

Applying hierarchical and role-based access control to XML documents

Jason Crampton

Information Security Group, Royal Holloway, University of London, United Kingdom. Email: jason.crampton@rhul.ac.uk

W3C Recommendations XML Encryption and XML-Digital Signature can be used to protect the confidentiality of and provide assurances about the integrity of XML documents transmitted over an insecure medium. In this paper we show how to control access to XML documents, once they have been received. This is particularly important for services where updates are sent to subscribers. We describe how certain access control policies for restricting access to XML documents can be enforced by encrypting specified regions of the document. These regions are specified using XPath filters and the policies are based on the hierarchical structure of XML documents. We also describe how techniques for assigning keys to a security lattice can be adapted to minimize the number of keys that are distributed to users, and compare our approach with two other access control frameworks. We then consider the limitations of this approach and describe a view-based technique for implementing more complex access control policies. This approach is applicable to broadcast XML data and also provides a lightweight method for implementing a more conventional access control mechanism that responds to user access requests. Finally, we illustrate how our technique might be deployed using XACML.

Keywords: XML, XPath filter, encryption, hierarchical access control, role-based access control

1. INTRODUCTION

XML is fast becoming the *de facto* format for document-based information exchange. This is particularly evident in emerging standards supporting security features for Web services such as SAML, XACML and WS-*, which are all based on XML.

XML is a markup language like HTML. Unlike HTML, which describes both the content and presentation of documents, XML describes the content and *structure* of documents. A well-formed XML document is based on an XML schema or a document type definition (DTD), which define elements, inter-element relationships and attributes of the data contained in the document.

We use an imaginary ACM catalogue as a running example. The catalogue is sent in encrypted form to ACM sub-

scribers.

The recipients of the catalogue will be given only the encryption keys that can decrypt elements in the document that they are entitled to read.

Figure 1 illustrates the skeleton of an XML document containing information about the ACM catalogue. (The schema on which this document is based can be found in the appendix.) Elements typically contain data, while attributes contain information that will be useful for any subsequent processing of the XML document. The `acm-catalog` element includes the `issue-date` attribute, for example.

The explicit structure of an XML document means that it is easy to identify distinct aspects of the document's content. Hence it becomes possible to specify restrictions to particular information contained *within* an XML document. In short, when data is stored in XML format, we can consider

```

<acm-catalog issue-date=" " issue-number=" ">
  <journal>
    <name>...</name>
    <date>...</date>
    <volume>...</volume>
    <number>...</number>
    <table-of-contents>
      <item>
        <toc-entry>...</toc-entry>
        <page-number>...</page-number>
      </item>
      ...
    </table-of-contents>
    <paper>
      <title>...</title>
      <pages>...</pages>
      <author>...</author>
      <abstract>...</abstract>
      <body>...</body>
      <references>...</references>
      <bibtex-entry>...</bibtex-entry>
    </paper>
    ...
  </journal>
  ...
  <proceedings>
    <conference>
      <name>...</name>
      <year>...</year>
      <location>...</location>
    </conference>
    <table-of-contents>...</table-of-contents>
    <paper>...</paper>
    ...
  </proceedings>
  ...
</acm-catalog>

```

Figure 1 A skeleton of an XML document representing the ACM catalogue

more fine-grained access control than has previously been possible.

There has been considerable interest in controlling access to stored or broadcast XML data, both in academia [4, 6, 15, 18] and from standards bodies [21]. Most authorization frameworks for controlling access to XML data [4, 6, 15, 21] are essentially extensions of the protection matrix model [13]. Recall that the protection matrix essentially encodes authorizations as access triples of the form (s, o, a) , where s is a subject, o is an object and a is an access right.

Damiani *et al.*, for example, define an ‘access authorization’ to be a tuple (s, o, a, p, t) , where p determines whether it is a positive or negative authorization and t determines the type of authorization [6]. The type determines the scope of the authorization and its precedence in resolving authorization conflicts. For instance the scope may either be local to the XML element or recursive and apply to all sub-elements. The default order of precedence is that authorizations defined on an instance of a schema override any authorizations at the schema level. However, the type can be used to modify this behaviour.

The Author- χ framework, developed by Bertino *et al.* follows a broadly similar approach. It defines an ‘access control policy’ to be a tuple (t, s, o, a, p) , where t is a temporal restriction on the use of the authorization and p determines the propagation options for the authorization.

We would argue that these approaches are unlikely to scale well and incorporate all the problems of scalability and administrative burden associated with access control matrices. Moreover, authorizations for XML data have a natural

ordering, which can be used to organize authorizations into a hierarchy.

This paper makes use of ideas from role-based access control. In particular, the fact that XML documents have a hierarchical structure is used to form a hierarchy of permissions, which are grouped to form roles. Our work is also based on the trivial observation that granting access to an XML element and all its sub-elements can be implemented by encrypting that data with a key and supplying an authorized user with that key. Moreover, if we wish to deny access to some of the sub-elements, we can simply further encrypt those elements with a different key. This means that many useful access control policies can be effectively implemented by selectively encrypting parts of an XML document and distributing appropriate decryption keys to authorized users. In our framework a role becomes synonymous with a set of encryption keys.

There exist several schemes for associating keys with elements in a partially ordered set (poset) in such a way that a user with $k(x)$ can derive $k(y)$ for any $y \leq x$ [1, 12, 17, 23].¹ Bertino *et al.* have employed one such scheme to good effect [3]. However, the fact that the hierarchy they use is the powerset of authorizations, which is exponential in the number of authorizations, means that their approach is unlikely to scale well.

