Int. J. Inf. Secur. (2007) 6:255–284
DOI 10.1007/s10207-007-0020-3

REGULAR CONTRIBUTION

A system for securing push-based distribution of XML documents

Elisa Bertino · Elena Ferrari · Federica Paci · Loredana Parasiliti Provenza

Received: 13 November 2006 / Published online: 16 March 2007
© Springer-Verlag 2007

Abstract Push-based systems for distributing information through Internet are today becoming more and more popular and widely used. The widespread use of such systems raises non trivial security concerns. In particular, confidentiality, integrity and authenticity of the distributed data must be ensured. To cope with such issues, we describe here a system for securing push distribution of XML documents, which adopts digital signature and encryption techniques to ensure the above mentioned properties and allows the specification of both signature and access control policies. We also describe the implementation of the proposed system and present an extensive performance evaluation of its main components.

Keywords Push-based data dissemination · XML · Security policies · Authentication

E. Bertino
CERIAS and CS & ECE Departments, Purdue University, West Lafayette, IN 47907-2086, USA
e-mail: bertino@cerias.purdue.edu

E. Ferrari
Department of Computer Science and Communication, University of Insubria, Via Mazzini, 5, 21100 Varese, Italy
e-mail: elena.ferrari@uninsubria.it

F. Paci · L. Parasiliti Provenza (✉)
DICO, University of Milan, Via Comelico, 39/41, 20135 Milano, Italy
e-mail: parasiliti@dico.unimi.it

F. Paci
e-mail: paci@dico.unimi.it

1 Introduction

Push-based distribution of digital information is today widely used to transmit information over the web [1,7,12,20,45]. According to such an approach, a data dissemination service periodically or whenever the information source is updated, broadcasts the same information to all users subscribed to the service. Of course, in some environments the information to be broadcast requires protection from security and privacy threats. First of all, access control is an essential service that has to be ensured in distributing sensitive information. Additionally, the authenticity of the released data is also a crucial need, especially when data are disseminated through the Internet. Though several models, languages, and mechanisms have been developed by the research community to control the access to sensitive information [9,10,19], for what concerns authenticity [35,40], research is mainly focused on digital signature techniques. No tools or policy languages have been devised supporting the specification of which subjects have to sign which portions of some data. To address this lack, we have proposed [8] an XML-based policy language, specifically tailored to XML documents, which allows one to specify two different kinds of security policies: conventional access control policies, stating that a subject can (or cannot) exercise an access privilege on a data item or a part of it, and a new kind of security policies, called *signature policies*, stating the duty one or more subjects have to sign a data item or a part of it. To ensure the security requirements expressed by the specified policies, in [8] we have also proposed a framework that adopts selective symmetric encryption [7], in order to enforce access control, and digital signature tools to meet strong authenticity requirements [40].

Encryption and digital signature operations can be performed according to two alternative strategies, namely *encrypt-then-sign* and *sign-then-encrypt*, which differ on the order according to which the two operations are executed. Combining encryption and signature operations must be carefully managed in that a subject must be able to verify the signature even if s/he can access only selected portions within an XML document.

In this paper, we present the detailed system architecture and implementation we have made of the high level framework proposed in [8]. In particular, the prototype system exploits XML to represent all the information needed for authentication and access control. To represent the encryption and signatures affixed on the various document portions we adopt the `XML Encryption` and `XML Signature` W3C standards [43,44], which allow one to respectively express in an XML format any ciphered datum (even not XML data) along with the information useful for its decryption and the value of a signature affixed on any data item together with all the information for a correct signature validation (i.e., the digest and signature algorithms, the signer public key). However, it is necessary to extend the two standards in order to meet the requirements of our system (e.g., a finer encryption granularity). Another important issue considered in this paper is key management. This aspect must be carefully considered in that the use of selective encryption may cause the management of a high number of keys. To minimize the number of keys to be distributed to the authorized subjects we adopt a key management scheme recently introduced in [2] for key management in access hierarchies and we show in the paper how it can be customized to our context.

Moreover, in this paper we report the results of the extensive analysis we have conducted in order to evaluate the performance of the entire system, and, more specifically, of the two strategies: *encrypt-then-sign* and *sign-then-encrypt*. First of all, we identify the parameters that affect the efficiency of each system component. Then, the performance results we have obtained through several experiments are analyzed to assess the performance of the system for different workloads, which are the parameters that most affect performance, and which strategy performs better and under which conditions. We believe that this is a key result to assess the feasibility of adopting our system in real world push-based dissemination environments.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 briefly describes the system for securing push-based distribution of XML documents proposed in [8]. Besides discussing the motivations for our work, it describes the key features of the system, that is, the policy language adopted by the system and the system architecture. Section 4 presents the key management scheme we have applied to optimize the key management process. Section 5 describes in details the implementation of the system, and discusses the extensions we have made to the `XML Encryption` and `XML Signature` W3C standards. Section 6 presents the analysis study we have carried out to estimate the performance of the system. Finally, Sect. 7 concludes the paper.

2 Related work

The work presented in this paper has strong relationships with security solutions for push-based data dissemination systems [6,7,12,45]. However, several techniques recently proposed for third-party scenarios [5,15,16,28] for pull-based distribution of sensitive information are also relevant to our work since they could be applied also with data distribution paradigms different from the pull one.

In what follows, we first review the research proposals in the area of push-based distribution of digital information, focusing on those addressing security issues. Then, we discuss some security solutions proposed for third-party scenarios.

2.1 Push-based distribution

Given the recent advances in network technologies—wireless and cable networks and high-bandwidth satellite—information push imposes itself as a valid and efficient alternative with respect to the pull mode when the same data have to be frequently released to a large number of users.

Over the last years, in the area of broadcast transmission many research issues have been addressed, in particular those related to architectural and data structure aspects [1]. Some proposals deal with broadcast content scheduling, others with data structures and update or concurrency control. However, only recently, a great attention has been devoted to ensure security properties [6–8,12,20,45].

Most of these proposed approaches address the problem of access control that is crucial when information is disseminated according to the push distribution strategy. In such a case, indeed, the straightforward view-based techniques for access control, widely used in case of pull distribution, are not efficient, since the high number of subjects to which the document has to be frequently broadcast could lead to the creation, management, and distribution of a high number of views. A recent alternative solution [7] exploits encryption techniques [20] to efficiently enforce access control requirements when broadcasting sensitive information over the Web to a large community of users. It encrypts different portions of the same document with different encryption keys, on the basis of the specified access control policies. Then, the same encrypted copy of the document is broadcast to all users; each user only receives the key(s) corresponding to the portion(s) s/he is enabled to access. In our system, we use the same idea and combine it with digital signature techniques to enforce authenticity in addition to confidentiality requirements. Further work [2,4,5,12,45] adopts more flexible symmetric key assignment schemes in order to minimize the number of encryption keys that have to be generated and distributed.

In [2], Atallah et al. propose a dynamic key management scheme based on access hierarchies, which we apply to our context to reduce the number of secret keys to be distributed to each user to correctly decrypt the authorized portions.

A cryptography-based solution, known as Cryptolope^{TM}, has been also proposed by IBM [24] to ensure access control when chunks of information—ranging from high-value business reports to graphics, videos and softwares—are distributed online, on diskette, or even are broadcast on the Internet. IBM Cryptolope is an electronic package that parcels up chunks of content, along with pricing and usage rules, in an encrypted file [21]. Users see the content excerpts—visible as plain text in the Cryptolope—and decide whether to buy the information. If they decide to buy, after filling in a billing form during an automated session, they receive an electronic key opening the Cryptolope. Compared to our system, the proposed solution does not provide great flexibility and granularity levels in access control; however, it shows, although only theoretically, an interest of a big ICT company toward cryptographic-based solutions to enforce access control in this new data distribution paradigm.

Access control, however, is not the only security service to be guaranteed when disseminating data according to the push approach. Ensuring authenticity is a key issue when digital data are stored and exchanged among different information systems, and, particularly, when pushing electronic documents to a large community of users. A large number of techniques have been developed that exploit both symmetric and asymmetric cryptography, to guarantee authenticity properties [17,36,40]. Among these proposals, digital signature technology represents the most important tool to ensure strong authenticity properties (providing also non-repudiation assurance). In this area, novel schemes have been recently proposed to provide digital signatures suitable for different contexts and usages (group and ring signatures [11,37,38] and threshold and multisignature schemes [13,30]). However, each of these approaches solves only a part of the problem related to data authenticity, disregarding the need of stating which are the authenticity requirements for the given application domain, in terms of high-level policies specifying who has to authenticate what and under which mode. This is a crucial requisite in a data dissemination system where the information to be released could be managed by different subjects who could be required to certify in different ways the information they distribute. To the best of our knowledge our system is the first addressing the problem of data authenticity in the context of push-based systems and providing a high level language to specify `signature policies`.

Finally, to ensure both access control and data authenticity requirements the attention of the research community has been mainly devoted to the development of cryptography-based techniques composing encryption and authentication operations both in symmetric and asymmetric setting: in the context of symmetric cryptography, several researchers [3,26] have investigated how to efficiently and securely combine encryption schemes with MAC techniques; in the public-key setting, instead, a new cryptographic primitive, termed as "`signencryption`"—obtained by composing a public-key encryption technique and a digital signature scheme—has been introduced [27,46,47]. Unlike these approaches, we address the problem of access control and data authenticity in a more complex way, since we provide both an XML-based language to specify access control and signature policies as well as a technique to enforce such policies. Regarding the enforcement technique adopted by our system, as in the work mentioned above it combines encryption and authentication operations in order to enforce the security policies our system supports. However, the scenario we consider requires a hybrid composition of the two cryptographic primitives. For the selective dissemination of documents under the push mode a symmetric encryption technique is necessary, instead of public-key encryption, because each decryption key must be shared among all the subjects entitled by the given policies to access the corresponding encrypted portion. For authenticity purposes, instead, we adopt digital signature techniques. Note that, in the scenario we consider, a public group signature scheme does not represent an efficient alternative with respect to a symmetric encryption scheme, given the high costs of a public-key technique and, additionally, since the set of subjects that have to share the same decryption key could frequently change with the update of the access control policies.

2.2 Secure data distribution in a third party scenario

Several solutions [5,15,16,22,28] have been proposed to ensure security properties in third party scenarios. In such a scenario three parties are involved: the information Owner, which produces the information to be disseminated or published over the web, a Subject which is interested in accessing (a portion of) such information, and the Publisher, a third party which is responsible for managing the Owner information. Most of the approaches proposed so far aim at ensuring Subject and Owner security requirements whenever the Publisher is not trusted. Although the distribution strategy adopted in these approaches is the pull one—under which the Publisher answers Subject queries—some of the ideas underlying the proposed solutions can be applied to the push dissemination mode as well.

The problem of ensuring data confidentiality against the Publisher itself has been already addressed [22,23,39], where the Owner sends to the Publisher his/her information, previously encrypted, in order to prevent the Publisher from accessing the information they manage. In [39] a technique has been proposed to allow query execution over encrypted textual data, which supports keyword-based search— single words or phrases—within an encrypted text. In [23], instead, the authors describe a solution that allows the Publisher to manage an encrypted relational database without knowing its content. In this work database tuples are encrypted using software or hardware encryption (while the database schema is not encrypted). To support query execution on the encrypted database, the database is equipped with an encrypt and a decrypt function, which the authorized Subject use with the decryption key received from the database Owner to execute a selection query over the encrypted database on the Publisher side. Such approach also allows the authorized Subjects to insert database tuples by using the encrypt function provided within the database. By contrast, according to the technique proposed in [22] the entire database is encrypted (both tuples and schema) and it is equipped with additional information, a "coarse index" for the relational attributes, to allow the query execution at the Publisher site without the need of decrypting the stored data. This technique allows the Publisher to partially execute at its site as many SQL queries as possible over encrypted data, whereas decryption and the remainder of the query processing are performed at the Subject site. In all these approaches, the problem of data confidentiality against unauthorized Subjects, which is one of the goals of our system, is marginally addressed through a simple identity based model adopted by the Information Owner for key distribution.

In case of an untrusted Publisher, the Subject also needs to be assured that the portion of information s/he receives from the Publisher is authentic, that is, it was actually originated from the Owner and that it has not been tampered with. A first attempt to address such problem has been made in the context of query authentication in outsourced database systems (ODBs) for static databases (characterized by unfrequent updates) [16]. This work provides a technique, based on Merkle hash trees [29], to ensure authenticity and non-repudiation of the query results produced by the Publisher from relational data. More specifically, it allows the Publisher to create the so-called *verification object* (VO), that is, a hard-to-forge authenticity proof consisting of a hierarchy of digests that the Subject can use to authenticate query results by exploiting the Owner's signature on the whole database. In [15] the approach was extended to provide authenticity and completeness when publishing XML documents over the web using particular authenticated data structures, called xtries. In [28], instead, the attention was focused on developing an algorithm for the construction of verification objects that are both secure and compact and, moreover, efficient to compute and verify. The general approach of using authenticated data structures to provide query authentication was also used in the context of "Edge Computing" [34], where a trusted central server outsources parts of the database to proxy servers situated at the edge of a network. By contrast, a different strategy to query authentication in ODBs appeared in [31,32]. They propose an alternative approach according to which the Owner affixes his/her digital signatures at the granularity of individual tuples, then the Publisher can securely combine the individual signatures affixed by the Owner on the tuples matching the Subject query by adopting the aggregated signature scheme proposed in [31]. The work in [32] further integrates a signature chaining mechanism to assure completeness of the query results. Both approaches to query authentication are very useful whenever a subject has to verify the authenticity of a portion of data signed by an entity (e.g., its owner) and, thus, can be applied to our system. However, both approaches assume that the verifier access only to a portion of the data to be authenticated. In the push distribution scenario we consider, instead, the whole encrypted document has to be sent to each user even if s/he can decrypt only a portion of it (the one for which s/he has the proper authorizations). We have thus decided to exploit, in our work, all the information, even if encrypted, sent to each user to correctly perform the authenticity verification of each data portion.

The goal of the work described above is to provide a mechanism that enables a `Subject` to verify the authenticity of query results without taking into account any access control rules in place for the information managed by the `Publisher`, nor authentication policies requiring to authenticate different data portions with different techniques. A comprehensive architecture for the selective and authentic third-party publication of XML documents has been proposed for the first time in [5]. This work integrates a Merkle hash tree technique for query authentication within a comprehensive framework for access control, which supports a subject subscription component, a model to specify credential-based access control policies and a mechanism to enforce the stated policies. We borrow from this work the credential-based access control model and the idea of selectively encrypting documents to enforce confidentiality. However, the work reported in [5] does not provide any type of authentication policies. As such the authentication requirements it addresses are simpler with respect to the ones addressed in this paper. What has to be assured in [5] is only that the `Publisher` does not maliciously modify the data it manages. By contrast, our system supports `signature policies`, as such we have also to ensure that the generated signatures are compliant with the requirements stated by the policies. The problem of ensuring both data authentication and access control in a third party scenario has been also addressed in [33]. The main focus, however, is the development of a technique to ensure the `Subject` about the correctness of relational query results, and its integration with an access control system applying on the given database, such that the query results do not contradict the access control policies applying to the database. Such scheme simply rewrites the queries posed by the `Subject` so as to meet the given access control policies in force for the given database. Again, this work addresses the problem of developing data authenticity techniques in the simple case where only a subject, the `Information Owner`, has to authenticate the whole information s/he generates. Our work, instead, aims at enforcing in a comprehensive way data authenticity requirements in a wider scenario where different subjects have to ensure different authenticity requirements to different portions of the same document.

3 Secure push-based distribution of XML documents

In this section we briefly describe our approach for a secure push-based dissemination system previously presented in [8]. The system has been designed to deal with XML documents because XML imposed itself as a standard format for document exchange over the web, and several XML-based technologies have been developed by the research community. However, the approach we propose can be applied to other document and data exchange formats. The system adopts an XML-based language to specify both access control and signature policies. The system is composed of two main components: the server side, to selectively encrypt and sign XML documents according to the specified security policies, and the client side, for the decryption and signature validation of the encrypted and signed document generated by the server. In the remainder of this section, we first briefly present the motivation for our work and describe our running example. Then, we describe the policy language on which our system relies. Finally, we illustrate the architecture of the system.

3.1 Motivation

Push-based dissemination systems are today widely adopted by many companies and organizations to manage and deliver their data more timely and efficiently, allowing their partners and customers to easily receive information anywhere and anytime (consider, for instance, virtual organizations, companies that have to release their industrial projects to their staff and partners or enterprises dealing with electronic commerce of information—e.g., stock prices, sport news). A push-based dissemination system has to release the information it collects in a selective manner, satisfying in addition to user preferences also security constraints. With respect to security, both access control and data authenticity requirements are crucial within push-based dissemination systems.

