axis (no //). In this case, a rule is said to be active, – meaning that its scope covers the current node and its subtree – if all final states of its ARA contain a token. A rule is said pending if the final state of its navigational path contains a token while the final state of some predicate path has not yet been reached, i.e., the rule depends on predicates (called pending predicates) which occur later in the parsing. The Authorization Stack registers the NT tokens having reached the final state of a navigational path, at a given depth in the document. The scope of the corresponding rule is bounded by the time the NT token remains in the stack. This stack is used to solve conflicts between rules. The status of a rule present in the stack can be fourfold: positive-active (denoted by ⊕), positive-pending (denoted by ⊕?), negative-active (denoted by ⊖), negative-pending (denoted by ⊖?). By convention, the bottom of the stack contains an implicit negative-active rule materializing a closed access control policy (i.e., by default, the set of objects the user is granted access to is empty).

– *Rule instances materialization:* Taking into account the descendant axis (//) in the access control rules expressions makes things more complex to manage. Indeed, the same element names can be encountered at different depths in the same document, leading several tokens to reach the final state of a navigational path and predicate paths in the same ARA, without being related together[4]. To tackle this situation, we label navigational and predicate token

[4] The complexity of this problem has been highlighted in [PfC03].

proxies with the depth at which the original predicate token has been created, materializing their participation in the same rule instance[5]. Consequently, a token (proxy) must hold the following information: RuleId (denoted by R, S, …), Navigational/Predicate status (denoted by n or p), StateId and Depth[6]. For example, $Rn2_2$ and $Rp4_2$ (also noted 2_2, 4_2 to simplify the figures) denotes the navigational and predicate tokens created in Rule R's ARA at the time element b is encountered at depth 2 in the document. If the transition between states 4 and 5 of this ARA is triggered, a token proxy $Rp5_2$ will be created and will represent the progress of the original token $Rp4_2$ in the ARA. All these tokens refer to the same rule instance since they are labeled by the same depth. A rule instance is said active or pending under the same condition as before, taking into account only the tokens related to this instance.

– *Predicate Set:* this set registers the PT tokens having reached the final state of a predicate path. A PT token, representing a predicate instance, is discarded from this set at the time the current depth in the document becomes less than its own depth.

Stack-based data structures are well adapted to the traversal of a hierarchical document. However, we need a direct access to any stack level to update pending information and to allow some optimizations detailed below. Figure 6.c represents an execution snapshot based on these data structures. This snapshot being almost self-explanatory, we detail only a small subset of steps.

- Step 2: the *open* event *b* generates two tokens $Rn2_2$ and $Rp4_2$, participating in the same rule instance.

- Step 3: the ARA of the negative rule S reaches its final state and an active instance of S is pushed in the Authorization Stack. The current authorization remains negative. Token $Rp5_2$ enters the Predicate Set. The corresponding predicate will be considered true until level 2 of the Token Stack is popped (i.e., until event */b* is produced at step 9). Thus, there is no need to continue to evaluate this predicate in this subtree and token $Rp4_2$ can be discarded from the

---

[5] To illustrate this, let us consider the rule R and the right subtree of the document presented in Figure 6. The predicate path final state *5* (expressing //b[c]) can be reached from two different instances of b, respectively located at depth 2 and 3 in the document, while the navigational path final state *3* (expressing //b/d) can be reached only from b located at depth 3. Thus, a single rule instance is valid here, materialized by navigational and predicate tokens proxies labeled with the same depth 3.

[6] If a same ARA contains different predicate paths starting at different levels of the navigational path, a NT token will have in addition to register all PT tokens related to it.

Token Stack.

- Step 5: An active instance of the positive rule R is pushed in the Authorization Stack. The current authorization becomes positive, allowing the delivery of element *d*.
- Step 16: A new instance of R is pushed in the Authorization Stack, represented by token *Rn3₃*. This instance is pending since the token *Rp5₂* pushed in the Predicate Set at step 12 (event *c*) does not participate in the same rule instance.
- Step 18: Token *Rp5₃* enters the Predicate Set, changing the status of the associated rule instance to positive-active.