Our approach has some similarities to that of Miklau and Suciuc [18], who model an XML document as a tree and specify the keys that are required to ‘unlock’ each element of the tree. However, their approach results in users having to manage several different keys. For large and complex documents, this may prove to be a limiting factor on user acceptance and scalability.

Another limiting factor may be the demands made on the end-user. The approaches described above either require the user to derive a number of different encryption keys or expect the user to manage a number of different keys, and to selectively decrypt different regions of the XML document with each of these keys. We suggest that in order to enforce more complex access control policies and to remove the processing burden from the end-user, it is more appropriate to generate custom views of the XML document and circulate different views to different groups of users. This approach can also be readily adapted to implement a more conventional access control mechanism that receives access requests and either returns the requested information or denies the request.

In the next section, we provide an overview and motivation for our approach. In Section 3 we describe how to use encryption to enforce a class of access control policies for broadcast XML documents and how the Akl-Taylor scheme [1] can be used to minimize the number of keys required by recipients of encrypted documents. We describe the related work and illustrate how our approach complements these frameworks. We also discuss a number of methods for enforcing more complex access control policies using cryptography and explain why we believe such methods are unlikely to be practicable. In Section [4], we describe some

1. We will use the terms ‘poset’ and ‘hierarchy’ interchangeably. Formally, a hierarchy is modelled as the (directed) graph of the reflexive, transitive reduction of the partial order relation. In other words, a hierarchy is modelled as the Hasse diagram [7].

alternative approaches to policy enforcement. We describe how the techniques of Akl and Taylor can be modified to provide a method for assigning a security level to each element in an XML document, which can be used to generate different views for different users. We then illustrate how security levels can be used within the XACML framework to provide a lightweight role-based authorization mechanism for controlling access to XML data. Finally, we conclude with some ideas for future work.

2. AN OVERVIEW

As in classical access control, we define a *protection object* (or simply *object*) to be data for which an access control policy is defined. However, we make one restriction: namely, an object can only be a sub-tree rooted at an element of the document. In fact, this is not as limiting as would first appear, because we may also deny access to any such object, and hence by granting access to an element (and the associated sub-tree) but denying access to other sub-elements, we can provide fine-grained access control. For example, we could grant a user u access to the `journal` element but deny access to the `paper` element. Hence, u would only be able to access elements such as `name` and `table-of-contents`.

2.1 The XPath transform

The XPath transform forms part of the XML digital signature specification [28, Section 6.6.3] and is used to specify those parts of an XML document that are to be signed [27]. For this reason, XPath expressions have often been used to specify regions of a document to which access is restricted [4, 6].

XML documents have a hierarchical structure, determined by the nesting of elements within the document. XPath expressions describe the nodes that satisfy a certain path in this hierarchy. For example, the expression `acm-catalog/journal` selects all journal elements within the `acm-catalog` element. (An expression of this form is similar to an SQL `SELECT` statement.) XPath expressions can also conditionally select particular nodes satisfying a particular path. The expression `acm-catalog/journal[name='TISSEC']`, for example, selects all journal elements in which the `<name>` element is equal to 'TISSEC' (ACM Transactions on Information and System Security). (An expression of this form is similar to an SQL statement of the form `SELECT ... WHERE`.)

An important abbreviation in XPath syntax is `//`, which selects descendant elements at any depth in the document. Hence, `//paper` would select all papers in either ACM journals or proceedings. The wildcard character `*` can also be used: for example, `/acm-catalog/*` selects all child elements of the `catalog` element. The XPath expression `/journal[name='TISSEC']/paper` identifies all nodes corresponding to papers appearing in Transactions on Information and System Security (TISSEC). Damiani *et al* provide an accessible and authoritative introduction to XPath expressions and their use in specifying authorizations [6].

2.2 XML-signature XPath filter 2.0

Unlike previous researchers, we will define objects using the

XPath filter transform [29]. The XML-signature XPath filter specification describes a new signature filter transform that, like the XPath transform, provides a method for computing a portion of a document to be signed.

XPath Filter was developed to improve the efficiency of digital signature implementations. XPath expressions are used to select the roots of document sub-trees, which are then combined using set intersection, subtraction and union. The fact that XPath filters use document sub-trees and set operations makes it ideal for our purposes.

An *input document* contains all the nodes available to processing by the transform. The *filter node set* is computed by evaluating a sequence of XPath expressions and combining their results. The *sub-tree expansion* of a node set is defined to be the set of subtrees rooted at any node in the node set. Initially, the filter node set comprises the entire input document. In sequence, each XPath expression is then evaluated, sub-tree expanded, and then used to transform the filter node set using one of the three set operations. After all XPath expressions have been applied, the resulting node-set is used to filter the input document.

For example, if we wished to specify the object consisting of all journal information except the bodies of papers, we specify the XPath filter

```
<dsig-xpath:XPathFilter="intersect">//journal
  </dsig-xpath:XPath>
<dsig-xpath:XPathFilter="subtract">//journal
  //body</dsig-xpath:XPath>
```

In the remainder of this paper we will use the terms object and XPath filter interchangeably. However, we will generally prefer to use a single letter to denote an object, rather than cluttering the paper with the corresponding XPath filters. Where necessary, we will specify an XPath filter using a combination of XPath expressions and set operations, rather than using the more cumbersome XML syntax. The subtract operation is denoted by `-`. For example, the XPath filter above would be written as

```
//journal - //journal//body
```

2.3 Containment

Wadler provides a semantics for patterns [24], on which the syntax of XPath expressions is based. Part of the semantics identifies the set of nodes in an XML document selected by an XPath expression. Given an XML document D and an XPath filter x , we write $x(D)$ for the set of nodes in D selected by x . Given a schema S and XPath expressions x and y , we will write $x \ll S y$ iff for all documents D that conform to schema S , $x(D) \subseteq y(D)$.