For instance, pay-per-view channels are well-known examples of commercial push-based dissemination systems that have to selectively broadcast multimedia data to a large set of users. Digital libraries along with systems to distribute electronic news are further examples of information-centered applications that have to release their data in a differentiated way. Research projects often involve several partners from different countries and require secure dissemination of their project deliverables and all those documents related to project management and administration tasks. E-contracting systems, additionally, provide critical virtual environments where document authenticity and confidentiality are fundamental requisites in the dissemination process.

To illustrate our system we consider as running example an organization, called *Enterprice*, that has to selectively disseminate confidential and authentic information among its employees, for management purposes. In particular, we consider the selective dissemination of

```
<Employee_dossier Emp_ID="BLMD34">
    <Profile>
        <Resume Date="15/04/99">
            <Personal_Data name="Bill" surname="Madison" >
                <Reserved>
                    <health> list of vaccines </health>
                    <criminal>fraud against. . . </criminal>
                </Reserved>
            </Personal_Data>
            <Instruction> . . . </Instruction>
        </Resume>
        <Administrative_data SSN="BLMD3456748" bank_code="US65032"/>
    </Profile>
    <Activity date="8/9/00" role="secretary">
        <pay_packet working_days="28" period="July" salary="1500"> . . . </pay_packet>
    </Activity >
    <Evaluation date="20/05/00">
        <psychological_eval Psych_eval_auth="CTRM34"> . . . </psychological_eval>
        <technical_eval>. . . <technical_eval>
        <overall_eval>. . . <overall_eval>
    </Evaluation>
    <Career date="31/12/99">
        <Position salary="300" date="5/12/99">Intern...</Position>
    </Career>
    <Benefits>
        <Meal_tickets> . . . </Meal_tickets>
        <Production_bonus> . . . <Production_bonus>
    </Benefits>
</Employee_dossier>
```

Fig. 1 An example of XML document

information about the employees of the organization. We assume that such information are contained into an XML document, called `Employee_dossier`. An example of such document is reported in Fig. 1.[1]

Example 1 The `Employee_dossier` document, uniquely identified by attribute `Emp_ID`, contains the employee's profile, the description of the activities carried out by the employee within the organization, an evaluation of the employee, information about his/her career within the organization and his/her benefits. Three elements, namely `Evaluation`, `Career`, and `Benefits`, are of particular interest for the remainder of the discussion. The `Evaluation` element contains three sub-elements, representing, respectively, three different evaluations of the employee. The first is a psychological evaluation, the second is a technical evaluation by the technical manager of the organization, and the third one an overall assessment about the employee skills and human abilities. The `Career` element provides information on the previous employee's position(s) in the organization hierarchy, whereas the `Benefits` element specifies the list of the employee benefits.

3.2 $\mathcal{X}$-Sec language

The language on which our system relies is $\mathcal{X}$-Sec [8]. It is an XML-based language that supports the specification of credentials and credential-based security policies. In what follows, we present the main features of such a language that are relevant to understand the proposed system. Formal details on $\mathcal{X}$-Sec and a full description of its features can be found in [8].

[1] Appendix A shows the formal graph-based model we adopt throughout the paper to represent XML documents.

$\mathcal{X}$-Sec allows one to specify security policies at different granularity levels, and provides selective protection both at intensional and extensional level, enabling policy specification both on a document or a DTD/XML Schema. As mentioned before, $\mathcal{X}$-Sec supports the notion of credential as a flexible way to qualify the subjects to which a policy applies. A credential allows one to identify a subject by means of a set of properties and/or through the roles a subject has within an organization. In accordance with $\mathcal{X}$-Sec, all the credentials a subject possesses are collected into an XML document called `X-profile`.

Example 2 Figure 2 shows an example of X-profile collecting all the credentials belonging to John Watson, which is both a personnel consultant and an employee supervisor within Enterprice.

3.2.1 $\mathcal{X}$-Sec access control policies

$\mathcal{X}$-Sec provides a template to specify credential-based access control policies stating which access privilege (`view`, `navigate`, `browse_all`, `write`, `append`, `author_all`) a subject can (or cannot) exercise on a given XML document, or on a portion of it. Subjects

```
<X-profile sbjID="JHNWTS" PIssuer="CA16" Pki_URI=". . . ">
  <Personnel_consultant credID="JHNWTS54" CIssuer="CA16">
    <full_name> John Watson </full_name>
    . . .
    <company> Enterprice </company>
  </Personnel_consultant >
  <Emp_supervisor credID="JHNWTS67" CIssuer = "CA08">
    . . .
    <company> Enterprice </company>
  </Emp_supervisor>
  . . .
</X-profile>
```

Fig. 2 An example of $\mathcal{X}$-profile

are qualified by specifying conditions on their profiles, expressed as $\mathcal{X}$Path [42] compliant expressions, referred to as `credential expressions`. An $\mathcal{A}$ccess $\mathcal{C}$ontrol $\mathcal{P}$olicy $\mathcal{B}$ase, briefly denoted with $\mathcal{ACPB}$, is an XML document instance of the $\mathcal{X}$-Sec access control policy base template. Figure 3 illustrates the graph representation of the template, whereas Table 1 presents a brief description of its attributes. We refer the interested reader to [8] for a detailed description of such template and its semantics. As usually, the semantics of an access control policy acp*i*, briefly denoted with ϵ(acp*i*), is the set of authorizations enacted by the policy. Each authorization can be modeled as a tuple (*s*,*o*,*p*,*t*), where *s* is a subject, *o* is a protection object, *p* denotes the access privilege on *o*, and, finally, *t* states whether granting or denying *s* the access authorization on the object *o* according to the privilege *p*.

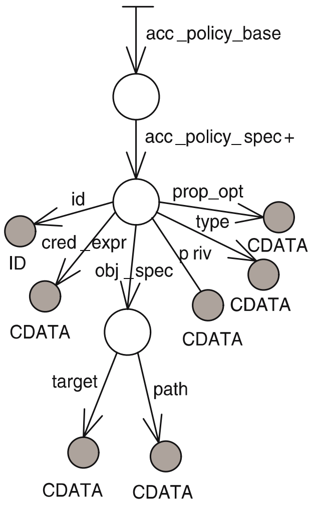


Fig. 3 Graph representation of the $\mathcal{X}$-Sec $\mathcal{A}$ccess $\mathcal{C}$ontrol $\mathcal{P}$olicy $\mathcal{B}$ase template

Table 1 Attribute description of the $\mathcal{X}$-Sec $\mathcal{A}$ccess $\mathcal{C}$ontrol $\mathcal{P}$olicy $\mathcal{B}$ase template

Attr_name	*Attr_value*
id	character string (acp*i*) identifying an access control policy
cred_expr	Xpath compliant expression on $\mathcal{X}$-profiles
priv	the access mode enabled by the policy (`view`, `navigate`, `browse_all`, `write`, `append`, `author_all`)
type	policy type: positive (`grant`) or negative (`deny`)
prop_opt	the depth level $n \in \mathbb{N} \cup \{*\}$ to which the policy propagates
target	a DTD/XML-Schema/document name to which the policy applies
path	Xpath compliant expression denoting selected portions within the `target` document(s)

Example 3 Figure 4 presents an example of an $\mathcal{ACPB}$, specifying a set of access control policies, defined for a document source $\mathcal{S}$ including the `Employee_dossier` document previously illustrated. According to the first two policies, managers of Enterprice are authorized to view all the documents conforming to the Employee_dossier.dtd, except for those portions containing employee administrative data and information about his/her career, as expressed by their semantics: ϵ(acp1) = {(*s*, *o*, `browse_all` , `grant`)|*s is a manager, o is any node in the documents conforming to the specified DTD*}, and ϵ(acp2) = {(*s*, *o*, `browse_all`, `deny`)|*s is a manager, o is the* `Administrative_data` *element or a node within the subtree rooted at the* `Career` *element* }. The semantics of the third policy, instead, is: ϵ(acp3) = {(*s*, *o*, `browse_all`, `grant`)|*s is a secretary, o is a node within the subtrees rooted at the* `Administrative_data`, `Activity`, `Career` *and* `Benefits` *elements*}. Therefore, acp3 authorizes a secretary to view all the information about employee activity within the organization, his/her career, his/her administrative data together with the benefits the employee has. Finally according to the fourth and fifth policies, each member of the board of directors is able to view the overall evaluation and the employee's information contained in the document (ϵ(acp4) = {(*s*, *o*, `view`, `grant`)|*s is a member of the board of directors, o is a node within the subtrees rooted at the* `overall_eval`, `Resume` *and* `Career` *elements*}), with exception of the employee reserved data along with his/her salary. (ϵ(acp5) = {(*s*, *o*, `view`, `deny`) |*s is a member of the board of directors, o is a node within the subtree rooted at the* `Reserved_data` *element* or it is the `salary` attribute}).

3.2.2 $\mathcal{X}$-Sec signature policies

$\mathcal{X}$-Sec also supports the specification of credential-based signature policies, which express the duty of one or more subjects, whose $\mathcal{X}$-profiles satisfy the specified credential expressions, of signing the document(s) or document portions referred in the policy. The characteristics of the template according to which a signature policy can be specified are summarized in Fig. 5 and Table 2.

Figure 5 illustrates the graph representation of the template, whereas Table 2 provides its attribute description. A $\mathcal{S}$ignature $\mathcal{P}$olicy $\mathcal{B}$ase ($\mathcal{SPB}$ for short), is therefore an XML document instance of the $\mathcal{X}$-Sec signature policy base template. A remark is needed for the *duty* attribute, which provides the signature type (`single` or `joint`) required by the policy. If *duty* is set to `sign`, a single signature policy is specified requiring that only one subject (identified by only one `cred_expr` element)

```
<acc_policy_base>
    <acc_policy_spec id="acp1" cred_expr="//Manager" priv=" browse_all" type="grant" propt_opt="*">
        <obj_spec target="Employee_dossier.dtd" path="/Employee_dossier"/>
    </acc_policy_spec>
    <acc_policy_spec id="acp2" cred_expr="//Manager" priv=" browse_all" type="deny" propt_opt="*">
        <obj_spec target="Employee_dossier.dtd" path="//Administrative_data| //Career"/>
    </acc_policy_spec>
    <acc_policy_spec id="acp3" cred_expr="//Secretary" priv="browse_all" type="grant" propt_opt="*">
        <obj_spec target="Employee_dossier.dtd" path="//Administrative_data| //Activity| //Benefits| //Career"/>
    </acc_policy_spec>
    <acc_policy_spec id="acp4" cred_expr="//Board_dir_member" priv="view" type="grant" propt_opt="*"/>
        <obj_spec target="Employee_dossier.dtd" path="//Evaluation|//Resume| //Career"/>
    </acc_policy_spec>
    <acc_policy_spec id="acp5" cred_expr="//Board_dir_member" priv="view" type="deny" propt_opt="*"/>
        <obj_spec target="Employee_dossier.dtd" path="//Reserved|@salary"/>
    </acc_policy_spec>
</acc_policy_base>
```

Fig. 4 An example of $\mathcal{A}$ccess $\mathcal{C}$ontrol $\mathcal{P}$olicy $\mathcal{B}$ase

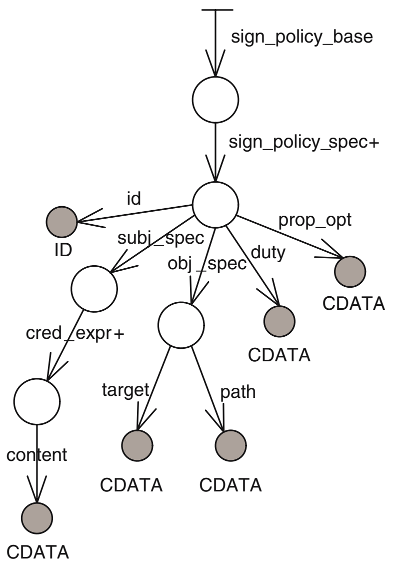


Fig. 5 Graph representation of the $\mathcal{X}$-Sec $\mathcal{S}$ignature $\mathcal{P}$olicy $\mathcal{B}$ase template

Table 2 Attribute description of the $\mathcal{X}$-Sec $\mathcal{S}$ignature $\mathcal{P}$olicy $\mathcal{B}$ase template

Attr_name	*Attr_value*
id	character string (sp*i*) identifying a signature policy
duty	the signature type required by the policy (`sign`/ `joint_sign`)
prop_opt	the depth level $n \in \mathbb{N} \cup \{*\}$ to which the policy propagates
target	a DTD/XML-Schema/doc name to which the policy applies
path	Xpath compliant expression denoting selected portions in the `target` document(s)

signs the authentication object[2] specified by the policy. If *duty* is set to `joint_sign`, instead, a joint signature policy is required, stating that more than one subject (each of which is identified by a `cred_expr` element) must sign the same authentication objects. Formal definitions of such template and its semantics are reported in [8]. The semantics of a signature policy, sp*i*, (denoted as ϵ(sp*i*)) is defined as a set of triples ($\bar{s}$,o,d), the first component of which is a subject or a set of subjects $\bar{s}$ that potentially must sign, the second component specifies an authentication object o that must be signed, and, finally, the last component is the signature duty d expressing the need of a single signature by only one subject or a joint signature by more than one subject.

Example 4 Figure 6 presents an example of a $\mathcal{SPB}$ defined for the document in Fig. 1. The first policy is an example of joint signature policy. It specifies that each employee must jointly sign his/her profile together with the employee supervisor, as expressed by its semantics: ϵ(sp1) $=$ $\{((s_1, s_2), o,$ `joint _sign`$)|$ s_1 *is the employee that owns the considered* `Employee_dossier` *document,* s_2 *is one of the employee supervisor, o is a node within the subtree rooted at the* `Profile` *element*$\}$. The second policy is a single signature policy requiring that a personnel consultant signs the evaluation of the employee (ϵ(sp2) $=$ $\{(s, o,$ `sign`$)|$ *s is a personnel consultant, o is a node within the subtree rooted at the* `Evaluation` *element*$\}$). The third policy is a single signature policy that imposes that the psychologist who has written the psychological evaluation of employee Madison affixes his/her signature on his/her psychological evaluation (ϵ(sp3) $=$ $\{(s, o,$ `sign`$)|$ *s is the psychologist who has written the psychological evaluation of employee Madison, o is the* `psychological_eval` *element*$\}$). Finally, the fourth signature policy is a joint policy stating that each pay statement of the employee must be jointly signed by the employee, by a secretary, and by the head of the payroll department, as expressed by its semantics ϵ(sp4) $=$ $\{((s_1, s_2, s_3), o,$ `joint_sign`$)|$ s_1 *is the employee that owns the considered* `Employee_dossier` *document,* s_2 *and* s_3 *are, respectively, a secretary*

[2] The term *authentication object* denotes the nodeset that must be signed according to a signature policy.

Fig. 6 An example of $\mathcal{S}$ignature $\mathcal{P}$olicy $\mathcal{B}$ase

```
<sign_policy_base>
  <sign_policy_spec id="sp1" duty="joint_sign" prop_opt="*">
    <subj_spec>
      <cred_expr id="c1"> //Employee[@credID=Emp_ID] </cred_expr>
      <cred_expr id="c2"> //Emp_supervisor </cred_expr>
    </subj_spec>
    <obj_spec target="Employee_dossier.xml" path="//Profile"/>
  </sign_policy_spec>
  <sign_policy_spec id="sp2" duty="sign" prop_opt="*">
    <subj_spec>
      <cred_expr id="c5"> //Personnel_consultant </cred_expr>
    </subj_spec>
    <obj_spec target="Employee_dossier.dtd" path="/Employee_dossier/Evaluation"/>
  </sign_policy_spec>
  <sign_policy_spec id="sp3" duty="sign" prop_opt="0">
    <subj_spec>
      <cred_expr id="c6"> //Psychologist[@credID=Psych_eval_auth] </cred_expr>
    </subj_spec>
    <obj_spec target="Employee_dossier.dtd" path="//psychological_eval"/>
  </sign_policy_spec>
  <sign_policy_spec id="sp4" duty="joint_sign" prop_opt="*">
    <subj_spec>
      <cred_expr id="c7"> //Employee[@credID=Emp_ID] </cred_expr>
      <cred_expr id="c8"> //Payroll_dept_secret </cred_expr>
      <cred_expr id="c9"> //Payroll_dept_head </cred_expr>
    </subj_spec>
    <obj_spec target="Employee_dossier.dtd" path="//pay_packet"/>
  </sign_policy_spec>
</sign_policy_base>
```

and the head of the payroll department, o is the `pay_packet` *element*}.