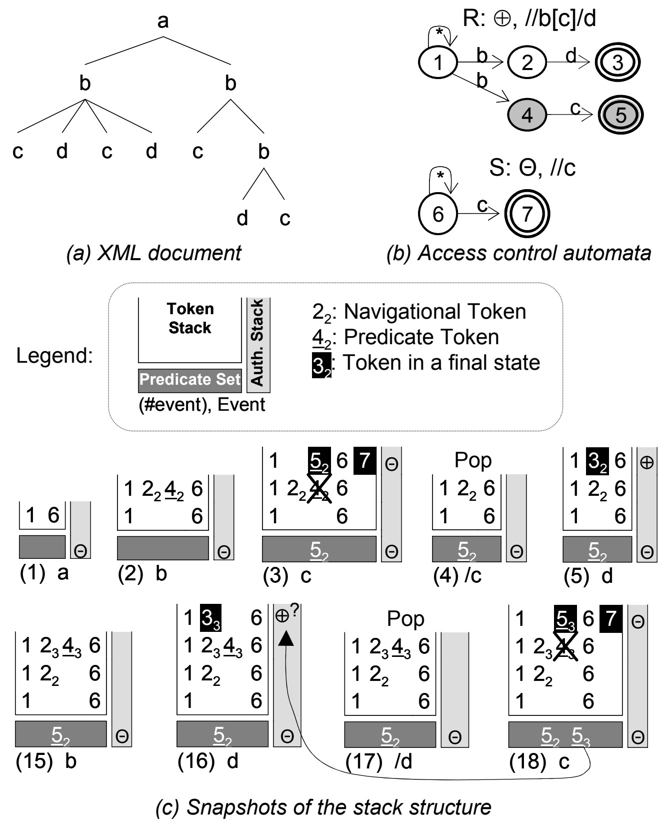


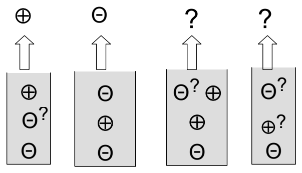
Figure 6: *Execution snapshot*

4.2 Conflict Resolution

From the information kept in the Authorization Stack, the outcome of the current document node can be easily determined. The conflict resolution algorithm presented in Figure 7 integrates the closed access control policy (line 1), the *Denial-Takes-Precedence* (line 2) and *Most-Specific-Object-Takes-Precedence* (lines 5 and 7) policies to reach a decision. 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. Recursion captures the fact that a decision may be reached even if the rules at the top of the stack are pending, depending on the rule status found in the lower stack levels. Note, however, that the decision can remain pending if a pending rule at the top of the stack conflicts with other rules. In that case, the current node has to be buffered, waiting for a delivery condition. This issue is tackled in [BDP04b]. The rest of the algorithm is self-explanatory and examples of conflict resolutions are given in the figure.

```
DecideNode(depth) → Decision ∈ {⊕, ⊖,?}

If depth = 0 then  return '⊖'
 elseif '⊖'∈ AS[depth].RuleStatus then return '⊖'
  elseif '⊕' ∈ AS[depth].RuleStatus and
     '⊖?' ∉ AS[depth].RuleStatus then return '⊕'
   elseif DecideNode(depth - 1) = '⊖' and
     ∀t∈{'⊕?','⊕'} t∉ AS[depth].RuleStatus then return '⊖'
    elseif DecideNode(depth - 1) = '⊕' and
       '⊖?' ∉ AS[depth] RuleStatus then return '⊕'
     else return '?'
```



Examples of conflict resolution

Figure 7: *Conflict resolution algorithm*

The DecideNode algorithm presented below considers only the access control rules. Things are slightly more complex if queries are considered too. Queries are expressed in XPath and are translated in a non-deterministic finite automaton in a way similar to access control rules. However, a query cannot be regarded as an access control rule at conflict resolution time. The delivery condition for the current node of a document becomes twofold: (1) the delivery decision must be true and (2) the query must be interested in this node. The first condition is the outcome of the DecideNode algorithm. The second condition is matched if the query is *active*, that is if all final states of the query ARA contain a token, meaning that the current node is part of the query scope.