The containment problem, determining whether $x \ll S y$, has been extensively studied in recent years [19,20,26]. In general the containment problem is undecidable, but there are known to be several fragments of XPath for which it is tractable. In particular, the containment problem is in PTIME for XPath expressions that include node tests, `/` and any two of `*` `[]` and `//`. (The containment problem is co-NP complete if we allow all three of `*` `[]` and `//`.) Henceforth, S will be clear from context and we will simply write $x \ll y$ to denote that the set of nodes selected by x is a subset of those selected by y .

2.4 Role-based access control

Role-based access control is beginning to establish itself as the most promising access control paradigm for modern computing systems [10]. The central notion is that of a *role* to which both users and permissions are assigned. The role provides a ‘bridge’ between users and permissions and, since the number of roles is typically orders of magnitude smaller than either the set of users or permissions, the administrative burden is significantly reduced. The central components of role-based access control are: a set of users U , a partially ordered set of roles R , a set of permissions P , a user-role assignment relation $UA \subseteq U \times R$ and a permission-role assignment relation $PA \subseteq P \times R$. An XML access control policy (XACP) will specify a set of objects O and which roles have access to those objects.

Our aim in this paper, is to show how the use of a partial ordering on the set of objects coupled with role-based access control can simplify the expression of access control policies for XML documents. Generally, a permission is an object-access pair. However, we need only consider read access in this paper, so a permission is synonymous with an object.

2.5 Cryptographic access control

Information flow policies have been of interest to the security community since the early 1970s [2,8]. In such policies each user and object is associated with a security level. Since the mid-1980s a number of attempts have been made to implement such policies using cryptography [1,12]. An object is encrypted with the key corresponding to its security level. Each user is given a single key corresponding to his security level. Certain information is made publicly available, which enables the user to derive the key of any security level below his own. We will adapt these techniques to protect XML documents.

We generate a *policy hierarchy* containing objects and roles. Objects are encrypted with keys based on their relative seniority within the policy hierarchy. A user is supplied with a master key, which is synonymous with a role, enabling him to decrypt those objects for which he is authorized. We can control access to published data by encrypting the data and selectively distributing keys. In the context of published XML data, we encrypt different elements of the XML document with different keys.

3. SPECIFICATION AND ENFORCEMENT OF XACPS

Given an XML schema S , we first identify the objects of instances of S that are to be subject to access restrictions. These objects are specified using XPath filters.

Definition 1

Let S be an XML schema. An XML access control policy (XACP) for S is defined to be a tuple (O, R, ρ) , where O is a set of objects specified as XPath filters, R is a set of roles and $\rho \subseteq O \times R$.

The relation ρ specifies the objects to which each role should

have access. We use ρ to induce a partial ordering on the set of roles.

Definition 2

Let $r, r' \in R$ and let $O(r) = \{o \in O : (o, r) \in \rho\}$. We write $r \ll r'$ iff $O(r) \subseteq O(r')$

Definition 3

Let $\mathcal{P} = (O, R, \rho)$ be an XACP. Define

$$\mathcal{O} = \{O(r) : r \in R\} \text{ and} \quad (1)$$

$$\mathcal{O}^i = O^{(i-1)} \cup \{o \cap o' : o, o' \in O^{(i-1)}\} \quad (2)$$

Let j be the least integer such that $\mathcal{O}^j = \mathcal{O}^{(j+1)}$ and define $\mathcal{O}^* = \mathcal{O}^j$. Such a j exists (and is less than the cardinality of O) since \mathcal{O}^i is monotonically increasing, $\mathcal{O}^i \subseteq 2^O$ for all i , and O is finite.

Definition 4

\mathcal{O}^* is the set of policy objects. The policy hierarchy is defined to be the Hasse diagram of the partially ordered set $(\mathcal{O}^*, \subseteq)$.

Note that, the role hierarchy is embedded in the policy hierarchy, since R is equivalent to \mathcal{O} by (1) and $\mathcal{O} \subseteq \mathcal{O}^*$ by (2). We now introduce an example policy for the ACM catalogue, which will make some of these concepts clearer.

3.1 Example policy

We now consider an example for the ACM catalogue. It is assumed that subscribers pay a subscription fee that provides them with a certain level of access to the catalogue. These subscription categories are assumed to be synonymous with roles. There are four distinct classes of subscription: *full* provides access to all parts of the ACM catalogue; *journal* provides access to all parts of all journals in the ACM catalogue; *restricted* provides access to all parts of each journal and proceedings except for the bodies of papers; and *proceedings* provides access to all the contents of ACM proceedings. The table of contents of each journal is freely available.

Figure 2 illustrates an XACP for the ACM catalogue. In the next section we will show how to enforce this policy using cryptographic techniques. Figure 2(c) summarizes the relation ρ . Hence, *restricted* subscribers, for example, would be given access to objects A, B, D and E , but not objects C and F . (Notice that D is not a protected object and does not appear in the relation ρ .)