Note that policy `sp2` in Fig. 6 requires that `psychological_eval` element be signed by a personnel consultant, whereas according to the signature policy `sp3`, the same element must be signed by one among the psychologists who made the employee psychological evaluation. In such a case, the question arises of which signature duty must be enforced, that is, who must sign the psychological evaluation. A possibility is to combine these signature duties in such a way to require a joint signature by both a personnel consultant and a psychologist. Alternatively, it can be a single signature of a subject which has both the two roles within the organization (if such a subject exists). In any case, there is the need to state how to combine different signature duties expressed on the same authentication object. We will elaborate on that in the next section.

3.3 System architecture

The system we propose (see Fig. 7) includes a server and a client side. It exploits cryptographic techniques to enforce access control and digital signature mechanisms to ensure authenticity requirements. The server side selectively signs and encrypts the given XML document according to the access control and signature policies the Security Administrator (SA) specifies for it. The result is a ciphered document along with the symmetric keys (denoted as k_i in Fig. 7) and the generated signature(s) (denoted with S_i in Fig. 7), which are recorded in a database. For security reasons, each symmetric key in the database is encrypted with the public key of the subject to which it refers to. Then, a distribution module is responsible for broadcasting the encrypted and signed document to all the subjects subscribed to the system (each subject s_i is identified by its $\mathcal{X}$-profile, $\mathcal{X}$-p_{si} in

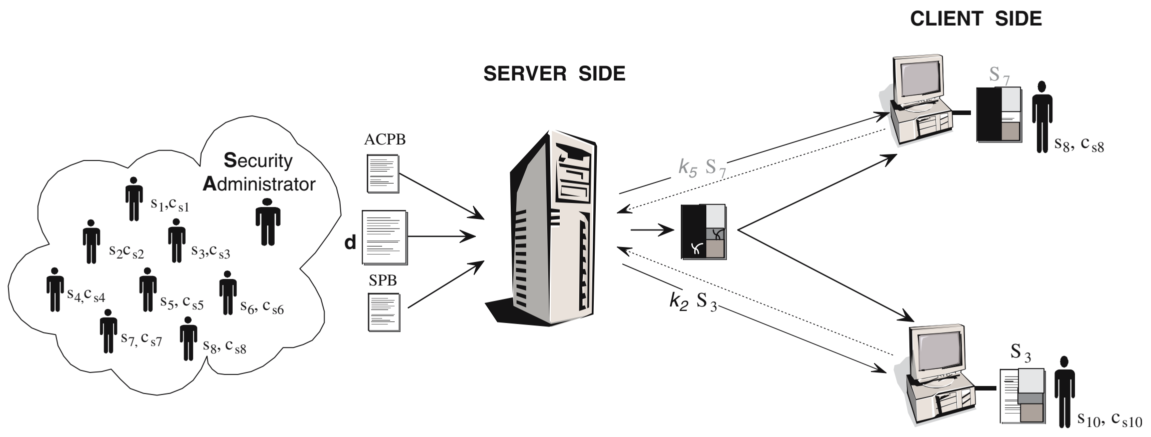


Fig. 7 Overall system architecture

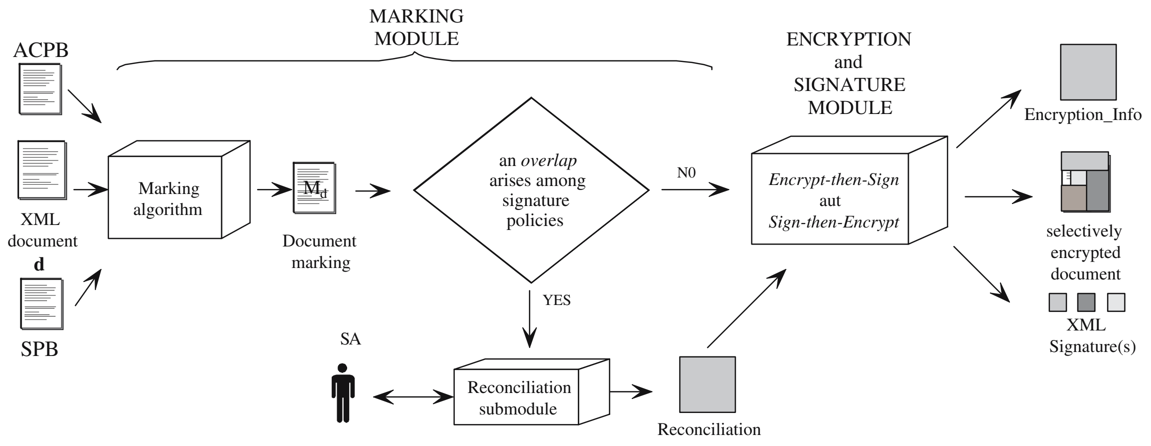


Fig. 8 Server components

Fig. 7). Furthermore, such module provides each receiving subject with the set of keys and the signatures the subject needs to decrypt and validate the authenticity of the document portions s/he can access, on the basis of the specified policies. Two different approaches for key and signature distribution are supported: the *online mode*, according to which the signature(s) and keys are directly sent to each subject by the distribution module; and the *offline mode*, according to which the decryption keys and the signatures are stored in an LDAP directory at the server side and accessed by subjects when needed. Finally, the client component, which can be downloaded from the server during the subscription phase to the distribution service, decrypts the document portions by means of the received keys, and verifies the authenticity of the authorized portions, by using the XML Signature(s) s/he obtains from the distribution module. In the following, we describe the server and client components we have implemented, with the only exception of the distribution module the implementation of which is under way.

3.3.1 Server side

The server side (see Fig. 8) consists of the following modules: the *Marking* module, which contains an optional submodule, called *Reconciliation*, and the *Encryption* and *Signature* module. The *Marking* module receives as input an XML document `d`, the access control and signature policy bases defined for the source $\mathcal{S}$ containing `d`, and it associates each portion of the document with the identifiers of the access control and signature policies applied to that portion. The result is the *document marking*, which is a set of triples (el, ID_{acp}, ID_{sp}), where *el* is an expression identifying a node in `d`, ID_{acp} is the identifier of the `access control policy configuration`, PC_{acp}, that contains the identifiers of positive access control policies applying on *el* that are not overwritten by a negative one, and ID_{sp} identifies the `signature policy configuration`, PC_{sp}, that is, the set of signature policy identifiers applying on *el*.[3]

Example 5 Consider the `Employee_dossier` document presented in Fig. 1, the $\mathcal{ACPB}$, and $\mathcal{SPB}$ illustrated in Figs. 4, and 6, respectively. The resulting marking is shown in Fig. 9. Note that keyword TAG is used to denote that the policy marking applies only to the start and end tags of the corresponding element. Additionally, each color corresponds to a different access control policy configuration.

If more than one signature policy applies to the same document portion, the *Marking* module activates the *Reconciliation* submodule. The reconciliation of signature policies is needed to make sure that each document node is marked with at most one signature policy. Its aim is to substitute with the overlapping policies only one signature policy that must be used in the signature process. If the overlapping signature policies require the signature of the same subject categories, the system automatically solves the overlap by replacing such policies with only one of them. Otherwise, the SA is required to choose how to solve the overlap, by selecting one among the provided reconciliating operators (*And*, *Choice* and *Concatenation*). The *And* operator applies only to single overlapping signature policies: the resulting signature

[3] PC_{acp} can contain the keyword TAG, if at least two attributes of element *el* are marked with two different access control policy configurations, with the aim to denote that the start and end tag of the given element must be encrypted with a different key with respect to its content.

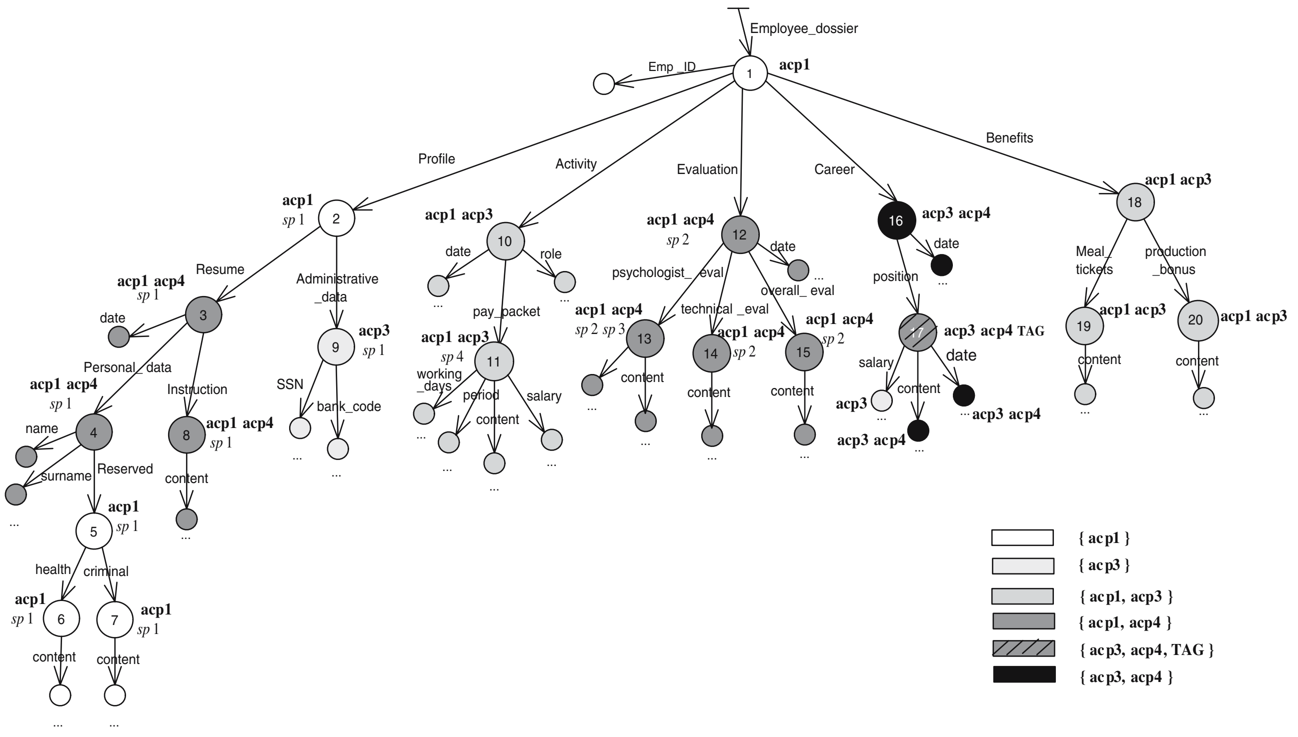


Fig. 9 Graph representation of the marking of the `Employee_dossier` document in Fig. 1 according to the security policies shown in Figs. 4 and 6

policy is a single policy too, whose `cred_expr` element is the composition through the XPath symbol '|' of each credential expression in the overlapping policies. The *Choice*$_k$ operator, on the contrary, allows the SA to select the identifier of one among the overlapping policies: the index k represents the identifier of the selected signature policy. Finally, the *Concatenation* operator generates a joint signature policy, whose subject specification contains the credential expressions of each overlapping policy. The *Reconciliation* module creates a table, called `Reconciliation`, that stores information about the operators chosen to reconcile the overlapping policies.

Example 6 With reference to Fig. 6, signature policies `sp2` and `sp3` overlap on the `psychological_eval` element: `sp2` requires a single signature by a personnel consultant, whereas `sp3` states that one among the psychologists who made the psychological evaluation of the employee must sign the `psychological_eval` element. Therefore, the *Reconciliation* module is invoked in order to allow the SA to reconcile the overlapping policies by selecting one among the reconciliation operators. In this case, if the SA selects the *Concatenation* operator, then the `psychological_eval` element must be jointly signed by a personnel consultant and by a psychologist. Alternatively, the SA can select only one of the overlapping policy, e.g. `sp2`, by specifying the *Choice* operator of index 2. Finally, the SA can select the *And* operator in such a way to require a single signature by a subject that is both a personnel consultant and the psychologist who wrote the psychological evaluation of the considered employee (if such a subject exists).

Afterwards, the *Encryption and Signature* module is activated. This module receives as input the document `d`, the *marking* of `d` generated by the *Marking* module and the `Reconciliation` table, if the *Reconciliation* module has been activated; then, it executes both the encryption and signature operations. Different portions of the input document are encrypted by using different symmetric keys and are authenticated by means of different signatures according to the specified policies. More specifically, each document portion that is labeled with the same access control policy configuration PC_{acp}, hereafter called *access region* and denoted with AR_{acp}, is encrypted with a symmetric key k. Similarly, each document portion marked with the same signature policy configuration PC_{sp}, which we call in what follows *signature region* and denoted with SR_{sp}, is authenticated by means of the same XML Signature generated according to the policy in the configuration. The result is an encrypted document along with the generated XML Signature(s), and one table, called `Encryption_Info`, storing all information which allow the receiver subjects to correctly decrypt the received document. Encryption and signature operations can be performed according to the following strategies: *encrypt-then-sign* and *sign-then-encrypt*. According to the first strategy, the given document is first encrypted and then signed. By contrast, the second strategy affixes the signatures required by the specified signature policies on the clear document and then generates the document encryption. Such a strategy, additionally, requires the encryption of the generated signature(s), in order to avoid guessing attacks. Both strategies have their advantages and drawbacks. The *encrypt-then-sign* strategy has two main advantages: the first one is that if a subject has to sign different document portions each of one cyphered with different keys (i.e., a signature region covers several access regions), the system that supports this strategy generates only one signature for all the portions. A receiving subject can, indeed, validate the given signature on the encrypted document even if he/she can access only a subset of the signed portion given that he/she receives the whole encrypted document. The second one is that is not necessary to encrypt signatures before sending them to the clients, because signatures are applied on the cyphered document. Another advantage of this strategy is the limitation of the number of unnecessary decryption operations. The receiver subject, in fact, can verify the authenticity of the document portions s/he can access without decrypting them. This strategy has, however, the drawback that the signer authenticates encrypted content. The *sign-then-encrypt* strategy, instead, has the main advantage that the signer sees what s/he authenticates; the drawbacks are first that, when a subject has to sign different portions cyphered with different keys—i.e., access regions—, the system has to create a signature for each access region. Moreover, the documents containing the signatures should be encrypted in order to avoid sniffing of information about the clear content. The SA can select the strategy that better fits the characteristics of the considered application domain, based on the security requirements, efficiency constraints, and so on.

Finally, note that according to the approach for key management adopted by the system presented in this section, if $\mathcal{ACPB}$ specifies N_{acp} access control policies, the system should generate and manage in the worst case $2^{N_{acp}}$ different symmetric keys, one for each different policy configuration that could be generated from N_{acp} policies, and distribute to each subject the keys (encrypted with the subject's public key) to decrypt the access regions s/he is entailed to view. In the next section, we will present the new key management scheme we have decided to adopt, which requires to permanently store a number of keys linear in the number of (positive) access control policies.

3.3.2 Client side

The client side consists of only one module, called *Verification and Decryption* (see Fig. 10). The client module receives as input the encrypted document and the keys and signatures necessary to decrypt and verify the authenticity of the document portions the receiver subject can access. Additionally, for each key also the references (expressed as XPath compliant expressions) to the document portions encrypted with that key are received. The *Verification and Decryption* module can operate according to two different strategies: *verify-then-decrypt* and *decrypt-then-verify* corresponding to the two different encryption and signature strategies that the server can use. The *verify-then-decrypt* strategy is executed if the server side has applied the *encrypt-then-sign* strategy. It performs the validation of the signatures applied to the encrypted document, and, then, if this operation successfully executes, the client system notifies which document nodes have proved to be authentic and then decrypts the document portions the subject can access. Otherwise, the system notifies the subject which document portions are not authentic and does not execute the decryption of these portions, thus limiting the number of unnecessary decryption operations. By contrast, the *decrypt-then-verify* strategy is performed if the server has operated according to the *sign-then-encrypt* approach. According to the *decrypt-then-verify* strategy, the client first decrypts the document portions the receiving subject can access along with the signatures related to such portions, and then it executes the signature validation process.