4.3 Optimization issues

A possible optimization is to suspend dynamically the evaluation of ARA that become irrelevant or useless inside a subtree. The knowledge gathered in the Token Stack, Authorization Stack and

Predicate Set can be exploited to this end. The first optimization is to suspend the evaluation of a predicate in a subtree as soon as an instance of this predicate has been evaluated to true in this subtree. This optimization has been illustrated by Step 3 of Figure 6.c. The second optimization is to evaluate dynamically the containment relation between active and pending rules and take benefit of the elimination condition mentioned above. From the Authorization Stack, we can detect situations where the following local condition holds: $(T \subset S \subseteq R) \wedge (R.Sign=S.Sign \wedge S.Sign \neq T.Sign)$, the stack levels reflecting the containment relation inside the current subtree. S can be inhibited in this subtree. If stopping the evaluation of some ARA is beneficial, one must keep in mind that the two limiting factors of our architecture are the decryption cost and the communication cost. Therefore, the real challenge is being able to take a common decision for complete subtrees, a necessary condition to detect and skip prohibited subtrees, thereby saving both decryption and communication costs.

Without any additional information on the input document, a common decision can be taken for a complete subtree rooted at node *n* iff: (1) the DecideNode algorithm can deliver a decision D (either $\oplus$ or $\ominus$) for *n* itself and (2) a rule R whose sign contradicts D cannot become active inside this subtree (meaning that all its final states, of navigational path and potential predicate paths, cannot be reached altogether). These two conditions are compiled in the algorithm presented in Figure 8. In this algorithm, AS denotes the Authorization Stack, TS the Token Stack, TS[i].NT (resp. TS[i].PT) the set of NT (resp. PT) tokens registered at level *i* in this stack and top is the level of the top of a stack. In addition, t.RuleInst denotes the rule instance associated with a given token, Rule.Sign the sign of this rule and Rule.Pred a Boolean indicating if this rule includes predicates in its definition.

The immediate benefit of this algorithm is to stop the evaluation for any active NT tokens and the main expected benefit is to skip the complete subtree if this decision is $\ominus$. Note however that only NT tokens are removed from the stack at line 6. The reason for this is that active PT tokens must still be considered, otherwise pending predicates could remain pending forever. As a conclusion, a subtree rooted at *n* can be actually skipped iff: (1) the decision for *n* is $\ominus$, (2) the DecideSubtree algorithm decides $\ominus$ and (3) there are no PT token at the top of the Token Stack (which turns to be empty). Unfortunately, these conditions are rarely met together, especially when the descendant axis appears in the expression of rules and predicates. The next section introduces a Skip index structure that gives useful information about the forthcoming content of the input document. The goal of this index is to detect a priori rules and predicates that will become irrelevant, thereby increasing the probability to meet the aforementioned conditions.

```
DecideSubtree() → Decision ∈ {⊕, ⊖,?}

    D = DecideNode(AS.top)
    if D = '?' then return '?'
    if not (∃ nt ∈TS[top].NT / nt.Rule.Sign ≠ D
           and (not nt.Rule.Pred
                or (∃ pt ∈TS[top].PT / pt.RuleInst = nt.RuleInst))
        then TS[top].NT = ∅; return (D)
   else return '?'
```

Figure 8: *Decision on a complete subtree*

When queries are considered, any subtree not contained in the query scope is candidate to a skip. This situation holds as soon as the NT token of the query (or NT tokens when several instances of the same query can co-exist) becomes inactive (i.e., is no longer element of TS[top].NT). This token can be removed from the Token Stack but potential PT tokens related to the query must still be considered, again to prevent pending predicate to remain pending forever. As before, the subtree will be actually skipped if the Token Stack becomes empty.