The role hierarchy contains four roles: *full* is greater than all the other roles and the other three roles are pairwise incomparable. We have

$$\mathcal{O} = \{\{A, B, C, E, F\}, \{A, B, C\}, \{A, B, C\}, \{A, B, E\}, \{A, E, F\}\}$$

Hence,

$$\mathcal{O}^0 = \{\{A, B, C, E, F\}, \{A, B, C\}, \{A, B, C\}, \{A, E, F\}\}$$

$$\mathcal{O}^1 = \mathcal{O}^0 \cup \{\{A, B\}, \{A\}, \{AE\}\}$$

$$\mathcal{O}^2 = \mathcal{O}^1 = \mathcal{O}^*$$

The resulting policy hierarchy is shown in Figure 2(d).

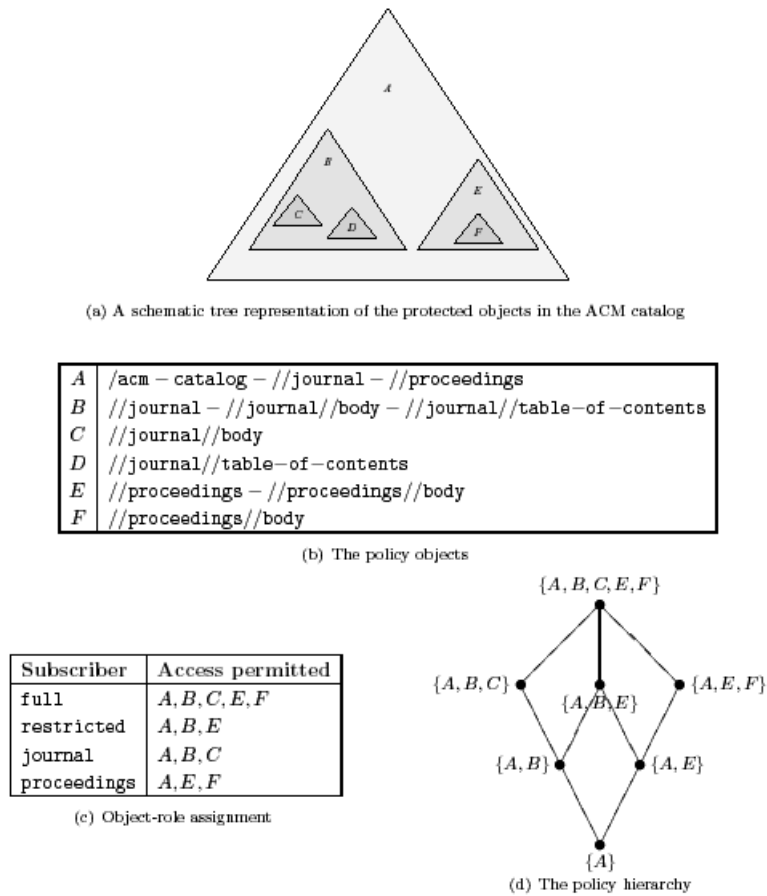


Figure 2 A schematic tree representation of the protected objects in the ACM catalog

3.2 Policy enforcement

What this policy actually means is that full subscribers have recursive read access to the root element of the ACM catalogue, whereas other subscribers do not. Such a policy can be enforced by encrypting more deeply nested elements in the document tree with different keys and distributing appropriate keys to each subscriber. A journal subscriber would not receive the key that is used to encrypt element E, for example.

We implement this policy by encrypting different objects with different keys. We will write $k(x)$ for the key used to encrypt object x . The important question is: How do we choose the encryption keys?

A simple approach to broadcasting this document (while enforcing the XACP) would be to generate six keys $k(A), \dots, k(F)$ and give each recipient the keys required to decrypt those portions of the document he or she is authorized to read. This is the approach adopted by Miklau and Suci (see Section 3.3.2). However, in complex XML documents, this will probably lead to users having to manage a large number of keys. Ideally, we would like to give each recipient a single key that can be used to decrypt appropriate elements in the document. We now describe how this can be achieved by defining a key hierarchy and using techniques due to Akl and Taylor.

3.2.1 Policy and key hierarchies

We will associate an encryption key $k(x)$ with each element

x in the policy hierarchy. If $x \in R$, the key will act as a master key and be used to derive the encryption key of y for all $y \ll x$. Figure 3 shows the key hierarchy for the ACM catalogue policy. (The policy hierarchy has been reproduced for the reader's convenience.) There are four master keys k_f, k_j, k_r and k_p corresponding to the roles full, journal, restricted and proceedings, respectively. Note that if $x \subseteq x'$ we use $k(x')$ to encrypt objects in $x' - x$. Hence, the key corresponding to element $\{A, E\}$, for example, will only be used to encrypt element E , as element A has already been encrypted with $k(A)$. Note also that a master key may be used to encrypt objects, as in the case of objects C and F , for example.

3.2.2 The Akl-Taylor scheme for hierarchical access control

Akl and Taylor developed a method for assigning symmetric encryption keys to a hierarchy of security labels [1, 7]. Their scheme had the property that if a user u was assigned to security label l then he could decrypt any object with security label $l' \ll l$.

The advantage of the key hierarchy is that we can now apply such established methods for generating keys to encrypt an XML document.

Keys $k(A), \dots, k(F)$ will be used to encrypt the document and each user will be given a single key from the set $\{k_f, k_j, k_r, k_p\}$ with which he can either directly decrypt the document or use it to derive appropriate keys.