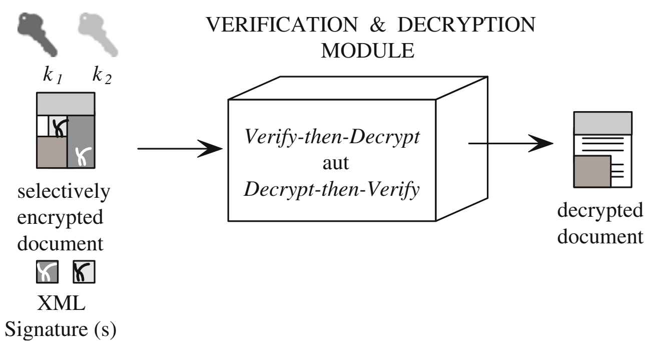


Fig. 10 Client architecture

4 Key management

The problem of key management is crucial in our system that exploits a symmetric encryption technique to selectively protect the access to sensitive information. According to the key management approach adopted in [8], for each subject to which policies acp1,..., acp*m* apply the system has to sent to the subject the keys—cyphered with his/her public key—to decrypt each access region marked with a policy configuration that contains at least one of the policies acp1,..., acp*m*. Therefore, given the high number of subjects to which the document has to be frequently released, the management and distribution of secret keys represent a crucial task both for security and efficiency reasons. Hence, to address such issue we have extended the system proposed in [8] by applying the more efficient key management strategy proposed by Atallah et al. in [2] and by customizing it to our context. In this work, the authors address the key management problem in an access hierarchy. In general, an access hierarchy is modeled as a partially ordered set of classes where a subject, who is entitled to have access to a certain class C, can also access its descendant classes in the hierarchy. An access hierarchy is represented as a directed graph where the vertexes represent the access classes and the edges the access relations between them. A key management scheme based on an access hierarchy assigns a key to each access class and distributes to each subject only a subset of the keys assigned to the class(es) the subject is authorized to access, that is, only those one that permit the subject to obtain access to his/her class(es) and all descendants in the hierarchy. The scheme proposed by Atallah et al. in [2] associates with each vertex of the graph a private key k_i in $\{0,1\}^{\rho}$ and a label l_i in $\{0,1\}^{\rho}$ (with ρ an integer), which represent, respectively, the private and public information associated with the class represented by that vertex. Then, it labels each edge between two vertexes v_i and v_j in the graph with the public value $y_{i,j} = k_j \; XOR \; H(k_i, l_j)$, where H: $\{0,1\}^* \rightarrow \{0,1\}^{\rho}$ is an hash function which returns a string of length ρ, for any input binary string.

A subject, who knows the private key k_i assigned to vertex v_i, can easily derive the secret key k_j associated with a child vertex v_j by executing the XOR operation between $y_{i,j}$ and the hash value of k_i and the public label l_j. Note that this scheme uses only hash functions for a vertex to derive each descendant key from its own key and requires only $O(l)$ bit operations, where l is the length of the path between the vertexes. Furthermore, since any hash function is not computationally invertible only the subject which knows the private information (secret key) associated with a vertex can deduce the secret key associated with each descendant vertex in the hierarchy.

This scheme for key management naturally applies to our system, where each access region AR_{acp} marked with an access control policy configuration PC_{acp} (policy configuration for short) constitutes a (non-trivial) access class, as formally stated by the following definition.

Definition 1 (Access class) Let d be an XML document to be released according to the policies specified in $\mathcal{ACPB}$. Let us denote with $PC^{\mathtt{d}}_{\mathtt{acp}}$ the set of the policy configurations $\mathrm{PC}_{\mathtt{acp}}$ identified in the marking M_d of d and with $PC^{1}_{\mathtt{acp}}$ the set of all the possible policy configurations consisting of a single positive access control policy, that can be generated starting from the policies in $\mathcal{ACPB}$. The set of nodes in d to which a policy configuration $PC_{\mathtt{acp}} \in PC^{\mathtt{d}}_{\mathtt{acp}} \cup PC^{1}_{\mathtt{acp}}$ applies is defined as the *access class* C associated with PC_{acp}. If $PC_{acp} \in PC^{\mathtt{d}}_{\mathtt{acp}}$, the access class associated with PC_{acp} is the access region AR_{acp} marked with PC_{acp}, which is said to be a *non-trivial* access class. Otherwise it is an empty node-set, which is said to be a *trivial* access class.

Hereafter we simply denote with $PC_{\mathtt{acp}i,j,k}$ the policy configuration consisting of policies acp*i*, acp*j* and acp*k*, and with $C_{i,j,k}$ the corresponding access class.

Note that a subject which is entailed to access a class C_1 marked with $PC_{\mathtt{acp1}} = \{\mathtt{acp1}\}$ is obviously authorized to access class $C_{1,3}$ marked with $PC_{\mathtt{acp1,3}} = \{\mathtt{acp1}, \mathtt{acp3}\}$ and, in general, all the node-sets marked with at least acp1. This because the policy configuration associated with a class states which subjects can access the nodes belonging to that class. These are all the subjects to which one of the policies in the corresponding configuration applies. Therefore, given two access classes C and C' marked with $PC_{\mathtt{acp}}$ and $PC'_{\mathtt{acp}}$, respectively, if $PC_{\mathtt{acp}} \subset PC'_{\mathtt{acp}}$, then we can assert that all the subjects which can access C can also access C'. We can thus define a hierarchy among our access classes from the inclusion relation between policy configurations in $PC^{\mathtt{d}}_{\mathtt{acp}} \cup PC^{1}_{\mathtt{acp}}$.

Definition 2 (Dominance relation) Let $PC_{\mathtt{acp}}$ and $PC'_{\mathtt{acp}}$ be two policy configurations in $PC^{\mathtt{d}}_{\mathtt{acp}} \cup PC^{1}_{\mathtt{acp}}$. Let C and C' be the access classes corresponding to $PC_{\mathtt{acp}}$ and $PC'_{\mathtt{acp}}$, respectively. We say that C *dominates* C' (or equivalently that C' is *dominated* by C) — denoted with $C \geq C'$ – if and only if $PC_{\mathtt{acp}} \subseteq PC'_{\mathtt{acp}}$.

The dominance relation between our access classes, defined above, is a partial order, since it is defined starting from the inclusion relation among policy configurations in $PC^{\mathtt{d}}_{\mathtt{acp}} \cup PC^{1}_{\mathtt{acp}}$. The dominance relation thus identifies an access hierarchy among the access classes defined by Definition 1. In the graph representation of this access hierarchy a source vertex represents a dominant class, that is, an access class that is not dominated by any other class in the hierarchy. Such classes are all and only those classes associated with a policy configuration in $PC^1_{\texttt{acp}}$.

We apply the scheme proposed in [2] to the subgraph obtained from the graph representing the dominance relation by considering all the vertexes in the graph but only the edges originating from the source vertexes. Once each access class has been encrypted with a different secret key, the scheme in [2] applied to this subgraph allows to distribute to each subject only those keys to decrypt the dominant classes the subject is authorized to access. A subject, indeed, can easily derive from such keys (and other public information) all and only the keys needed to decipher their descendants in the hierarchy. More precisely, the scheme in [2] applied to our subgraph assigns to each vertex v of an access hierarchy (representing an access class C) a symmetric encryption key k and a public label l. As label l we use the access control policy configuration $PC_{\texttt{acp}}$ corresponding to the access class. As a consequence, each edge between vertexes v and v' (to which k' and $PC'_{\texttt{acp}}$ have been assigned) is labeled with the XOR of k' and $H(k, PC'_{\texttt{acp}})$, which allows only who knows k the derivation of the symmetric key k' associated with v'.

Example 7 Figure 11 illustrates the labeled class hierarchy associated with the `Employee_dossier` document in Fig. 1 to which the access control policies in Fig. 4 apply. The graph contains six different non-trivial access classes C_1, C_3, $C_{1,3}$, $C_{1,4}$, $C_{3,4}$, and $C_{3,4,\text{TAG}}$ corresponding, respectively, to the policy configurations {`acp1`}, {`acp3`}, {`acp1,acp3`}, {`acp1, acp4`}, {`acp3, acp4`}, and {`acp3,acp4,TAG`} applying to the `Employee_dossier` document, as shown in Fig. 9, plus the trivial class C_4 associated with the policy configuration {`acp4`}. The graph in Fig. 11 will be published on the web by the server to allow each subject to correctly derive the keys the subject needs from the keys s/he received related to the dominant classes s/he is authorized to access. For instance, a subject, who is a secretary, is authorized to access (non-trivial) classes C_3, $C_{1,3}$, $C_{3,4}$ and $C_{3,4,\text{TAG}}$, that is, all the access regions in the `Employee_dossier` document to which at least policy `acp3` applies. Unlike the standard approach for key management, s/he will receive along with the encrypted XML document, and the XML signatures, only key k_3 ciphered with his/her public key. Then his/her client module will use k_3 to decrypt the access class C_3 and to derive keys k_{13}, k_{34} and $k_{34\text{TAG}}$ (for decrypting $C_{1,3}$, $C_{3,4}$ and $C_{3,4,\text{TAG}}$) from k_3 (received by the server) and the edge labels $y_{3,13}$, $y_{3,34}$ and $y_{3,34\text{TAG}}$. By contrast, a member of the Board of Directors, to which policy `acp4` applies, will receive the key k_4 associated with the trivial class C_4, which s/he will use only to derive k_{14}, k_{34} and $k_{34\text{TAG}}$ for decrypting the (non-trivial) access classes $C_{1,4}$, $C_{3,4}$ and $C_{3,4,\text{TAG}}$ s/he is authorized to access.

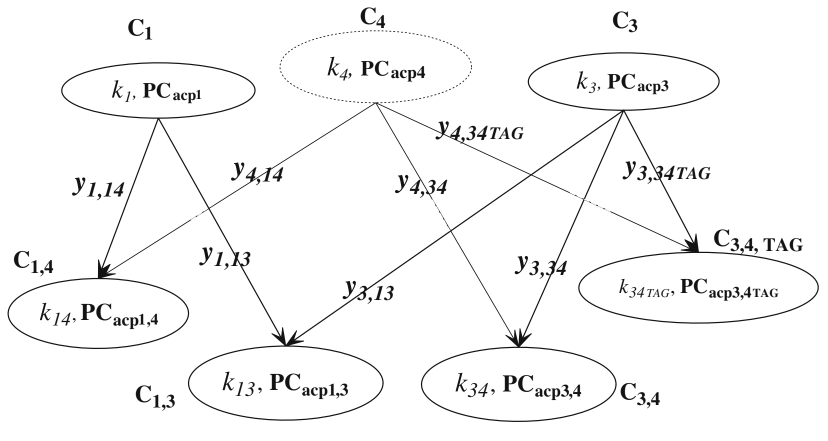


Fig. 11 The access hierarchy for our running example

$y_{1,14} = k_{14}$ XOR $\mathrm{H}(k_1, \mathbf{PC}_{\mathbf{acp1,4}})$ $\quad y_{3,13} = k_{13}$ XOR $\mathrm{H}(k_3, \mathbf{PC}_{\mathbf{acp1,3}})$

$y_{1,13} = k_{13}$XOR $\mathrm{H}(k_1, \mathbf{PC}_{\mathbf{acp1,3}})$ $\quad y_{3,34} = k_{34}$ XOR $\mathrm{H}(k_3, \mathbf{PC}_{\mathbf{acp3,4}})$

$y_{4,14} = k_{14}$ XOR $\mathrm{H}(k_4, \mathbf{PC}_{\mathbf{acp1,4}})$ $\quad y_{4,34} = k_{34}$ XOR $\mathrm{H}(k_4, \mathbf{PC}_{\mathbf{acp3,4}})$

$y_{4,34TAG} = k_{34TAG}$ XOR $\mathrm{H}(k_4, \mathbf{PC}_{\mathbf{acp1,4,TAG}})$ $\quad y_{3,34TAG} = k_{34TAG}$XOR $\mathrm{H}(k_3, \mathbf{PC}_{\mathbf{acp3,4,TAG}})$

5 System implementation

In this section, we describe the implementation of server and client sides. The system has been developed in Java and uses the eXcelon XML Platform 3.5 [18] as XML document storage technology. XML data are recorded in a storage unit, called `XMLstore`, where data can be manipulated using the Document Object Model (DOM) [41]. All the necessary information for the correct functioning of the system is encoded in XML.

5.1 Server implementation

The implementation of the server module consists of five Java components, called *Server*, *Marking*, *Reconciliation*, *Sign_then_Encrypt* and *Encrypt_then_Sign*, which are composed of several methods. The first component states the input parameters of the system and activates the other four Java components, which correspond to the system modules described in the previous section. The *Server* module makes use of a repository, called `XMLstore`, which stores the XML document `d` to be encrypted and signed along with its DTD/XML Schema, the $\mathcal{ACPB}$ and $\mathcal{SPB}$ containing, respectively, the access control and signature policies defined for `d`, and, finally, an XML document providing information about the subjects involved in the dissemination process, such as personal data, subject X-profiles and other information useful during the signature operation (e.g., the subject's public key). The *Server* component receives as input parameters the name of the `XMLStore` along with the names of the input documents it contains. Its task is to create a session with the eXcelon `XMLStore` and then to activate the *Marking* component, to generate the document marking. If there are overlaps among the signature policies applying to `d`, the *Reconciliation* component is invoked. After that, the SA chooses which component to execute for the encryption and signature operations: the *Encrypt_then_Sign* component is activated whenever the *encrypt-then-sign* strategy is selected, otherwise the *Sign_then_Encrypt* component is activated. In the following sections we illustrate the implementation of the above mentioned components.

5.1.1 *Marking component*

The *Marking* component performs the marking operation, using the strategy illustrated before. The first step it performs is the SAX parsing [25] of the input XML document `d`, in order to state the type (IDREF, URI or others) of the attributes in the document. Then, the DOM parsing of `d` is performed. The next step is the insertion of an ID attribute into all the elements of `d`, performed by the `Add_Id` method. This is needed to easily refer each element in the input document (as well as in the encrypted document, as we will explain in the following) by means of the inserted (not sensitive) ID attributes. Then, the methods `Create_acc_control_pol_list` and `Create_sign_pol_list`, generate the lists containing, respectively, all the access control and signature policies defined for `d` or for its DTD/XML Schema from the input $\mathcal{ACPB}$ and $\mathcal{SPB}$, by using XPath. Policy lists are implemented by the Vector Java class. Each element of the list representing a policy is an object, that is, an instance of the *Policy* Java class. Such object includes a class variable for each policy specification element and one class variable for the XPath expressions specifying the nodes the policy is applied to. Then, the method `Compare_lists` verifies whatever a positive policy is overwritten by a negative one, and, in this case, it updates the class variable related to the positive policy, excluding the nodes denied by the negative one. After this, access control and signature policies lists are merged. Then, the method `Label` traverses, for each node n belonging to `d`, the list to determine the identifiers of the policies applied to n. The resulting marking is modeled through an XML document that has the same elements of the input document `d`: to each element three child nodes are appended containing, respectively, the marking of the element, of its attributes and data content. Representing the document marking in an XML format simplifies the access to the information it contains because it can be manipulated using DOM and XPath. An example of such document is given in Fig. 12. After the creation of the document marking, the *Marking* component verifies for each element of the marking if the start and end tag of the element have to be encrypted with a different key with respect to its attributes and content. If this is the case, the component marks the considered element with the "TAG" label. At the end, the document marking is stored in the `XMLStore`.

5.1.2 *Reconciliation component*

The *Reconciliation* component is invoked by the *Marking* component whenever a set of signature policies overlap on the same document portion. The component first verifies the type of the overlap (trivial or not). If the overlap is trivial, that is, the overlapping policies on a document region require the signature(s) by the same subject categories, the overlap is automatically solved by selecting anyone of the overlapping policies. Otherwise, an interactive phase is activated in which the SA is required to choose how to reconcile the overlapping policies. For each overlapping signature policy

```
<Employee_dossier Id="1">
  <TAG>acp1</TAG>
  <ATTR id_attr="1.a1" name="Emp_ID">acp1</ATTR>
  ...
  <Evaluation Id="12">
    <TAG>acp1 acp4 sp2</TAG>
    <ATTR id_attr="12.a1" name="date">
    acp1 acp4 sp2
    </ATTR>
    <psychological_eval Id="13">
      <TAG>acp1 acp4 sp2 sp3</TAG>
      <ATTR id_attr="13.a1" name="psych_eval_auth">
        acp1 acp4 sp2 sp3
      </ATTR>
      <TEXT>acp1 acp4 sp2 sp3</TEXT>
    </psychological_eval>
    ...
  </Evaluation>
  <Career Id="16">
    <TAG>acp3 acp4</TAG>
    <ATTR id_attr="16.a1" name="date">
    acp3 acp4
    </ATTR>
    <Position Id="17">
      <TAG>acp3 acp4 TAG</TAG>
      <ATTR id_attr="17.a1" name="salary">
        acp3
      </ATTR>
      <ATTR id_attr="17.a2" name="date">
        acp3 acp4
      </ATTR>
      <TEXT>acp3 acp4</TEXT>
    </Position>
  </Career>
  ...
</Employee_dossier>
```

Fig. 12 Marking of the subtrees rooted at `Evaluation` and `Career` elements of the `Employee_dossier` document in Fig. 1, according to the $\mathcal{ACPB}$ and $\mathcal{SPB}$ shown in Figs. 4 and 6

configuration in the XML document, a method, called `Reconciliate`, that notifies the SA of overlaps among signature policies is invoked and the nodeset with respect to which the overlap arises is displayed. Then, the SA can decide how to combine the overlapping policies to associate with them only one signature policy, selecting from a menu one among *And, Choice* and *Concatenation* operators. Information on the selected operator are stored into an XML document, called `Reconciliation`. More precisely, for each overlapping signature policy, the document contains an element, called `Configuration`, consisting of two attributes: `Config` and `Operator`. The `Config` attribute provides the identifiers of the overlapping signature policies, whereas the `Operator` attribute specifies the operator selected by the SA in order to reconcile the overlapping policies.