5 Distributed infrastructure

This section focuses on the way the C-SXA engine can be integrated in a distributed architecture including the smart card, the server and the user terminal. This integration is an important issue, considering the fact that the tamper resistance of the access control relies not only on the smart card but also on the whole environment (e.g., communication protocol, access rights update protocol, etc.). This issue has not been deeply discussed in [BDP04b], where the focus was put on the components embedded in the smart card.

On-board storage model

The main requirements of the storage model are first, to provide a compact representation of the XML data (while increasing, the smart card stable storage capacity is still limited) and second to offer an efficient scheme to converge quickly towards the authorized fragments of these data. In [BDP04a] we proposed a recursive indexation structure primarily designed to skip the irrelevant part of a streaming external document. We explain below how this structure has been adapted to serve as a compact indexed storage model for the on-board data.

The document structure is compressed thanks to a dictionary of element names, also called *tags* [ABC04]. In most cases, the dictionary may be disclosed since it is generally not confidential.

Then, we append to every XML element, the size of its subtree as well as a bitmap representing all the tags present in its subtree. The bitmap is used to decide whether a rule may or not be active in the subtree (e.g., the rule //a//c cannot apply on a subtree containing only the tags a and d). In some situations stated in Section 4.3, the subtree can be skipped thanks to the size attribute. This extra information is compressed recursively using the following basic idea. For each node, the corresponding set of tags is encoded as a bit array referring to its parent set of tags and the subtree size is encoded on the number of bits required to encode its parent subtree size.

External storage model

In order to reduce the communication and decryption costs, a document is split into fragments. When a client issues a query, only the fragments relevant for this query are downloaded. The scope and size of fragments are determined by the application. To illustrate this, in the Agenda application, each fragment corresponds to a particular day.

In our prototype the remote storage is delegated to a relational database server (MySQL). The main tables are represented in Figure 9. Columns pictured in gray are encrypted thanks to the owner secret key (see next subsection). The XPath column is a (compressed) XPath expression identifying each fragment of a document, itself identified by (owner, type). Type and XPath can be stored in plaintext provided they do not divulgate confidential information[7]. Note that the XPath (for documents) and RuleId (for rules) are stored both in plaintext and encrypted form in order to allow the smart card checking the correspondence between client request and server answer. Otherwise, malicious users could substitute encrypted data to gain access to unauthorized one.

DOCUMENTS

Owner	Type	XPath	N-gram	Encrypted data
Philippe	Agenda	2[**2004**] 3[**03**] 4[**18**]	1100111000100	1 2[2004] 3[03] 4[18] 5 6[Work] 7 8[08:00] 9[12:00] 10[Busy] 11 12[Luc] 13[ACI … 5 6[Work] 7 8[14:00] 9[17:00] 10[Busy] 11 12[François] …..
Philippe	Agenda	2[**2004**] 3[**03**] 4[**19**]	0100011101110	1 2[2004] 3[03] 4[19] 5 6[Work] 7 8[08:00] 9[12:00] 10[Busy] 11 12[Luc] 13[ACI … 5 6[Perso] 7 8[22:00] 9[24:00] 10[Out] 11 15[Cinema] …
Philippe	Agenda	2[**2004**] 3[**03**] 4[**20**]	1100111000111	1 2[2004] 3[03] 4[20] 5 6[Perso] 7 8[10:00] 9[12:00] 10[Out] 11 15[Sport] 13[… 5 6[Perso] 7 8[13:00] 9[16:00] 10[Out] 11 15[Museum] …

RULES

Owner	Type	Grantee	RuleId	Encrypted rule
Philippe	Agenda	Luc	7	7 ⊕ : //Appointment[//Contact = "Luc"]/Content
Philippe	Agenda	Luc	8	8 ⊕ : //Appointment/General