Let X denote the policy hierarchy. To initialize the

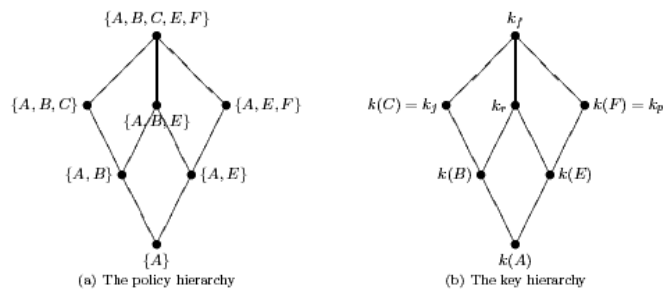


Figure 3 Policy and key hierarchies for the ACM catalogue policy

Akl-Taylor scheme, the document owner performs the following steps:

1. Choose large primes p and q and publish $n = pq$
2. Choose $k \in [2n-1]$, such that $(k, n) = 1$
3. For each $x \in X$, choose a distinct prime $p(x)$
4. For each $x \in X$, define and publish $\pi(x) = \Pi(x)y \leq x \ll p(x)$
5. For each $x \in X$, compute secret key $k(x) = k^{\Pi(x)} \bmod n$.

Figure 4 illustrates how values p and Π are associated with each element of the hierarchy. Hence k_f is defined to be k , k_j is $k_{2,5,7,13}$, etc. Given key $k(x)$ in the hierarchy, it is possible to derive key $k(y)$, where $k(y) \leq k(x)$, by computing

$$k(x)^{\Pi(y)/\Pi(x)} = (k^{\Pi(x)})^{\Pi(y)/\Pi(x)} = (k^{\Pi(y)}) = k(y)$$

Note that $\Pi(y)$ is public information and $\Pi(y)$ is divisible by $\Pi(x)$ whenever $k(y) \leq k(x)$ by construction. Note also that it is not feasible to derive a key $k(z)$, where $k(z) \geq k(x)$ because this would entail computing integral roots of $k \bmod n$ [22]. The scheme is also secure against a set of users pooling information in an attempt to derive keys for which they are not authorized [1].

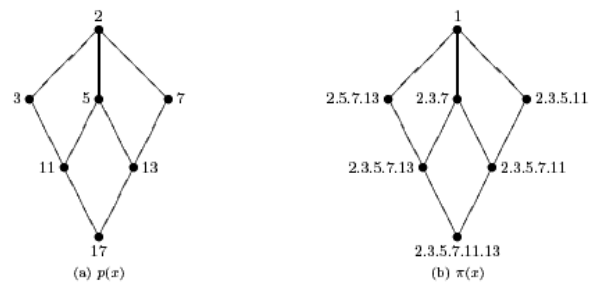
3.3 Related work

In this section we briefly describe two other approaches that use cryptography to limit access to published XML documents. We also illustrate how our methods can be applied to examples from the literature, in one case leading to smaller keys and a simpler implementation of the encryption and in the other reducing the number of keys required by each user to a single one. This suggests that our methods could be combined with the more sophisticated features in the other frameworks to provide a more powerful overall mechanism.

3.1.1 author- \mathcal{X}

author- \mathcal{X} is an access control framework designed for XML documents by Bertino *et al* [4]. A *policy base* is a set of authorizations of the form (t, s, o, a, p) , where s is a subject, o is an object (specified as an XPath expression), a is an access mode, p is a propagation flag and t specifies the time period for which the authorization is valid.

Bertino *et al* describe a distribution strategy for XML data, in which a restricted set of data is ‘pushed’ to subscribers. They describe a method for enforcing a policy base, which encrypts portions of the XML document with different keys in such a way that a subscriber can only decrypt the



Role	Key name	Key value
full	k_f	κ
journal	k_j	$\kappa^{2.5.7.13}$
proceedings	k_p	$\kappa^{2.3.5.11}$
restricted	k_r	$\kappa^{2.3.7}$

(c) Keys

Figure 4 The AKI-Taylor scheme for the key hierarchy in figure 3

information she is permitted to access [3].

The scheme is based on a method of Tzeng, which associates an encryption key with each element in a partially ordered set [23]. Tzeng’s method is based on Harn and Lin’s method for assigning keys to elements in a hierarchy [12], which is itself based on the Akl-Taylor scheme. The main difference between the Harn-Lin and Akl-Taylor schemes is that the former uses asymmetric cryptographic techniques in an effort to reduce the size of the keys associated with minimal elements in the hierarchy.

The main contribution of Tzeng’s scheme is that it can be used to give keys a limited lifetime. Clearly, this is relevant to author- \mathcal{X} because it explicitly includes temporal constraints on authorizations.

Bertino *et al* consider all possible subsets of the set of authorizations and label each node of the XML document with the set of authorizations that applies to the node. Naturally, the powerset of authorizations is a partial order and Tzeng’s scheme can be employed.

However, there are several problems with this approach. Firstly, the powerset of authorizations grows exponentially with the number of authorizations, so the number of nodes in the hierarchy to which Tzeng’s scheme is applied is likely to be very large. This in turn means that it will be expensive to derive encryption keys as the public information associated with each element is the product of primes and may include as many as 2^n primes, where n is the number of authorizations. Secondly, Tzeng’s scheme is not secure against collaborative attacks, in which two or more users combine their key information to deduce keys higher in the hierarchy [31]. Finally, each user receives a key for each authorization that applies to them. This means that users may have to use and manage a large number of keys in order to decrypt a document.

The example used by Bertino *et al* concerns a simple access control policy designed to protect the contents of an electronic newspaper published weekly and distributed in XML format. The newspaper is encrypted and sent to subscribers who are supplied with keys enabling them to decrypt those portions of the newspaper to which they have access. Figure 5 shows the access control policy (as originally presented [3, Table 2]).