Example 8 Figure 13 shows an example of the `Reconciliation` document created by the *Reconciliation* component with respect to the document marking partially shown in Fig. 12. It specifies the reconciliation operators chosen by the SA for combining the overlapping policies `sp2` and `sp3` on the `psychological_eval` element.

```
<Configurations>
  <Configuration Config="sp2sp3" Operator="Concatenation">
</Configurations>
```

Fig. 13 An example of `Reconciliation` document

5.1.3 *Encrypt_then_Sign and Sign_then_Encrypt components*

The *Encryption and Signature* module includes two major components: *Encrypt_then_Sign* and *Sign_then_Encrypt*. The first one operates according to the *encrypt-then-sign* strategy described before, whereas the second one applies the *sign_ then_encrypt* strategy. Both components generate and represent document encryption and signature according to the W3C `XML Encryption` and `XML Signature` standards [43,44]. However, both standards have been extended to make them suitable for our requirements, (e.g., a finer encryption granularity, a compact representation of joint signatures). Such extensions are described in Sects. 5.1.4 and 5.1.5, respectively. Both components executes the *BuildSets* method to create the sets MARK1 and MARK2, whose elements are the identifiers of the access control and signature policy configurations contained in the document marking. The method also generates a set E_l for each access control policy configuration l in MARK1 and for each policy configuration l $\notin$MARK1 containing only a positive access control policy in the policy list created from the input $\mathcal{ACPB}$. Each of these sets, E_l with $l \in L$ $=$MARK1$\cup$ $\{\{\mathtt{acp}i\}$ *where* $\mathtt{acp}i$ *is a positive policy in* $\mathcal{ACPB}$ *and* $\{\mathtt{acp}i\} \notin$ MARK1$\}$, contains the XPath expressions of those document nodes marked with the access control policy configuration l. Then, the *Encrypt_then_Sign* component generates a set T_m, for each signature policy configuration identified by m in MARK2. T_m contains the XPath expressions of document nodes to which signature policy configuration m applies. Sets MARK1 and MARK2 are implemented as instances of the Vector Java class. Sets E_l, $l \in L$, instead, are implemented as instances of a Java class, called `SetEl`, that has three class variables `l`, `elements` and `key` `l` is a string representing the identifier of the access control policy configurations; `elements` is an instance of the Vector Java class that contains the XPath expressions identifying those nodes to which the configuration l applies; `key` is a string representing the value of the encryption key with whom all the nodes belonging to `elements` must be encrypted. By contrast, the sets T_m, m $\in$MARK2 are represented as instances of a Java class, called `SetTm`, that has two class variables `m` and `elements`: `m` is a string that represents the signature policy configuration identifier, whereas `elements` is an instance of the Vector Java class containing the XPath expressions denoting the nodes to which configuration m applies. After executing *BuildSets*, the *generate_keys* method is invoked in order to generate an encryption key for each $l \in L$, which is used to encrypt the nodes to which the access control configuration l is applied. Then, an instance of the class *ClassHierarchy* is created. The constructor of this class creates the hierarchy of access classes presented in Sect. 4. The vertexes of the class are instances of the class LabeledVertex representing access control policy configurations. Each instance of LabeledVertex has two class variables: id representing the policy configuration and *key* representing the secret key associated with the access control policy configuration. The arcs between the vertexes are instances of the class DirectedLabeledEdge: they represent the partial order relation among access control policies configurations. The class DirectedLabeledEdge has a class variable called label that is the result of the *XOR* operation between the hash value of the id value of the LabeledVertex target of the edge and the symmetric key of the LabeledVertex source of the edge, and the symmetric key of the LabeledVertex source of the edge. The hierarchy is used, later on, by the distribution module, whose implementation is ongoing, to decide which decryption keys must be sent to the subjects. The distribution system will send only the decryption keys corresponding to access classes that are not dominated by any other class in the hierarchy. Hence, the *encrypt_doc* method performs the encryption, node by node, of the input document d by using the TripleDES algorithm [40]. The resulting encrypted document d^e is modeled in a format compliant with the XML Encryption standard (see Sect. 5.1.4 for a detailed description and an example of the document encryption). Finally, the nodes on which no access control policy applies are replaced with an element called Node. The resulting encrypted document is stored into the XMLStore. After the encryption, the *Encrypt_then_Sign* component creates and stores in the XMLStore an XML document, called Encryption_Info, containing all the information necessary to correctly decrypt the ciphered document. This document contains an element, called Configuration, for each access control policy configuration in L. Each Configuration element has one Id attribute and two child nodes: Nodes and Key. The Id attribute identifies an access control policy configuration. The Nodes element, instead, specifies the XPath expressions denoting the nodes to which the configuration applies, whereas the Key node contains the key used to encrypt the specified nodes. The Key element is optional: when an access control policy configuration is dominated by other policy configurations in the hierarchy previously created, the corresponding Configuration element has no Key child node. The client module will retrieve the information to derive the key related to the access control policy configurations corresponding to Configuration elements not having a Key subelement. Then, the *GenerateSignatures* method is activated to sign the encrypted document by using the DSA-SHA1 algorithm [40]. For each signature policies configuration m in MARK2 the set T_m containing the XPath expressions denoting the nodes in document d to which m applies is associated with m. Then, if m identifies a non overlapping signature policy configuration, the XML Signature related to the encrypted nodeset specified in T_m is created. Otherwise, if m is an overlapping signature policy configuration, first, the Operator chosen by the SA to manage the overlap is selected from the Reconciliation document. Then, the XML Signature required by the signature policy resulting from the application of Operator is created. All the signatures required by the specified signature policies are created according to a format compliant to the XML Signature standard (see Sect. 5.1.5 for the format description along with one example of a generated signature).

By contrast, after creating the sets MARK1, MARK2 and E_l, the *BuildSets* method executed by the *Sign_then_ Encrypt* component generates, for each signature policy configuration m in MARK2, the sets T_{lm} with l in L, holding the XPath expressions of nodes to which signature policy configuration m and access control policy configuration l apply. Then, the *Sign_then_ Encrypt* component performs the *GenerateSignatures* method to generate the signature. First, each set T_{lm} is associated with the signature policies configuration m in MARK2. For each set T_{lm}, an XML Signature element associated with m is created according to the signature policy m or the policy resulting from the reconciliation phase, as described for the other component. Afterwards, the encryption process is performed: the *generate_keys* method generates an encryption key k for each access control policy configuration l in L. Then, an instance of the class *ClassHierarchy* is created to generate the hierarchy of access classes. After that, the *encrypt_doc* method encrypts the document portion on which l applies together with the signature affixed on such portion. The resulting encrypted document d^e is generated and represented as previously explained. Finally, the XML document Encryption_Info is created and stored into the XMLStore.

5.1.4 XMLEncryption extensions

To represent the encrypted version of the XML document to be securely disseminated, we have chosen to be compliant, as much as possible, with the XML Encryption standard [43]. Such standard provides an XML format to represent data encryption as well as all the information needed for the decryption operation. According to the standard, the encryption of a datum is an XML document whose root element is referred to as EncryptedData. Besides optionally specifying the algorithm and the (encrypted) key used for the encryption operation, the `EncryptedData` element contains the `CipherData` element, that envelopes the cipher datum within its `CipherValue` subelement, or references it within the `CipherReference` subelement by means of an URI attribute. The encryption granularity allowed by the standard allows one to represent the encryption of arbitrary data (XML documents included), an XML element, or an element content, considered as the text (representing both markup or character data) between the start and end tag of the element. Hence, the standard does not allow the selective encryption of the data content of the element, its attributes and its subelements. Furthermore, the current standard does not allow nesting an `EncryptedData` element within another one.

However, as previously explained, our system allows access control at a finer granularity level than the element one, in that different access requirements can be specified at attribute level, and, thus, a different encryption of start and end tags, element data content, and attributes can be needed. Additionally, it is useful to maintain the structure (that is, the branching relationships among all the nodes) of the clear-text document also on the resulting encrypted document, in order to be able to use XPath during decryption. Therefore, some extensions are needed to the `XML Encryption` standard in order to meet the requirements of our system. First of all, we allow the nesting of `EncryptedData` elements, in such a way to maintain at `EncryptedData` level the structure of the input document. Second, we have extended the `XML Encryption` standard in order to allow the selective encryption at attribute, element data content, and tag level. For this purpose, we have extended the domain of the *Type* attribute of the `EncryptedData` element to add the values `Attribute`, `Text` and `Tag` (the `Text` value has been inserted to denote only the element data content).

Example 9 Figure 14 presents the encrypted version of the elements `Career` and `Position` of our running example, whose identifiers in the document marking illustrated in Fig. 12 are 16 and 17, respectively. Based on the document marking, the entire `Career` element is marked with the same access control policy configuration, that is `{acp3,acp4}`. Therefore, both the element tag and its *date* attribute are ciphered with the same key as a unique data item. The `Career` element is therefore replaced by an `EncryptedData` element, with a child node, called `CipherData`, that envelopes the encrypted element. It contains two attributes, *Id*, specifying the id value (16) of the clear-text element, and *Type*, whose value is the string `Element`. On the contrary, the *salary* attribute of the `Position` element is labeled with a different access control policy configuration, that is, `{acp3}`, with respect to the other element portions. Therefore, the start and end tags, the element content and the *salary* and *date* attributes are separately encrypted, and each of them is replaced, within the encrypted document d^e, by an `EncryptedData` element containing the cyphered datum, as shown by Fig. 14. To maintain, as much as possible, the structure of the input document, the `EncryptedData` element related to the element tags is inserted as a child node of the `EncryptedData` element conveying the encryption of the `Career` element. Moreover, the `EncryptedData` elements corresponding to the attributes and data content of the `Position` element are appended to the encrypted element corresponding to the tags of the `Position` element.

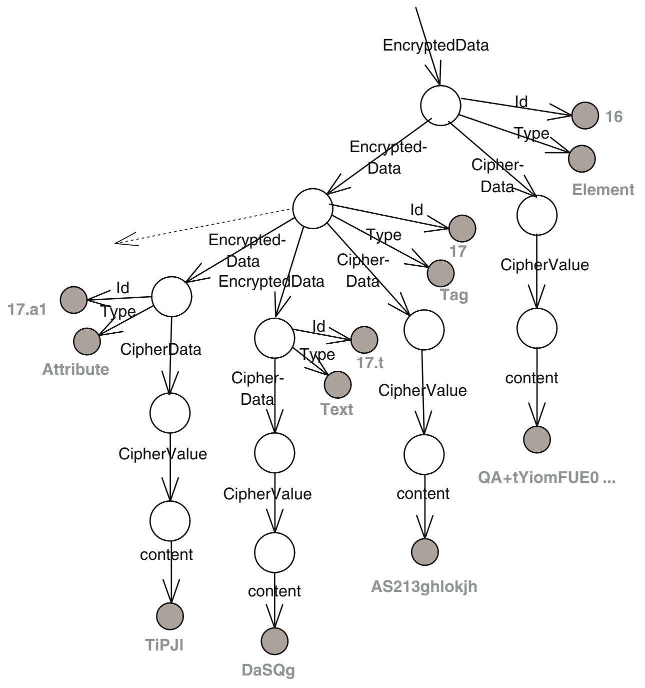


Fig. 14 Graph representation of the encryption of elements `Career` and `Position`, according to the $\mathcal{ACPB}$ presented in Fig. 4

5.1.5 XML Signature extensions

All signatures generated by our system along with information required for the signature validation are modeled in a format compliant with the `XML Signature` standard [44]. Such standard specifies an XML format to represent a signature of a datum together with all the information related to the signature generation that are relevant for its validation. The standard allows one to digitally sign any digital content (data object), including XML. Digital signatures are represented as an XML element, called `Signature`, that contains all information about the signature such as the data object being signed, the algorithm used for signature generation, the signature value and the signer's public key. The `Signature` element contains a `Reference` element for each data object being signed. All the `Reference` elements are collected into a `SignedInfo` element, which contains the information that is actually signed. The `SignatureValue` element contains the encrypted digest of the canonical form of the whole `SignedInfo` element. The `KeyInfo` element, instead, indicates the key to be used to validate the signature. The possibility our system supports of requiring a joint signature by more than one subject on the same authentication object requires an extension to the standard in order to be able to represent, within a single XML Signature, several signature values computed over the same `SignedInfo` element. We thus need to specify within a single XML `Signature` more than one signature value (corresponding to the same `SignedInfo` element) along with the signer. To achieve this goal, we have inserted a `Subjects` child node in the `Signature` element. This element can contain more than one `Subject` element, which in turn contains two child nodes: `SignatureValue`, specifying the canonical form of the `SignedInfo` element encrypted with the private key of the subject, and `KeyInfo`, containing the subject's public key.

Example 10 Figure 15 shows the XML Signature document containing the signatures affixed on the `psychological_eval` element (with Id attribute equal to 13) by a personnel consultant (subj 1, for instance, John Watson which is a personnel consultant as expressed by his $\mathcal{X}$-profile partially shown in Fig. 2) and by a psychologist (subj 2), according to the $\mathcal{SPB}$ and `Reconciliation` document specified for the `Employee_dossier` document. As we can see from Fig. 15, each of the two signatures generated by the two signers are inserted as a child node of a `Subject` element which is in turn contained into a `Subjects` element within the XML Signature document.

5.2 Client implementation

The client side is implemented by three Java components, called *Client*, *Verify_then_Decrypt* and *Decrypt_then_Verify*. Additionally, an `XMLStore` has been created, which contains the ciphered document broadcast by the distribution module, a view of the `Encryption_Info` document, generated on server side, that contains only those decryption keys—ciphered with the

```
<?xml version="1.0" encoding="UTF-8"?>
<Signature>
  <SignedInfo>
    <CanonicalizationMethod Algorithm=
      "http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <SignatureMethod Algorithm=
      "http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    <Reference Id="1" URI="CryptoDoc.xml">
      <Transforms>
        <Transform Algorithm=
          "http://www.w3.org/2002/06/xmldsig-filter2">
          <XPath Filter="intersect">//*[@Id="13"]</XPath>
        </Transform>
      </Transforms>
      <DigestMethod Algorithm=
        "http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue> tD5SBq5+89WCNSl= </DigestValue>
    </Reference>
  </SignedInfo>
  <Subjects>
    <Subject Id="1">
      <SignatureValue> L7aEtQfAhUAm0q...</SignatureValue>
      <KeyInfo>
        <KeyValue>
          <DSAKeyValue>
          ...
          </DSAKeyValue>
        </KeyValue>
      </KeyInfo>
    </Subject>
    <Subject Id="2">
      <SignatureValue> MC0CFmEpdexJ...</SignatureValue>
      <KeyInfo>
        <KeyValue>
          <DSAKeyValue>
          ...
          </DSAKeyValue>
        </KeyValue>
      </KeyInfo>
    </Subject>
  </Subjects>
</Signature>
```

Fig. 15 The XML Signature of the `psychological_eval` element according to the $\mathcal{SPB}$ and `Reconciliation` document shown in Figs. 6 and 13

subject's public key—corresponding to the dominant access classes for which the receiver has the proper authorization, along with the references to such portions (in terms of XPath compliant expressions), and, finally, the XML documents containing the XML signatures authenticating the portions the receiver can access. The *Client* component is the main class of the client side: it receives as input the names of the `XMLStore` and the documents it contains. The component creates a session with the eXcelon's `XMLStore` [18] to extract the ciphered document, the `Encryption_Info` document and the signature(s). Then, it executes the decryption and signatures validation process. The *Verify_then_Decrypt* component realizes the *verify-then-decrypt* strategy. Such a component is activated if, on server side, the *Encrypt_then_Sign* component has been executed to create the encrypted and signed document. Otherwise, the *Decrypt_then_Verify* component is executed. Both of them first execute the `Key_derivation` method, which derives the decryption keys of those access control policy configurations whose keys have not been sent. These policy configurations are the ones represented in the `Encryption_Info` document by a `Configuration` element with no `Key` child node. The *Verify_then_Decrypt* component first checks the validity of all the XML Signatures applied to the encrypted document and stored in the XMLStore. The component executes the *Verify_Signatures* method performing the reference and signature validation, as described by the W3C Recommendation [44], with some differences due to the extensions we have made to the current standard. More specifically, in the validation process, for each Subject specified in the Subjects element contained in the XML Signature, the signature value specified by the corresponding element is validated by means of the public key inserted into the corresponding KeyInfo node. If the validation of a signature value fails, the component informs the SA about which document portions proved not to be authentic. Otherwise, if the validation of all signature values successfully ends for all the XML Signatures in the XMLStore, the *Decrypt* method is executed. It adopts a decryption strategy node by node that firstly, checks, for each EncryptedData element in d^e, if the Encryption_Info contains the key necessary to decrypt it. If this is the case, the content of the CipherData element is decoded with the selected key and the EncryptedData element is replaced by the corresponding deciphered element. If the EncryptedData element cannot be decrypted, it is replaced with an element called Node. The resulting document is, then, stored in the XMLStore.