TAG DICTIONNARY

TagId	Tag
1	Agenda
2	Year
3	Month
….	…

Figure 9: *Data stored on the server*

[7] Potential inference on the external data can be avoided by inserting dummy appointments

To enable keyword searches on a document without having to download all its fragments, each fragment is indexed using n-grams [HIL02]. We consider a set of n-grams denoted by N={n1, n2, …} where n1, n2 are strings of at most n characters. Based on this set of n-grams, the fragment index value is defined as a bitmap where each bit corresponds to an n-gram in N. A bit is set if the corresponding n-gram is present in the fragment. For instance, let N be {“te”, “ta”, “se”} and let a fragment F contain the text “white house”, the index value of F is 101. To make a keyword search, a bitmap value V is computed from the keywords and is compared to every fragment index values using the AND logical operator. If the result equals V, then the fragment is a potential candidate and is thus downloaded. For example, if the word “house” is searched, V equals 001. The result of V AND 101 equals V, so the fragment may contain the word “house”. Let us now see how this index is actually exploited. Upon receiving a query, the terminal first computes the bitmap value V based on a set of public n-grams stored locally. This value is sent to the smart card which encrypts it using a secret key. The resulting value EV is then sent to the server which compares it to all its index values (previously encrypted with the same key). The server then returns the candidate fragments of the document. The overhead incurred by this index is rather small, the smart card doing only the encryption of the index value, the computation of the index on the terminal being negligible.

Confidentiality and integrity

In our context, the attacker can be the user himself. For instance, a user being granted access to an appointment X may try to extract unauthorized information from an appointment 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 [Sch96]), identical plaintext blocks will generate identical ciphered values. In that case, the attacker can conduct different attacks: substituting some blocks of appointments 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., X) and using it to derive unauthorized information from ciphertext (e.g., 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 appointments starting after 6 PM and he randomly alters the ciphertext storing the starting time).

To face these attacks, we exploit two techniques. Encryption and hashing are required to guarantee respectively the confidentiality and the integrity of external documents and access control rules. Standard integrity checking methods must be adapted to tackle the memory limitation of the smart card. This imposes to implement integrity checking in a streaming fashion.

Regarding encryption, the objective is to generate different ciphertexts for different instances of a same value. This property is 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 [Sch96].

Regarding integrity checking, the document is split into chunks whose size is determined by the memory capacity of the smart card. Each chunk contains an encrypted ChunkDigest (computed using a collision resistant hash function (e.g., SHA-1). To avoid chunk substitution or removal, a chunk digest is computed over the union between the chunk content itself and the digest of the preceding chunk (see figure 10.a). The document is thus protected against tampering and confidentiality attacks while remaining agnostic regarding the encryption algorithm used to cipher the elementary data.

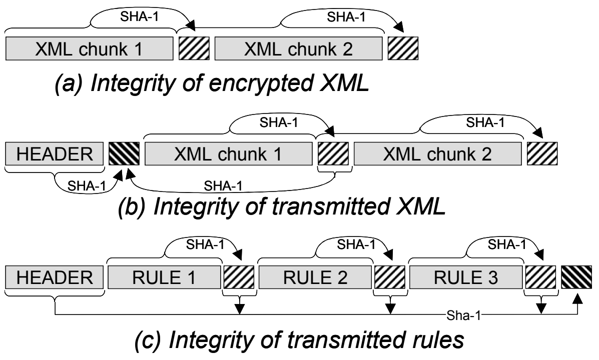


Figure 10: *Integrity*

Key distribution

Separating data encryption from access control allows using potentially a single encryption key for all documents and access control rules present in the system. However, to increase security and robustness, several keys can be used, e.g., one per user. In this case, when Alice wants to share (part of) her document with Bob, Alice's secret key must be transmitted to Bob's smart card. A secure key distribution is implemented by means of a classical Public Key Infrastructure (PKI).