The first element of each tuple indicates the temporal

```
((2002,All-days),//Subscriber/type="full",//Newspaper,VIEW,CASCADE)
((2002,Weekend),//Subscriber/type="week-end",//Newspaper,VIEW,CASCADE)
((2002,Wednesday),//Subscriber/type="wednesday",//Financial-supplement,VIEW,CASCADE)
((2002,Sunday),//Subscriber/type="sunday",//Literary-supplement,VIEW,CASCADE)
((2002,All-days),//Subscriber/type="light",//Front-page,VIEW,NO_PROP)
```

Figure 5 An example of Bertino *et al.*

restrictions that are to be applied to the authorization. In each case, the restrictions apply to days of the week in the year 2002. The last two elements of each tuple indicate whether the authorization provides normal read access (NO_PROP) or recursive read access (CASCADE), which also grants access to all sub-elements. In other words, CASCADE grants read access to the subtree.

We assume that users will be associated with an appropriately named role based on their subscription status. Each role will be associated with a key that can be used to decrypt all parts of the newspaper to which that role has access. For simplicity, we assume the existence of a <day> element in the newspaper schema and model temporal constraints as filters of the newspaper content: for example, we model the object accessible to the financial role using the XPath filter `Newspaper[day="wednesday"]//financial-supplement`.

The policy described in Figure 5 is very simple and gives rise to the role hierarchy shown in Figure 6(a). Note that the policy hierarchy differs slightly from the role hierarchy because the weekend and light roles can both access the Front-page element. Hence we include the object `F = //Front-page[day="saturday" or day="sunday"]` in the policy, which can be accessed by both roles. The remaining objects in the policy are uniquely associated with a single role and have been omitted from the policy hierarchy shown in Figure 6(b).

Following the Akl-Taylor scheme, we associate a prime with each element and then generate public information and an encryption key for each element. This is summarized in Figure 6(c), where k is co-prime to $n = pq$ and p and q are large primes. The primes associated with each element in the policy hierarchy are shown in brackets in Figure 6(b).

We encrypt each region of the newspaper with the appropriate key. We distribute the key k to all full subscribers, the key $k^{2.5.7}$ to all weekend subscribers, etc. The complexity of the keys in Figure 6 compares very favourably with the keys derived by Bertino *et al* for this example, where the key exponents were the product of as many as 16 large primes [3, Table 3].²

3.3.2 The work of Miklau and Suciu

Miklau and Suciu developed a comprehensive framework for controlling access to published XML documents [18]. Their framework includes policy specification, policy translation to primitive access control rules, the derivation of a protection scheme from those rules and cryptographic techniques for enforcing the protection scheme. They are able to specify more complex policies than have been considered in this paper, although the enforcement strategy they employ

means that their methods are only likely to be practical for simple XACPs.

We focus here on the enforcement of a protection scheme. A protection scheme P [18, Section 3] is a function from the set of nodes in an XML document tree to a set of guard formulae defined over a set of keys K . A guard formula σ satisfies the following grammar

$$\sigma : \text{true} \mid \text{false} \mid k \mid \sigma \wedge \sigma' \mid \sigma \vee \sigma'$$

where $k \in K$. An example of a schematic XML document and the associated guard formulae [18, Figure 3a] is shown in Figure 7. The necessity formula of an element e is defined to be the conjunction of the guards of all e 's ancestors in the XML tree. The necessity formula of element 4 in the figure, for example, is $k_1 \wedge ((k_1 \wedge k_1) \wedge k_4) \vee k_3$. Hence, any user wishing to access elements in the tree will typically require a number of different keys.

Miklau and Suciu enforce the protection scheme by first

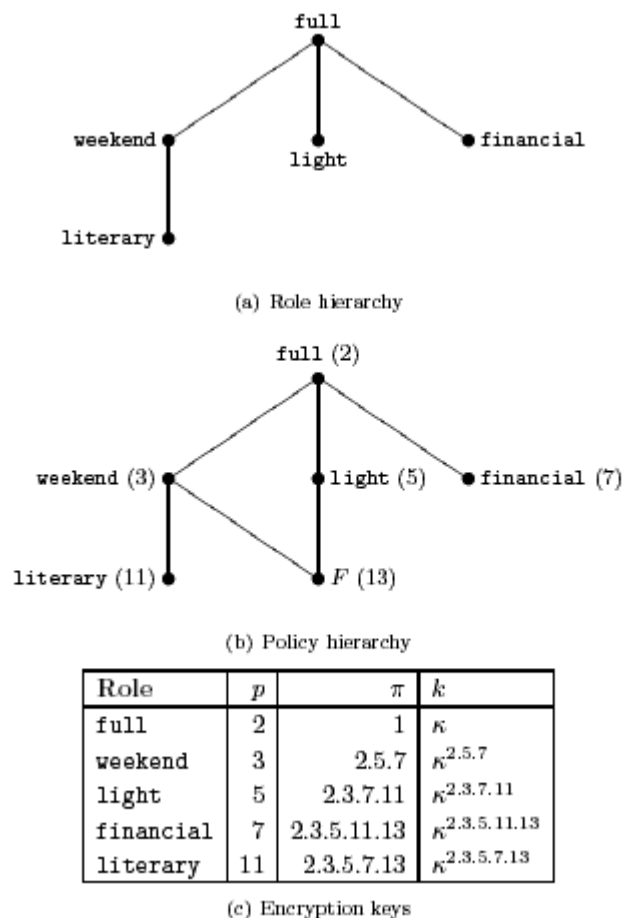


Figure 6 The role and policy hierarchies and encryption keys for the example in Figure 5

2. 'Large' in this context means suitable for use with asymmetric cryptosystems. This is a feature of Tzeng's scheme.