On the contrary, the *Decrypt_then_Verify* component, first performs the *Decrypt* method to decrypt the document portions the receiver can access along with the XML Signatures affixed on these portions, and then it validates the received XML Signature(s) through the *Verify_Signatures* method.

6 Performance evaluation

We have carried out an extensive performance evaluation of the implemented system to understand which parameters affect its effectiveness and in which degree. There are many dimensions that could influence the performance of our system, as we extensively discuss in the next section: the number of specified security policies, the number of nodes of the XML document to be protected, the number of existing overlaps among different signature policies, and so on. Therefore, we have executed a set of experiments in order to evaluate system performance along these dimensions. Besides reporting the performance of the overall system we have performed specific tests on each system module (i.e., *Marking*, *Reconciliation*, *Encryption and Signature* together with the client *Verification and Decryption* module). Additionally, we have performed a number of experiments to evaluate and compare the performance of the two strategies our system supports (that is, the *encrypt-then-sign* and *sign-then-encrypt* on server side, and the correspondent *verify-then-decrypt* and *decrypt-then-verify* on the client side) taking into account the different characteristics of the two approaches.

In what follows, we first illustrate the parameters we have identified which affect the performance of our system. Then we describe the dataset we have chosen to evaluate each system module. Finally, we present and discuss the most significant results obtained from the experiments we have executed.

6.1 Relevant parameters

For each software component implementing a system module, we have identified all the parameters affecting its performance. In the following, we discuss such parameters.

The labeling operation executed by the *Marking* component, obviously, depends on the number of security policies with which each document node must be marked. Also the type of specified access control policies (positive vs. negative) differently affects the time required by the marking operation, because a negative policy causes the deletion of the label of the positive policy it overwrites from the specified nodes. Another parameter affecting the performance of the labeling task is the number of TAG configurations, due to the fine granularity of the specified policies. Finally, the structure of the input document and, especially, the number of nodes is an essential parameter to assess the efficiency of the marking operation. In evaluating the performance of the *Reconciliation* component, instead, a parameter to take into account is the number of overlaps among different signature policies. Additionally, for each single overlap the number of overlapping policies affects the component performance, together with the structure of the document region on which the overlap arises and, especially, the number of nodes belonging to such a region. Finally, the performance of both the *Encrypt_then_Sign* and *Sign_then_Encrypt* components is obviously affected by the structure of the input document and, above all, by the document size in terms of number of nodes. Additionally, the performance of the two components also depends on the number of access regions marked with an access control policy configuration in the document marking, because for each region a different secret key has to be generated and managed. The number of signature regions also affects the efficiency of the two components (affecting the number of signature operations to be performed on the given document). Moreover, for each region a relevant parameter to assess is the number of signers required by the signature policy. However, we have to point out that, for the same number of signature regions, the *Sign_then_Encrypt* component could require a greater number of signature operations than the *Encrypt-then-Sign* component. More precisely, if a signature policy requires that one or more subjects sign different portions of a document, each of which must be encrypted with a different key, the *Encrypt_then_Sign* component requires the generation of only one XML Signature; the other component, instead, requires the generation of an XML Signature for each portion ciphered with the same key, in order to allow the signature validation of each access region intersecting the considered signature region. This behavior could impact the performance of the *Decrypt_then_Verify* component, on the client side. In any case, the *Verify_then_Decrypt* component has to validate a single XML Signature, whereas the *Decrypt_then_Verify* component could be required to validate more than one signature. Therefore, the performance of the *Sign_then_Encrypt* component, on server side, and the corresponding *Decrypt_then_Verify* component, on client side, also depend on the number of intersections between access and signature regions.

If each signature region intersects only one access region, that is, the overall intersection number is equal to the number of signature regions, then the two components perform the same number of signature operations. Otherwise, if such intersection number is higher than the number of signature regions, the *Sign_then_Encrypt* component has to execute a number of signature operations higher than the number of regions to be differently signed. We have, thus, performed a number of experiments to investigate and compare the performance trends of the two components, *Encrypt_then_Sign* and *Sign_then_Encrypt*, on server side, along with *Verify_then_Decrypt* and *Decrypt_then_Verify*, on client side, when increasing the overall number of intersections between access and signature regions. Additionally, we have investigated and compared the performance of both the two components when increasing the number of nodes of the input document.

Table 3 summarizes, for each system component, the main parameters considered during performance evaluation.

Table 3 Parameters affecting the performance of our system

Considered parameters	
Component	*Parameters*
`Marking`	- `Number of` security policies, - `Type of` access control policies, - `Number of` document nodes, - `Number of` TAG configurations
`Reconciliation`	- `Number of` overlaps, - `Number of` overlapping policies, - `size of` the region on which an overlap arises
`Encrypt_then_Sign` & `Verify_then_Decrypt`	- `Number of` document nodes, - `Number of` access regions, - `Number of` signature regions & signers
`Sign_then_Encrypt` & `Decrypt_then_Verify`	- `Number of` document nodes, - `Number of` access regions, - `Number of` signature regions & signers, - `Number of` of intersections between access and signature regions

6.2 Datasets

In what follows, we describe the datasets we have generated to evaluate the performance of each system component by varying each parameter illustrated in Table 3, when maintaining the other ones fixed.

To assess the efficiency of the *Marking* component with respect to the number of security policies specified in the policy bases, we have considered an `Employee_dossier` document, which is an extension of the one illustrated in Fig. 1: it contains 200 nodes, 100 of which are elements. Then, we have generated three pairs of policy base documents, $\mathcal{ACPB}$_10 and $\mathcal{SPB}$_10, $\mathcal{ACPB}$_50 and $\mathcal{SPB}$_50, $\mathcal{ACPB}$_100 and $\mathcal{SPB}$_100, in such a way to increase the number of both the access control and signature policies specified for the base `Employee_dossier` document (containing 200 nodes), where $\mathcal{ACPB}\_i$ and $\mathcal{SPB}\_i$ specifies i policies, $\forall i = 10, 50, 100$. As for access control, only positive policies have been specified. Furthermore, to evaluate how much the performance of the *Marking* component is affected by the policy type (positive vs. negative), we have generated the access control policy bases $\mathcal{ACPB}$_15p and $\mathcal{ACPB}$_20p which add, respectively, 5 and 10 positive policies to the policy base $\mathcal{ACPB}$_10. Then, the policy bases $\mathcal{ACPB}$_5n and $\mathcal{ACPB}$_10n, have been defined, where the previous 5 and 10 positive policies have been changed into negative ones. Then, in order to analyze the effect of the document size on the marking efficiency, we have increased the number of nodes of the base `Employee_dossier` document to 600, 1,000, 1,400, and 1,800 nodes, by repeating specific subtrees of the base `Employee_dossier` document, while maintaining for each resulting document the element number equal to the half of the whole node number. All such documents are conforming to the same DTD. Finally, the access control policy bases $\mathcal{ACPB}$_1_TAG and $\mathcal{ACPB}$_10_TAG, have been created: they contain 10 positive access control policies and determine, respectively, 1 and 10 TAG configurations.

To evaluate the performance of the *Reconciliation* component for increasing values of the number of overlaps, we have considered the base `Employee_dossier` document, the access control policy base $\mathcal{ACPB}$_10, generated to test the *Marking* component, along with the signature policy base $\mathcal{SPB}$_10, which specifies 10 non-overlapping signature policies.

Then, the policy bases $\mathcal{SPB}$_1*triv-overlap*, $\mathcal{SPB}$_1*overlap* and $\mathcal{SPB}$_10*overlaps* have been generated (all of them specify 10 signature policies), where the first document provides one trivial overlap between two signature policies on a single element, the last two, instead, specify, respectively, 1 and 10 overlaps between two signature policies on the same element. By contrast, to vary the number of signature policies overlapping on the same element, we have defined the policy bases $\mathcal{SPB}$_1*overlap*_5 *pol* and $\mathcal{SPB}$_1*overlap*_10*pol* providing one overlap among 5 and 10 signature policies, respectively. Finally, policy bases $\mathcal{SPB}$_1*overlap*_5*nodes* and $\mathcal{SPB}$_1*overlap*_10 *nodes* have been generated to analyze the efficiency of the *Reconciliation* component when increasing the size of the region on which an overlap arises.

To evaluate the performance of the encrypting and signing components, on server side, and the corresponding decryption and signature validation components, on client side, when the number of access regions increases, the access control policy bases $\mathcal{ACPB}$_10, $\mathcal{ACPB}$_50 and $\mathcal{ACPB}$_100, and the same signature policy base $\mathcal{SPB}$_100 have been considered in such a way that each signature region intersects only one access region (the number of access regions intersecting a same signature region altogether, for every signature region, is equal to 100, that is, the fixed number of signature regions). All these policy bases are defined for the base `Employee_dossier` document. For what concerns the signature regions, the signature policy bases $\mathcal{SPB}$_10, $\mathcal{SPB}$_50 and $\mathcal{SPB}$_100 have been considered together with the policy base $\mathcal{ACPB}$_10, all defined on the base `Employee_dossier` document. Moreover, we focus on a single signature region and we have recorded the signature times when increasing the number of signers (from 1 to 10). Finally, to assess the performance of the *Sign_then_Encrypt* component, on the server side, and the corresponding *Decrypt_then_Verify*, on the client side, and compare them with the alternative ones, we have increased the number of intersections between access and signature regions, while maintaining the number of access and signature regions fixed. The signature policy base $\mathcal{SPB}$_10 is considered, while, ten different access control policy bases: $\mathcal{ACPB}$_10k`int`, $k \in N, 1 \leq k \leq 10$ have been defined. All of them specify 10 access control policies, in such a way that 10k access regions specified by the access control policy base, $\mathcal{ACPB}$_10k`int`, $k \in N, 1 \leq k \leq 10$ intersect the signature regions determined by the policy base $\mathcal{SPB}$_10. In case of 10 intersections, there is a perfect overlap between the 10 access and signature regions, therefore both components create 10 XML Signatures. We have analyzed the performance of the two strategies when increasing such intersection number, from 10 to 100 first on the base `Employee_dossier` document and subsequently on the input documents with 600, 1,000 and 1,400 nodes, previously described. Finally, we have compared the performance of the two components when increasing the document nodes from 200 to 1,800 for 10 and 100 intersections between access and signature regions.

We point out that each policy base has been defined in such a way that all the document portions are subject to both an access control and a signature policy. The whole input document is, thus, encrypted and digitally signed.

6.3 Experimental results

We have performed our experiments on an AMD Athlon 64 Processor 3000 with 2 GHz processor and with 512 MB of RAM, under Microsoft Windows XP Professional OS. In the testing phase, the same work load of the system was ensured. Additionally, for each test eleven trials have been executed and the averages of the results obtained from the last ten trials have been computed, excluding the results of the first test. The performance of the server and client system has been measured in terms of CPU time (in milliseconds).

In the following, we present and analyze for each system component the most relevant results obtained from the performed experiments, and we compare the efficiency of the *Encrypt_then_Sign* and *Sign_then_Encrypt* components, on server side, along with the *Verify_then_Decrypt* and *Decrypt_then_Verify*, on client side.

6.3.1 Marking

The performance of the *Marking* component proved to be mainly affected by the dimension of the input document in terms of the number of document nodes. Figure 16 shows that the time required by the *Marking* component quickly grows (from 1.5 to 9 *s*) when increasing the number of nodes of the `Employee_dossier` document to be marked. The number of security policies with which the input document has to be labeled has also an impact on the efficiency of the marking operation. As we expected, the time required by the *Marking* component grows when the number of specified security policies increases. In particular, we have found an

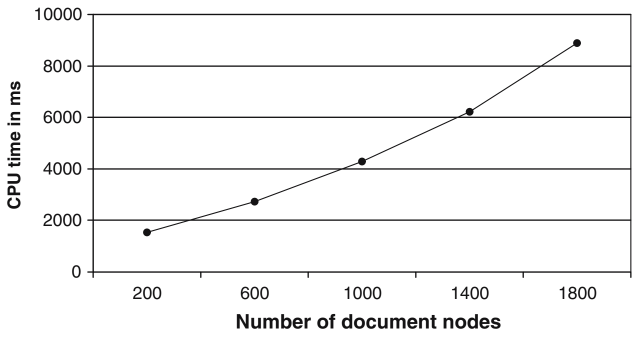


Fig. 16 Marking performance when increasing the number of nodes of the input document

increase around 1 s between the cases with 20 and 200 security policies. The reason of such trend is that an increase in the number of security policies implies a higher time to create the lists of access control and signature policies and to traverse the merged list, with the aim to determine the identifiers of all the policies that apply to each node of the input document. In this experiment, only positive access control policies have been considered. Additionally, we observed that a positive access control policy is as expensive as a negative one, which applies to the same document region. By contrast, a TAG configuration results in a slightly higher marking time than the case without TAG configuration, because in addition a different labeling operation of the start and end tag of an element has to be executed. Obviously, a higher number of TAG configurations much affects the performance of the marking operations (around 40 ms in case of 10 TAG configuration).

6.3.2 Reconciliation

The time requirements of the *Reconciliation* component grow when increasing the number of overlaps between two signature policies on a single element, because an increase in the number of overlaps implies a higher reconciliation time. For each overlap, the component has to perform: (1) the verification of the overlap type (trivial or not); (2) the reconciliation of the overlapping signature policies according to the operator specified by the SA; and (3) the creation of the corresponding element in the `Reconciliation` document specifying the operator selected by the SA. We found that, in case of a trivial overlap, the reconciliation takes more than half the time required to solve a non trivial one. Such time is due, above all, to the time required to check the policy overlap type (trivial or not), which is higher than the time taken by the actual reconciliation phase and the insertion of the selected operator in the `Reconciliation` document. In case of 10 overlaps the time required by the *Reconciliation* component is about 150 ms. Obviously, for each single overlap, a number of overlapping signature policies higher than two implies a higher reconciliation time. In case of a single overlap among 10 signature policies, the time slightly increases of about 42 ms with respect to the case with two overlapping policies. Indeed, the number of policies to be compared in order to verify if the overlap is trivial increases. Our experiments, however, proved that the size of the document region on which an overlap arises does not much affect the time required for the reconciliation.

6.3.3 Encrypt_then_Sign vs. Sign_then_Encrypt

Here, we present and compare the performance of the *Encrypt_then_Sign* and *Sign_then_Encrypt* components when varying the parameters we have identified.

Figure 17a shows the performance of the two components when the number of access regions increases (from 10 to 100), while the number of signature regions is fixed to 100 (which is also equal to the number of intersections between access and signature regions). In such a case, both the components have to create the same number of XML Signatures, which is equal to 100. We notice that the time requirements of the *Encrypt_then_Sign* component slowly grows (from 12.5 to 13 s) when increasing the number of access regions, because, even if a higher number of secret keys are generated and managed, the size of the document to be encrypted is fixed.