Secured communications

Messages transmitting documents and rules have to be protected against tampering (e.g., Bob may try to drop all negative rules transmitted by the server). To avoid checking integrity twice (at the communication level and at the rule/document chunk level), the message digest is computed on the message header and on the rule/document digest, as pictured in Figure 10.b and c. Finally, an increasing counter is added to each message to prevent *replay attacks* (e.g., Bob may try to substitute a rule update message with an old one containing no negative rules).

Performance assessment

On-board data management is obviously less expensive than external data management since the latter incurs an extra communication and decryption cost and may apply on larger documents. This section thus focuses on the management of external data. The discussion below summarizes the results of performance evaluations conducted both on the JavaCard e-gate platform described in Section 3.4 (30Mhz CPU, 4KB of RAM, 32KB of EEPROM and 16Kbps USB communications) and on a cycle-accurate hardware simulator provided by Axalto simulating faithfully the behavior of forthcoming smart card platforms (40Mhz CPU, 8KB of RAM, 1MB of Flash and 8Mbps USB communications). The prototype running on the cycle-accurate simulator has been rewritten in C, allowing predicting exactly the performance of the C-SXA engine on a real (i.e., industrial) setting.

The experiments related to the C prototype have been conducted both on real and synthetic data sets, each of which exhibiting different characteristics (shallow vs. deep documents, few vs. many distinct tags, non-recursive vs. recursive documents) and apply access control rules of varying complexity (non-recursive vs. recursive, few vs. many predicates). We showed that the access control time, representing 5% to 15% of the total response time is negligible wrt the decryption time (50% to 60%) and the communication time (about 30%). We showed that our method tackles well very different situations and produces a throughput ranging from 55KB/s to 85KB/s depending on the document and the access control policy. These preliminary results as encouraging when compared with xDSL Internet bandwidth available nowadays (ranging from 16KB/s to 128KB/s). The complete performance evaluation can be found in [BDP04a].

The performance of our JavaCard e-gate prototype has been assessed on two applications: a collaborative agenda and a video application. The first one implements the application as described in Section 3 and exhibits acceptable performance, e.g., retrieving appointments of a single day from the server takes few seconds. However, the e-gate smart card platform shows its limits when dealing with large objects, especially when response times have to be congruent with real time constraints (e.g., in video applications). In this context the limiting factors are the decryption time and above all the communication throughput. To solve this problem, we trade security for performance as follows. We consider videos encoded using the MPEG7 standard, which allows storing short descriptions of the scenes in the XML metadata. On the server, the metadata are stored encrypted and the video sequences are encrypted according to secret keys stored in that metadata. The smart card performs the access control on the XML metadata and delivers the decryption keys to the media player, according to the user's privileges (see Figure 11). The media player then downloads, decrypts and displays the authorized sequences (i.e., the ones for which it got the proper decryption keys). Since the decryption of the video sequences is performed on the terminal, video sequences can be played in real time and only a short latency (few seconds) is necessary at the beginning to process the metadata in the smart card.

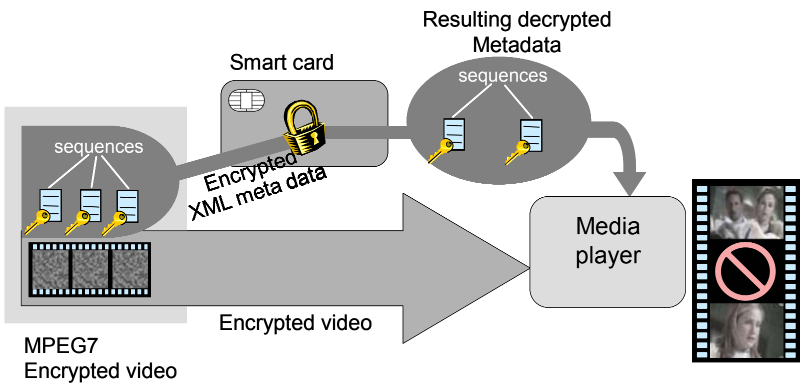


Figure 11: *Parental control*

6 Conclusion

In this paper, we proposed a smart card based architecture to enforce powerful and dynamic access control policies on confidential data and to guarantee the tamper resistance of these data. We showed that this architecture can be used in various contexts, such as secured personal folders, collaborative works, parental and teacher control and advanced DRM applications. Combining on-board XML storage and access control in a smart card provides important benefits, like the definition of different authorized views of the on-board data for different users and the definition of powerful access control policies where rules are defined on historical (or any other contextual) on-board data.