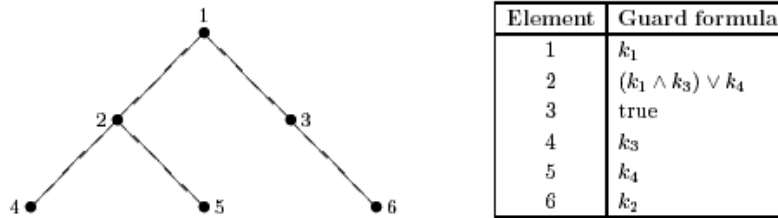


Figure 7 An example of a Miklau-Suciu protection scheme

normalizing the XML document [18, Section 3] so that each element in the new document is associated with a single atomic guard formula. The normalized document is encrypted using a recursive traversal of the normalized document tree, the (unique) key associated with each element and the W3C Recommendation for encrypting XML documents [30]. Hence element 6, for example, would be encrypted with both k_1 and k_2 .³

We suggest the following alternative approach. For each combination of keys $C \subseteq K$, compute the set of nodes that can be accessed using those keys. Intuitively, a node can be accessed using a set of keys C if its necessity formula is satisfied, with the convention that a ‘propositional variable’ k evaluates to true if $k \in C$ and false otherwise. Informally, we convert those sets into abstract roles and associate a new key with each role thus identified. More formally, we have the following definitions.

Definition 5

Let K be a set of keys, $C \subseteq K$, $k \in K$, σ_1 and σ_2 be guard formulae defined over K . Then we write

- $C \models \text{true}$;
- $C \models k$ if $k \in C$;
- $C \models \text{if } C \sigma_1 \wedge \sigma_2 \text{ if } C \models \sigma_1 \text{ and } C \models \sigma_2$
- $C \models \text{if } C \sigma_1 \vee \sigma_2 \text{ if } C \models \sigma_1 \text{ and } C \models \sigma_2$

We say C satisfies if $C \models \sigma$.

Definition 6

Let $C \subseteq K$ and $o \in O$ be a protected object with necessity formula σ_o . Then define $O(C) = \{o \in O : \models \sigma_o\}$.

Let $R = \{O(C) : C \subseteq K\}$. We define an partial order on R by subset inclusion. As before we create a set of policy objects O^* . Figure 8 illustrates the application of these techniques to our example. Figure 8(a) indicates which subsets of $\{k_1, k_2, k_3, k_4\}$ give rise to distinct access capabilities. Note that this table essentially enumerates an instance of the function acc_p defined by Miklau and Suciu [18, Section 3.1]. This function is used in the derivation of a protection scheme from a set of primitive rules [18, Definition 5.1].

We treat each set of keys as a role, and each element as an object. Hence, the role r_2 , corresponding to the set of keys set $\{k_1, k_3\}$ is associated with elements 1, 2, 3 and 4. This enables us to define ρ and hence to infer an ordering on the

set of roles and the set of policy objects. In particular,

$$\begin{aligned} \mathcal{O} &= \{\{1,3\}, \{1,2,3,4\}, \{1,2,3,5\}, \{1,2,3,4,6\}, \{1,2,3,4,5\}, \\ &\quad \{1,2,3,4,5,6\}\} \\ \mathcal{O}' &= \mathcal{O} \cup \{1,2,3\} \\ \mathcal{O}'' &= \mathcal{O}' = \mathcal{O}^* \end{aligned}$$

In Figure 8(b), we illustrate the policy hierarchy. For clarity, we have only included role names and indicated the additional elements that are explicitly assigned to each role in curly brackets. Finally, in Figure 8(c), we illustrate the Akl-Taylor values. The numbers in brackets indicate the primes numbers associated with each element in the hierarchy. These are used to derive the Akl-Taylor public information, also shown in Figure 8(c), and the master key values shown in the last column of Figure 8(a). Note that element 2 would be encrypted using key $\kappa^{2,3,5,7,11}$. This element could not be decrypted by any user assigned to role r_1 , but could be decrypted by any other user.

Encryption of the document could be performed directly using the keys in Figure 8(b) and the same techniques used by Miklau and Suciu. Element 4, for example, would be encrypted first with k'_3 , then with k'_2 and finally with k'_1 . (This nested encryption is the method proposed by Miklau and Suciu. As we observed earlier, this is not necessary, and element 4 could simply be encrypted with key k'_3 .) Note that no normalization is required when applying our method, whereas the normalized document in Miklau and Suciu’s scheme has an additional 6 elements and requires 4 additional keys.

Miklau and Suciu propose a rather rich policy specification language, supporting many of the features available in XACML [21]. However, given the number of keys and additional nodes that are required to enforce the relatively simple policy describe above, it would seem that using their methods to enforce more complex policies will result in users having to manage a very large number of keys. One further problem with the Miklau-Suciu approach is that the necessity formula of an element is the conjunction of the guard formulae on the ancestor-or-self elements. This means, that it is not possible to have a sub-element which is ‘less protected’ than its parent element. An example of why this might be a limitation of their approach is provided by the ACM catalogue example in which the table of contents for each journal is freely available, despite being within the <journal> element, parts of which are encrypted.

3.4 Enforcing more complex XACPS

Consider a new type of ACM subscription sigsac, which allows the user full access to all publications related to secu-

3. Miklau and Suciu assume the use of super-encryption, in which elements may be encrypted several times. An alternative approach, based on the Harn-Lin scheme [12], is to define a function k_i , where the i th element of $h(x)$ is defined to be 1 if k_3 and 0 otherwise.