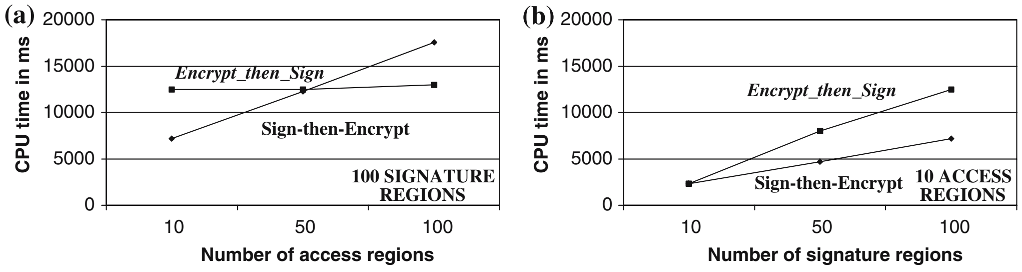


Fig. 17 CPU time requirements of the two server components when increasing **a** the number of access, and **b** signature regions

The time requirements of the *Sign_then_Encrypt* component, instead, shows a linear increase (from 7 to 17.5 s), because the component has to encrypt also the 100 XML Signatures with the same keys by which the correspondent access regions are encrypted. Further, we observe from Figure 17a that the *Sign_then_Encrypt* component performs better than the other one for a number of access regions less than or equal to 50. The reason is that, over this value, a finer granularity in encrypting the XML Signatures along with the access regions they refer to is more expensive than affixing all the signatures on the encrypted text, whose dimension is greater than the corresponding clear text, as we will better explain in the following. With respect to the signature regions, Fig. 17b shows the performance of the two components when the number of such regions increases (from 10 to 100), while the number of access regions is set to 10, whereas the intersection number of the two kinds of regions is equal to the number of signature regions. This assures that each time both components generate the same number of XML Signature(s) (which is equal to the number of signature regions). We can notice from Fig. 17b that the times of both the *Encrypt_then_Sign* and *Sign_then_Encrypt* components grow when the number of signature regions increases, because it results in the generation of a higher number of XML Signatures. In this case, we observe that, even if the two components create the same number of XML Signatures, the *Sign_then_Encrypt* component performs better than the other one (only for 10 signature regions the CPU time of the *Encrypt_then_Sign* is very close to the other one). The reason of this trend is due to the fact that the signature process on the encrypted data is more expensive than on clear text. As explained before, we adopt a format compliant to the `XML Encryption` standard [43] to represent the encrypted document according to which the encrypted nodes of the input document are wrapped into a different XML document, whose height is two units greater than that of the input document, and which contains three times the nodes of the clear document. This implies for the *Encrypt_then_Sign* component a greater time than the other component to select the (encrypted) nodes to be signed and to sign them. This observation is strengthened by the results obtained from the experiments to assess the impact of an increase in the number of signers that have to sign a same document region: for each signer, the signature time has been estimated to be about 20 ms, in case the *Sign_then_Encrypt* component is activated, 36 ms, if the other component is executed. However, as we have seen from Fig. 17a, when the number of access regions is close to the number of signature regions the *Encrypt_then_Sign* component is more efficient than the *Sign_then_Encrypt* (see the case when the number of access regions is higher than 50 in Fig. 17a, and the case of 10 signature regions in Fig. 17b), because the signatures on the encrypted text become less expensive than the encryption of the XML Signatures (on the correspondent clear text) along with the higher number of access regions. Anyway, the *Encrypt_then_Sign* component is more efficient than the other one especially when the number of intersections between the access and signature regions is higher than the number of signature regions, because in this case the *Encrypt_then_Sign* component avoids "breaking the signatures", which is, instead, needed by the other component. In particular, we have increased the number of intersections from 10 to 100, while maintaining the number of the access and signature regions equal to 10. In case of 10 intersections, which is also equal to the number of specified signature regions, both components create the same number of XML Signature(s). When the number of intersections is higher than 10, instead, the *Sign_then_Encrypt* component has to create an XML Signature for each different access region intersecting a signature region. Figure 18a shows the performance of the two components while varying the intersection number mentioned above, for the base `Employee_dossier` document containing 200 nodes. The time requirements of the *Encrypt_then_Sign* component are constant when increasing the number of intersections, because this component always generates 10 XML Signatures. The time requirements of the other component, instead, linearly grows, because an increase in the intersection number implies a higher signature time, since the component has to generate a greater number of XML Signatures, as well as a higher encryption time, since an increasing number of signature documents must be encrypted. Figure 18a shows that the *Encrypt_then_Sign* component is always the most efficient (only in case of 10 intersections the times are the same). However, since the signature on the encrypted region is more expensive than on the corresponding clear text, especially when the region dimension increases, the performance of the *Encrypt_then_Sign* component becomes worse more quickly than the other component when increasing the number of nodes, as we can see from Figs. 18b–18d. We notice, indeed, that the abscissa of the intersection point P between the two curves moves to the right: for the base `Employee_dossier` document the two components have the same performance in case of 10 intersections between access and signature regions. However, if we increase the document nodes to 600, 1,000 and 1,400, the performances of the two components become equal, respectively, for 15, 25 and 55 intersections. Additionally, we observe

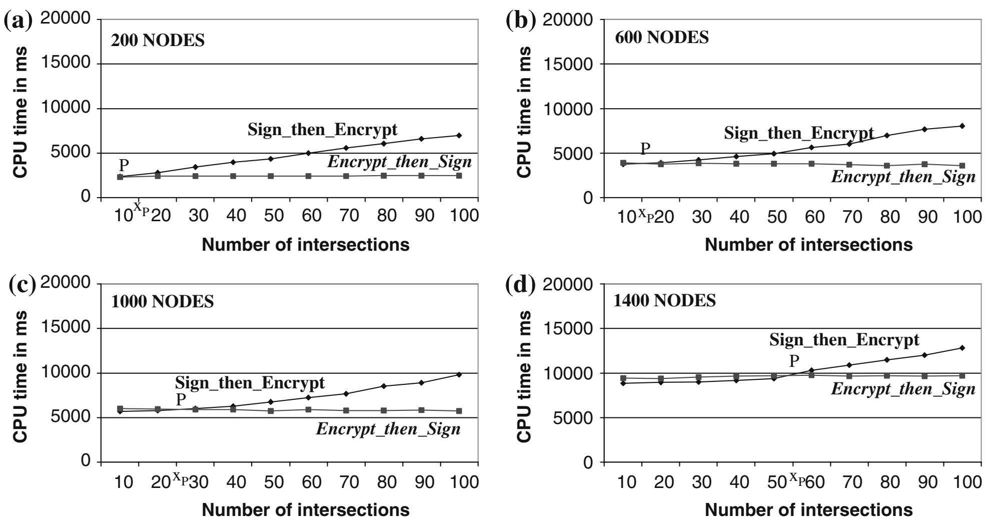


Fig. 18 *Encrypt_then_Sign versus Sign_then_Encrypt* when increasing the number of intersections between access and signature regions for an XML document containing **a** 200, **b** 600, **c** 1,000, and **d** 1,400 nodes

that before P the time requirements of the two components have approximately the same behavior (even if the *Sign_then_Encrypt* times are slightly lower). After P, instead, the *Encrypt_then_Sign* component becomes definitely the most efficient. Finally, Fig. 19a represents on the same graph the CPU times of the *Encrypt_then_Sign* and *Sign_then_Encrypt* components, while varying the number of nodes of the input XML document in case of 10 intersections between access and signature regions. In this case, the two curves are very similar: both grow when the number of nodes increases (from 2,3 s for 200 nodes to about 14 s for 1,800 nodes), showing the same trend (only for input documents containing more than 1,000 nodes the *Sign_then_Encrypt* component performs slightly better than the other one). However, if the intersection number is higher than 10 the *Sign_then_Encrypt* component gets worse than the other one: in case of 100 intersections the *Sign_then_Encrypt* component is always the less efficient for every input document we have considered (200, 600, 1,000, 1,400, 1,800 nodes), as we can see from Fig. 19b.

6.3.4 Overall server system

Here, we briefly discuss the performance of the overall system on server side based on the results previously shown and considering as main parameters the number of access and signature regions, the intersection number between access and signature regions, and, finally, the size of the input document. Then, some discussions about the performance of the two server components to encrypt and sign the input document are presented.

When the number of either access or signature regions increases, the time required by the *Marking* component increases, given the increase in the number of either access control or signature policies specified in the policy bases, as seen in Sect. 6.3.1. The reconciliation times, instead, are zero in both cases, since the considered $\mathcal{SPB}s$ do not result in any overlap among the specified signature policies. Thus, for an increase in the access and signature regions, the CPU time of the system on the server side is given by the marking time plus the time to encrypt and sign the input document performed by either the *Encrypt_then_Sign* or the *Sign_then_Encrypt* components. The performance trends, in terms of CPU times, of the server system are similar to those shown in the previous subsection, which have been observed for the two server components that encrypt and sign the input document. This is due to the fact that the server system performance is mainly affected by the times of the two components for the encryption and signature operations that are more expensive than the marking process. When increasing the number of access regions, the system times are about 16 s, if the *Encrypt_then_Sign* component is performed; they linearly increase from 9 to 20 s, whenever the other component is executed. By contrast, when the number of signature regions grows, the system times increase whatever strategy is applied (from 4 to about 9.3 s for the *Sign_then_Encrypt* component, from about 4 to 14.6 s for the other component). Similarly, when increasing the number of intersections between access and signature regions as well as the number of the document nodes, the system performance is characterized by the same trend registered for the *Encrypt_then_Sign* and *Sign_then_Encrypt* components, that has been previously described.

To conclude, in order to state which server component is the most efficient in encrypting and signing an

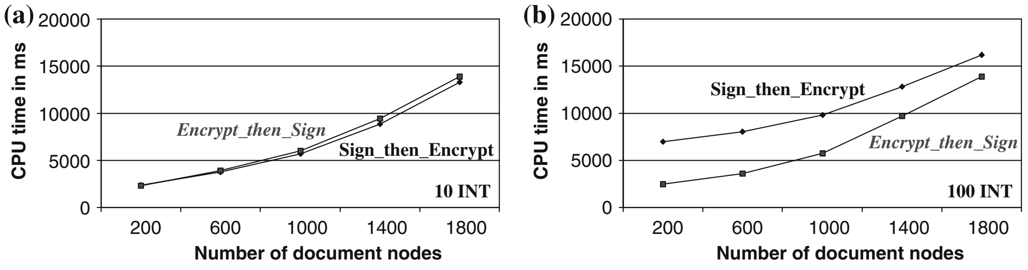


Fig. 19 *Encrypt_then_Sign versus. Sign_then_Encrypt* when increasing the number of document nodes for **a** 10 and **b** 100 intersections between access and signature regions

input document d according to the $\mathcal{ACPB}$ and $\mathcal{SPB}$ specified for such document, we adopt the following method. After the marking process, we compute the number of access and signature regions, N_A and N_S, respectively, together with the number of their intersections I. Additionally, we compute the number N of the document nodes to which both access and signature policies apply. Then, if $I = N_S, N \leq 200$, and the number of access regions is definitely less than the number of signature regions, $N_A << N_S$, the *Sign_then_Encrypt* component is more efficient. By contrast, if $N_A \rightarrow N_S$, the *Encrypt_then_Sign* component is the most efficient. If $I = N_S$ and $N > 200$, instead, the *Sign_then_Encrypt* component is always the most efficient (an increase in the number of nodes affects much more the *Encrypt_then_Sign* component rather than the other one). If $I > N_S$ and $N \leq 200$, the *Encrypt_then_Sign* component better performs for every intersection number $I > N_S$. However, if $N > 200$ the *Encrypt_then_Sign* component is more efficient than the other one whenever I is higher than X_P, that is, the abscissa of the intersection point P (whose trend is illustrated in Fig. 20) between the two curves shown in Figs. 18.

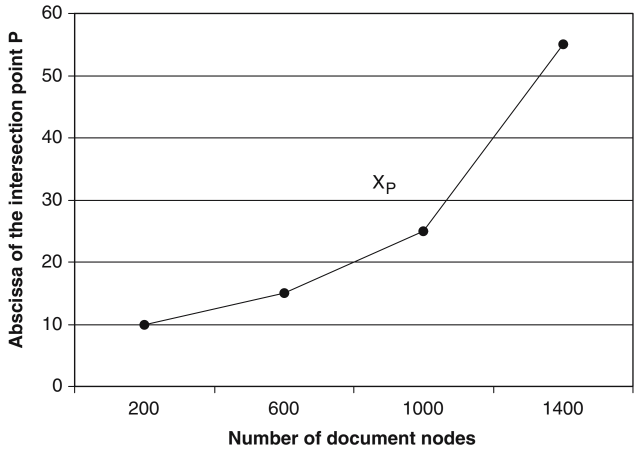


Fig. 20 Abscissa of the intersection point P between the two curves representing the performance of the *Encrypt_then_Sign* and *Sign_then_Encrypt* components when increasing the size of the input document

6.3.5 Verify_then_Decrypt versus Decrypt_then_Verify

As described in Sect. 5, the system on the client side mainly consists of the two alternative components, *Verify_then_Decrypt* and *Decrypt_then_Verify*. Thus, we present the time requirements of the client system by reporting and comparing the times of both such two components. In analyzing such components, we take into account the worst case in which the considered receiving subject can access the whole input document and, thus, s/he has to validate all the XML Signature(s) generated on server side.

Figure 21a shows the CPU times of both the *Verify_then_Decrypt* and *Decrypt_then_Verify* components when increasing the number of access regions, while maintaining fixed to 100 both the numbers of signature regions and intersections between access and signature regions. The time requirements of the *Verify_then_Decrypt* component are constant, since decryption is performed node by node starting from the root element, and it is, thus, independent from the access regions in which the document is split. A higher number of access regions causes, instead, a linear increase in the time requirements of the *Decrypt_then_Verify* component, because it has to decrypt 100 XML Signatures before validating them, and the number of decryption operations it has to perform is equal to the increasing number of access regions. Indeed, all the XML Signatures applied on the same access region are grouped together and cyphered with the same key used to encrypt the access region. With respect to the effect of an increase in the number of signature regions on the efficiency of the client component (while keeping the number of access regions equal to 10, and the number of intersections of the two kinds of regions equal to the number of signature regions), we notice that the time of the *Verify_then_Decrypt* component linearly grows because this component has to validate, in each case, an increasing number of XML Signatures (affixed on the encrypted data) equal to the number of signature regions. By

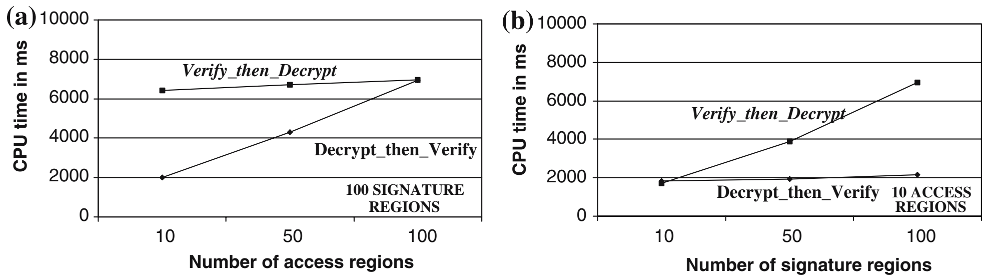


Fig. 21 CPU time requirements of the client system when increasing the number of **a** access, and **b** signature regions

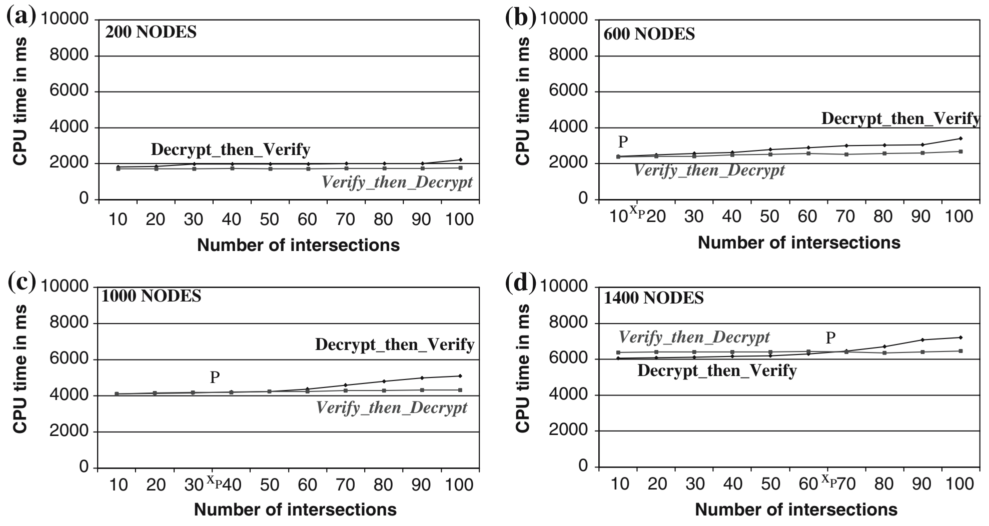
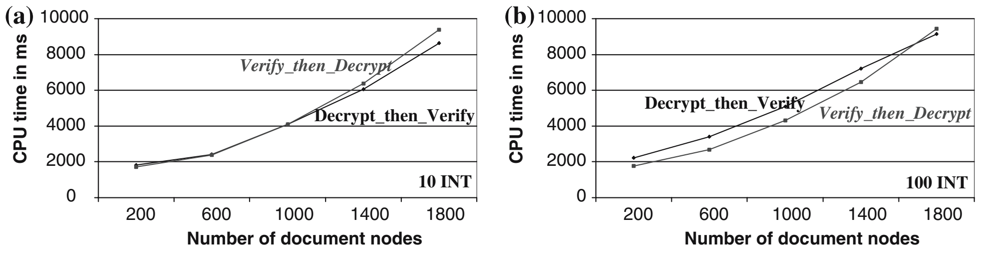


Fig. 22 CPU time requirements of the client system when increasing the number of intersections between access and signature regions for an XML document containing **a** 200, **b** 600, **c** 1,000, and **d** 1,400 nodes

contrast, the time requirements of the *Decrypt_then_Verify* component slowly grow because, in each case, it performs the decryption of the same input document along with the same number of signature documents grouped by the fixed 10 access regions. Moreover, the validation times on clear data slowly grow when increasing the number of signature regions, and, therefore, of the signatures to be validated.