To tackle the drastic hardware constraints of the smart card, we proposed: (i) efficient algorithms to evaluate access control rules on streaming XML data, (ii) a highly compact storage and indexation model for both on-board and external data enabling and (iii) ad-hoc encryption and integrity control mechanisms preventing tampering and confidentiality attacks. Finally we assessed the viability of our approach through experiments conducted on current and future smart card platforms. Current smart cards are still limited and cannot match strong response time constraints but, as STMicroelectronics announces, more powerful smart cards capable of managing video streams in real-time are currently being developed. They will break the current smart card limitations and thus give a new dimension to the C-SXA approach. Although our architecture relies today on a smart card, it can accommodate any other form of hardware secured device (e.g., chip secured set-top-boxes, smart tokens, secured co-processors).

7 References

[ABB01] N. Anciaux, C. Bobineau, L. Bouganim, P. Pucheral, P. Valduriez, "PicoDBMS: Validation and Experience", 27th Int. Conf. on Very Large Data Bases (VLDB), demo session, 2001.

[ABC04] A. Arion, A. Bonifati, G. Costa, S. D'Aguanno, I. Manolescu, A. Puglies, "Efficient Query Evaluation over Compressed Data", EDBT, 2004.

[ABM03] A. El Kalam, S. Benferhat, A. Miege, R. Baida, F. Cuppens, C. Saurel, P. Balbiani, Y. Deswarte, G. Trouessin, "Organization based access control", IEEE 4th Int. Workshop on Policies for Distributed Systems and Networks, 2003.

[AKS02] A. Agrawal, J. Kiernan, R. Srikant, Y. Xu, "Hippocratic Databases", 28th Int. Conf. on Very Large Data Bases, 2002.

[BCF01] E. Bertino, S. Castano, E. Ferrari, "Securing XML documents with Author-X", IEEE Internet Computing, 2001.

[BDP04a] L. Bouganim, F. Dang Ngoc, P. Pucheral, "Client-Based Access Control Management for XML Documents", 30th Int. Conf. on Very Large Data Bases (VLDB), 2004.

[BDP04b] L. Bouganim, F. Dang Ngoc, P. Pucheral, "Client-Based Access Control Management for XML Documents", INRIA internal report, June 2004.

ftp://ftp.inria.fr/INRIA/tech-reports/RR/RR-5282.ps.gz

[BoP02] L. Bouganim, P. Pucheral, “Chip-Secured Data Access: Confidential Data on Untrusted Servers”, 28th Int. Conf. on Very Large Data Bases (VLDB), 2002.

[Car99] L. C. Carrasco, “RDBMS’s for Java Cards? What a Senseless Idea!”, 1999. http://www.sqlmachine.com

[CFG02] C Chan, P. Felber, M. Garofalakis, R. Rastogi, “Efficient Filtering of XML Documents with Xpath Expressions”, ICDE, 2002.

[DDP02] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati, “A Fine-Grained Access Control System for XML Documents”, ACM TISSEC, vol. 5, n. 2, 2002.

[DF03] Y. Diao, M. Franklin, “High-Performance XML Filtering: An Overview of YFilter”, ICDE, 2003.

[DH04] I. Duthie, B. Hochfield, “Secure Megabytes on your Smart Card”, Card Forum International, January 2004.

[FBI03] Computer Security Institute, “CSI/FBI Computer Crime and Security Survey”, http://www.gocsi.com/forms /fbi/pdf.html.

[Gem04] Gemplus. Smart Times - Multi-Chip Smart Cards, http://www.gemplus.com/smart/ enews/st3/sumo.html

[GMO03] T. Green, G. Micklau, M. Onizuka, D. Suciu, “Processing XML streams with Deterministic Automata”, ICDT, 2003.

[HeW01] J. He, M. Wang, “Cryptography and Relational Database Management Systems”, IDEAS, 2001.

[HIL02] H. Hacigumus, B. Iyer, C. Li, S. Mehrotra, “Executing SQL over encrypted data in the database-service-provider model”, ACM SIGMOD, 2002.

[HoU79] J. Hopcroft, J. Ullman, “Introduction to Automata Theory, Languages and Computation”, Addison-Wesley, 1979.

[IBM] IBM-Harris Multinational Consumer Privacy Survey: http://www.pco.org.hk/english/infocentre/files/westin.doc.

[ISO99] International Standardization Organization (ISO), “Integrated Circuit(s) Cards with Contacts - Part 7: Interindustry Commands for Structured Card Query Language-SCQL”, ISO/IEC 7816-7, 1999.

[KLL+01]J. Karlsson, A. Lal, C. Leung, T. Pham, “IBM DB2 Everyplace: A Small Footprint Relational Database System”, International Conference on Data Engineering (ICDE), 2001.

[KmS00] M. Kudo, S. Hada, “XML document security based on provisional authorization”, ACM CCS, 2000.

[Mas02] MasterCard, “MasterCard Open Data Storage (MODS)”, 2002. https://hsm2stl101.mastercard.net/public/login/ebusiness/smart_cards/one_smart_card/biz_opportunity/mods.

[MiS02] G. Miklau and D. Suciu, “Containment and equivalence for an XPath fragment”, ACM PODS, 2002.

[MiS03] G. Micklau, D. Suciu, "Controlling Access to Published Data Using Cryptography", 29th Int. Conf. on Very Large Data Bases (VLDB), 2003.

[Mop04] The Mopass Card, http://www.mopass.info/english/index.html.

[ODR] The Open Digital Rights Language Initiative, http://odrl.net/.

[Ora02] Oracle Corporation, “Oracle 9i Lite - Oracle Lite SQL Reference”, Oracle Documentation, 2002.

[Ora04] Oracle Advanced Security Administrator's Guide 10g Release 1 (10.1). http://otn.oracle.com/pls/db10g/db10g.to_pdf?pathname=network.101%2Fb10772.pdf &remark=portal+%28Administration%29.

[PBV01] P.Pucheral, L. Bouganim, P. Valduriez, C. Bobineau, “PicoDBMS: Scaling down Database Techniques for the Smart card”, Very Large Data Bases Journal (VLDBJ), Vol.10, n°2-3, 2001.

[PfC03] F. Peng, S. Chawathe, “XPath Queries on Streaming Data”, ACM SIGMOD, 2003.

[PIC] W3C consortium, “PICS: Platform for Internet Content Selection”, http://www.w3.org/PICS.

[Rus01] Ryan Russel et al., “Hack Proofing Your Network”, Syngress Publishing, 2001.

[SAX] Simple API for XML, http://www.saxproject.org/.

[Sch96] B. Schneier, “Applied Cryptography”, 2nd Edition, John Wiley & Sons, 1996.

[Ses00] P. Seshadri, P. Garrett, “SQLServer For Windows CE – A Database Engine for Mobile and Embedded Platforms”, International Conference on Data Engineering (ICDE), 2000.

[Sky] SkyDesk: @Backup (Storage Service Provider). http://www.backup.com/index.htm

[Sma] The Smartright content protection system. http://www.smartright.org/

[Syb00] Sybase Inc., “The Next Generation Database for Embedded Systems”, White Paper, 2000.

[W3C] http://www.w3.org/XML.

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

.