Figure 22 shows the performance of the two client components when increasing the intersection number between access and signature regions for the `Employee_dossier` documents containing 200, 600, 1,000, and 1,400 nodes. The trend of the CPU times we have observed on client side for the *Decrypt_then_Verify* and *Verify_then_Decrypt* components is equal to the *Sign_then_Encrypt* and *Encrypt_then_Sign* trend, on the server side with some differences we explain in what follows. First, the times of the two client components are lower than the ones observed on the server side, because the decryption and signature validation are obviously less expensive than the encryption and signature generation performed on the server side. Second, the *Verify_then_Decrypt* component performs better than the other one for an input document containing 200 and 600 nodes, whatever is the number of intersections. By contrast, for an input document containing more than 600 nodes the performance of the two components depends on the number of intersections between access and signature regions, as we can see from Fig. 22. Finally, Fig. 23a and b show that the performance of both the client components grows when increasing the number of nodes and the number intersections between access and signature regions are equal to 10 and 100, respectively. The most relevant result, which strengthens the results observed on the server side, is the fact that the dimension of the input document affects much more the *Verify_then_Decrypt* component than the other one: we observe that the CPU time requirements of the *Verify_then_Decrypt* component exponentially grow, while the times of the other component are characterized by a linear increase. The reason is that, as previously explained, the signature validation as well as its

Fig. 23 CPU time requirements of client system when increasing the number of nodes of the input document for **a** 10 and **b** 100 intersections



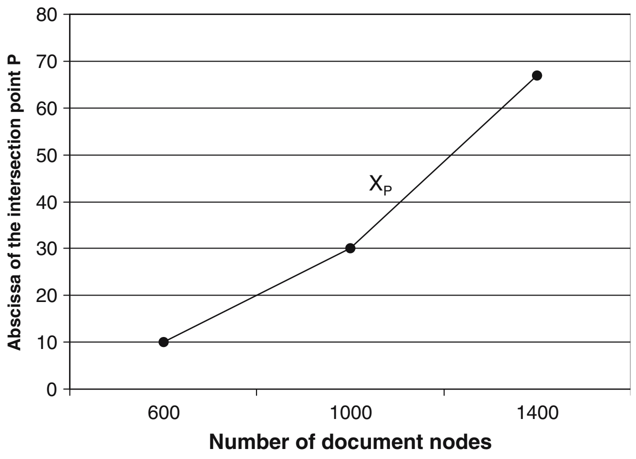


Fig. 24 Abscissa of the intersection point P between the two curves representing the performance of the *Decrypt_then_Verify* and *Verify_then_Decrypt* components when increasing the size of the input document

generation on the encrypted text is strongly affected by the number of document nodes. However, we observe that, if the intersection number is definitely higher than the number of signature regions, the *Verify_then_Decrypt* component performs better than the other one.

To conclude both strategies, as implemented by our system, proved to perform in a reasonable time the working load that a push-based system should probably execute in a real context. Obviously, each component better performs in specific conditions. From the experimental results we can conclude that in order to choose which component adopting on server side for the encryption and signature operations, based on the efficiency of the client components, the number N_S of signature regions along with the intersection number I between access and signature regions and the number N of document nodes must be considered. If $I = N_S$, then the *Sign_then_Encrypt* must be performed on server side, because the corresponding *Decrypt_then_Verify* component is more efficient than the other one. By contrast, if $I > N_S$ and the number N of the document nodes that must be encrypted and signed is less than or equal to 600, the *Encrypt_then_Sign* component has to be performed on the server side, since the corresponding *Verify_then_Decrypt* component has the best performance for any value of the intersection number. On the contrary, for document nodes higher than 600 the choice between the two components on server side depends on both the node number and the intersection number X_P between the access and signature regions (see Fig. 24). Whenever $I \leq X_P$ we can choose of applying indifferently one of the two components, given the small difference of their times. If $I > X_P$, instead, the *Encrypt_then_Sign* component must be executed on server side.

7 Conclusions

In this paper we have described the architecture and the implementation of a system able to meet authenticity and confidentiality requirements in push-based dissemination environments. Additionally, we have presented the evaluation study we have carried out to assess the performance of the implemented system and, above all, of the two strategies it supports, when varying several parameters. The two strategies meeting different security properties proved to efficiently work with different working loads. The system we developed thus provides great flexibility by allowing the SA to choose the strategy that better fits the different characteristics – security requirements, efficiency constraints—of the considered application domain. We believe that the results of our analysis can be a valuable contribution to better customize the security mechanism trading-off between security guarantees and efficiency. As a future work, we plan to complete the implementation of the distribution component which is currently under development and to investigate other methods to securely and efficiently enforce both access control and authenticity properties in data dissemination. Moreover, we plan to enrich the system and the policy language with the possibility of expressing a greater number of authenticity requirements, such as the possibility of specifying anonymous and threshold signature policies. To support this, the system should adopt new signature schemes that provide additional requirements besides the traditional authenticity and non-repudiation properties.

Acknowledgements The work reported in this paper has been partially supported by the EU under the IST Project TrustCoM and by the Sponsors of CERIAS.

A Appendix

In this paper, we adopt a formal model for XML documents (based on [14]), defined as follows. Let $\mathcal{I}_{\mathcal{E}}$ be a set of element identifiers, *Label* a set of element tags and attribute names, and *Value* a set of attribute/element values. An XML document is formally defined as follows.

Definition 1 (**XML document**) An XML document $\mathtt{d}$ is a tuple $\mathtt{d} = (V_{\mathtt{d}}, \bar{v_{\mathtt{d}}}, E_{\mathtt{d}}, \phi_{E_{\mathtt{d}}})$, where

- $V_{\mathtt{d}} = V^e_{\mathtt{d}} \cup V^a_{\mathtt{d}}$ is a set of nodes representing elements and attributes, respectively. Each $v \in V^e_{\mathtt{d}}$ has an associated element identifier $id_v \in \mathcal{I}_{\mathcal{E}}$, whereas each $v \in V^a_{\mathtt{d}}$ has an associated value $val \in Value$;
- $\bar{v_{\mathtt{d}}}$ is a node representing the document element (called *document root*);
- $E_{\mathtt{d}} = E^e_{\mathtt{d}} \cup E^a_{\mathtt{d}} \subseteq V_{\mathtt{d}} \times V_{\mathtt{d}}$ is a set of edges, where $e \in E^e_{\mathtt{d}}$ is an edge representing an element-subelement relationship or a link between elements due to IDREF(s) attributes (called *link edge*), and $e \in E^a_{\mathtt{d}}$ is an edge representing an element-attribute relationship;
- $\phi_{E_{\mathtt{d}}} : E_{\mathtt{d}} \rightarrow Label$ is the edge labeling function.

References

1. Al-Mogren, A., Dunham, M.: Data broadcast classification. In: IEEE pp 221–241 (2005)
2. Atallah, M.J., Frikken, K.B., Blanton, M.: Dynamic and efficient key management for access hierarchies. In: ACM CCS (2005)
3. Bellare, M., Rogaway, P.: Encode-then-encipher encryption: how to exploit nonces or redundancy in plaintexts for efficient cryptography. Advances in Cryptology· Asiacrypt 00 LNCS (1976) (2000)
4. Bertino, E., Carminati, B., Ferrari, E.: A temporal key management scheme for broadcasting XML documents. In: Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS'02) (2002)
5. Bertino, E., Carminati, B., Ferrari, E.: Securing XML data in third-party distribution systems. In: CIKM '05: proceedings of the 14th ACM international conference on information and knowledge management, pp 99–106. ACM Press, New York (2005)
6. Bertino, E. , Castano, S. , Ferrari, E. : Author- x: a comprehensive system for securing XML documents. IEEE Internet Comput. **5**(3), 21–31 (2001)
7. Bertino, E., Ferrari, E.: Secure and selective dissemination of XML documents. ACM Trans. Inform. Syst. Secur. **5**(3), 290–331 (2003)
8. Bertino E., Ferrari, E., Parasiliti Provenza, L.: Signature and access control policies for XML documents. In: Proceedings of 8th European symposium on research in computer security (ESORICS 2003) LNCS 2808(3):1–22 (2003)
9. Bertino, E., Sandhu, R.: Database security — concepts, approaches, and challenges. IEEE Trans. Dependable Secure Comput. **2**(1), 2–19 (2005)
10. Castano S., Fugini M., Martella G., Samarati P.: Secure database systems. In: Diaz O., Piattini M. (eds.), Advanced Databases: Technology and Design, Artech House, London (2000)
11. Chaum D., van Heijst, E.: Group signatures. In: Eurocrypt 91, vol. 547, pp 257–265. Springer, Berlin (1991)
12. Chiou, G.H., Chen, W.T.: Secure broadcasting using the secure lock. IEEE Trans. Softw. Eng. **15**(8) (1989)
13. Desmedt, Y., Frankel, Y.: Threshold cryptosystems. Cryptology Crypto 89:307–315 (1989)
14. Deutsch, A., Fernandez, M., Florescu, D., Levy, A., Suciu, D.: A query language for XML. In: Int'l Conference on World Wide Web. (1999) Available at: http://www.research.att.com/suciu
15. Devanbu, P.T., Gertz, M., Kwong, A.: Flexible authentication of XML documents. J. Compu. Secur. **12**(6), 841–864 (2004)
16. Devanbu, P.T., Gertz, M., Martel, C.U., Stubblebine, S.G. Authentic third-party data publication. In: DBSec, pp 101–112 (2000)
17. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Info. Theory **31**, 469–472 (1985)
18. eXcelon Corporation: eXcelon XML Platform (2001). Available at http://www.exln.com
19. Ferraiolo, D.F., Sandhu, R.S., Gavrila, S.I., Kuhn, D.R., Chandramouli, R.: Proposed NIST standard for role-based access control. TISSEC **4**(3), 224–274 (2001)
20. Fiat A., Noar M. (1994) Broadcast encryption. Advances in Cryptology (Crypto 93) LNCS (773):480–491
21. Gladney H., Lotspiech J. (1997) Safeguarding digital library contents and users. D-Lib Magazine. (1997) Available at http://www.dlib.org/dlib/may97/ibm/05gladney.html
22. Hacigümüs H., Iyer B.R., Li C., Mehrotra S.: Executing sql over encrypted data in the database-service-provider model. In: SIGMOD Conference, pp 216–227 (2002)
23. Hacigümüs H., Mehrotra S., Iyer B.R.: Providing database as a service. In: ICDE, pp 29–38 (2002)
24. IBM: Cryptolope$^{\text{TM}}$ (1996). Available at http://domino.research.ibm.com/comm/wwwr_thinkresearch.nsf/pages/packinginfo396.html
25. List, X.D.M.: Simple API for XML (SAX). (1998) Under the coordination of David Megginson. Available at http://www.saxproject.org/
26. M., B., C., N.: Authenticated encryption: relations among notions and analysis of the generic composition paradigm. ASIACRYPT **5**(3), 290–331 (2000)
27. Malone-Lee, J., Mao, W.: Signcryption using RSA. CT-RSA LNCS (2612), 211–225 (2003)
28. Martel, C.U., Nuckolls, G., Devanbu, P.T., Gertz, M., Kwong, A., Stubblebine, S.G.: A general model for authenticated data structures.. Algorithmica **39**(1), 21–41 (2004)
29. Merkle, R.C.: A certified digital signature. Advances in Cryptology-Crypto '89 (1989)
30. Micali, S., Ohta, K., Reyzin, L.: Accountable-subgroup multisignatures. In: ACM Conference on Computer and Communications Security, pp 245–254. ACM Press, New York (2001)
31. Mykletun, E., Narasimha, M., Tsudik, G.: Authentication and integrity in outsourced databases. In: NDSS (2004)

32. Narasimha, M., Tsudik, G.: Dsac: integrity for outsourced databases with signature aggregation and chaining. In: CIKM, pp 235–236 (2005)
33. Pang, H., Jain, A., Ramamritham, K., Tan, K.L.: Verifying completeness of relational query results in data publishing. In: SIGMOD Conference, pp 407–418 (2005)
34. Pang, H., Tan, K.L.: Authenticating query results in edge computing. In: ICDE, pp 560–571 (2004)
35. Pollmann, C.G.: The XML security page. Available at http://www.nue.et-inf.uni-siegen.de/g̃euer-poll-mann/xml_security.html
36. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**, 120–126 (1978)
37. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: ASIACRYPT 2001, vol. 2248, pp 552–565. Springer, Berlin (2001)
38. Shamir, A.: How to share a secret.. Commun. ACM **22**, 612–613 (1979)
39. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: IEEE Symposium on Security and Privacy, pp 44–55 (2000)
40. Stallings, W.: Network Security Essentials: Applications and Standards. Prentice Hall, Englewood Cliff (2000)
41. W3C: Document Object Model (DOM) (1998) Available at http://www.w3.org/DOM
42. W3C: XML Path Language (XPath). (1999) Available at http://www.w3.org/TR/xpath/
43. W3C: XML-Encryption Syntax and Processing (2000). Available at http://lists.w3.org/Archives/ Public/xml-encryption/2000Aug/att-0001/01-xmlencoverview.html
44. W3C: XML-Signature Syntax and Processing (2002). Available at http://www.w3.org/TR/xmldsig-core/
45. Zhang, J., Varadharajan, V., Mu, I.: Securing XML document sources and their distribution. In: Proceedings of the 18th international conference on advanced information networking and application (AINA'04) (2004)
46. Zheng, Y.: Digital signcryption or how to achieve cost (signature & encryption) $<<$ cost (signature) + cost (encryption). CRYPTO'97 LNCS (1294), 165–179 (1997)
47. Zheng, Y.: Identification, signature and signcryption using high order residues modulo an RSA composite. Public Key Cryptography (PKC 2001) LNCS (1992), 48–63 (2001)

Elisa Bertino is professor of Computer Science and Electrical and Computer Engineering at Purdue University and serves as Research Director of the Center for Education and Research in Information Assurance and Security (CERIAS). She has been a visiting researcher at the IBM Research Laboratory, at the Microelectronics and Computer Technology Corporation, at Rutgers University, at Telcordia Technologies. Prof. Bertino is a Fellow member of IEEE and a Fellow member of ACM and received the 2002 IEEE Computer Society Technical Achievement Award for "For outstanding contributions to database systems and database security and advanced data management systems".

Elena Ferrari is a full professor of Computer Science at the University of Insubria, Italy. She received the MS degree in Computer Science from the University of Milan (Italy) in 1992. In 1998, she received a Ph.D. in Computer Science from the same university. Her research activities are related to various aspects of data management systems, including web security, access control and privacy, multimedia and temporal databases. On these topics she has published more than a hundred scientific publications. Dr. Ferrari is in the Editorial Board of the VLDB Journal and the International Journal of Information Technology. She is a member of the ACM and senior member of IEEE.

Federica Maria Francesca Paci is a Ph.D. student at the University of Milan, Italy. She received a degree in Computer Science from the University of Milan in February 2004 with full marks. During the spring of 2005 and the spring and fall semester of 2006 Federica was a research scholar at Computer Science Department and CERIAS of Purdue University, West Lafayette, USA. Federica has also been involved in the TrustCom project. Her main research interests include the development of access control models for constraint workflow systems, Web services access control models and secure distribution of XML documents.

Loredana Parasiliti Provenza holds a post-doc position at the Department of Computer Science and Communication of the University of Milan (Italy). She received the Laurea degree in Mathematics with full marks from the University of Messina (Italy) in 2001 and the Master's degree in Information and Communication Security from the University of Milan in 2002. In 2006, she took the Ph.D. degree in Computer Science at the University of Milan. Her research interests include specification, design and development of languages and mechanisms for authenticity enforcement in Data Dissemination Systems, XML Security, Visual Interactive Systems, Theory of Visual Languages and Privacy Preserving Data Mining.