Role r	Keys C	$O(r)$	Master key k_r
r_1	$\{k_1\}$	$\{1, 3\}$	$\kappa^{2.3.5.7.11.13}$
r_2	$\{k_1, k_3\}$	$\{1, 2, 3, 4\}$	$\kappa^{2.3.5.11}$
r_3	$\{k_1, k_4\}$	$\{1, 2, 3, 5\}$	$\kappa^{2.3.5.7}$
r_4	$\{k_1, k_2\}$	$\{1, 2, 3, 4, 6\}$	$\kappa^{2.5.11}$
r_5	$\{k_1, k_3, k_4\}$	$\{1, 2, 3, 4, 5\}$	$\kappa^{2.3}$
r_6	$\{k_1, k_2, k_3, k_4\}$	$\{1, 2, 3, 4, 5, 6\}$	κ

(a) The access provided by different sets of keys

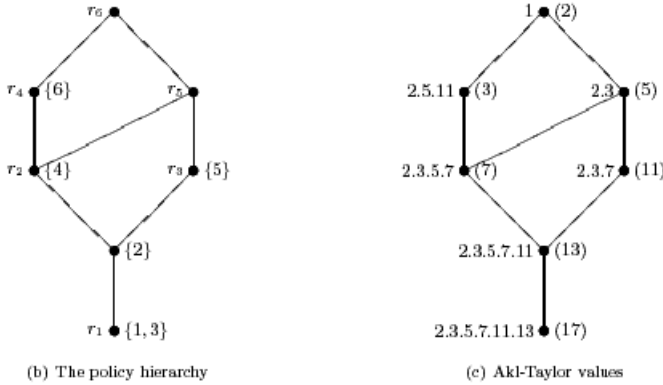


Figure 8 The Miklau-Suciu example using hierarchical and role-based access control

ity (such as TISSEC and the proceedings of CCS and SACMAT).⁴ Introducing this type of subscription, requires a change to the set of protected objects. Specifically, we need to be able to identify security-related content in the catalogue and to distinguish access to that content. The most problematic aspect of this is that while the objects associated with the sigsac role are entirely contained with those associated with the full role, they partially overlap with those objects associated with other roles. Hence, if we encrypt the objects accessible to the sigsac role with some key associated with that role, there will be some information that another authorized role cannot necessarily see. The restricted role, for example, should have access to the summary information for TISSEC but not the full articles, whereas the sigsac role should not have access to the summary information for TODS.⁵ (Of course, there is no problem if we assume that the sigsac role is entitled to have access to all parts of the catalogue granted to the restricted role: that is, $restricted < sigsac$.)

In general, the resolution of such conflicts will be dependent on the wider context. However, in all cases, we must consider the intersection of the overlapping regions and decide on an appropriate encryption strategy. Let o and o' be two overlapping regions and consider the object $o \cap o'$. There are two possibilities: the first is to make the regions disjoint (in other words, remove the intersection from one or other of the regions) and the second is to define the intersection of the two regions as a separate object with its own encryption key.

In the case of our example, consider the following possibilities:

4. SIGSAC stands for the ACM special interest group on security, audit and control. TISSEC stands for ACM Transactions on Information and System Security; CCS for ACM Conference on Computer and Communications Security; and SACMAT for ACM Symposium on Access Control Models and Technologies.
5. TODS stands for ACM Transaction on Database systems

- *Encrypt the summary information for security-related articles using k_r (the key for subscribers to the restricted role)*
In this case, sigsac subscribers would need to be given k_r . This is likely to be the solution that would be adopted in the case of this example: subscription services are often ‘layered’, with a subscriber adding more features (such as special interest groups) to her profile. In this scenario, all users pay for the restricted service and then add subscriptions to special interest groups and other features.

Figure 9 illustrates the policy hierarchy if this implementation is chosen. We split the set of journals (originally object C) into two disjoint objects, C' and C'' : C'' represents the bodies of papers appearing in TISSEC (XPath filter `//journal [name="TISSEC"] //body`), and C' represents all other journals. We make a similar split for the set of proceedings.

- *Encrypt the summary information for security-related articles using k_s (the key for subscribers to the sigsac role)*
In this case, restricted subscribers would have to be given k_s if they are to be able to access summary information for security-related content. It is unlikely that this will be a satisfactory solution in our example.

- *Encrypt the summary information of security-related articles using a new key that can be derived by both restricted and sigsac subscribers*
In this case, restricted subscribers and sigsac subscribers will need to derive an additional key to decrypt summary details for security-related publications. Again, this seems to be inappropriate in our particular example, but this method does provide a general resolution strategy and is the one that should be adopted where the application context means that neither of the above solutions are appropriate. Note that we saw a trivial example of this type of approach in Section 3.3.1, when it was necessary to create a separate object because of overlapping objects for the weekend and light roles.

In general, it would appear that policy hierarchies will become quite complicated in real-world applications, even when the number of roles is quite small. Using cryptography to implement XACPs will be problematic, because the user will either have to manage a large number of keys (as in the approach of Miklau and Suciu) or will have to derive a large number of keys. In the next section we consider how a different use of the Akl-Taylor scheme and the use of role views could provide a solution for complex access control requirements.

4. ALTERNATIVE APPROACHES

In this paper we have examined how to associate keys with different regions of an XML document. This approach was inspired by the work of Bertino *et al* on ‘push’ models for encrypting and distributing XML documents to subscribers. This method was employed in order to enforce an access control policy ‘at a distance’ having disseminated the protected information. We have seen that this approach is not without its problems: complex policies will be difficult to

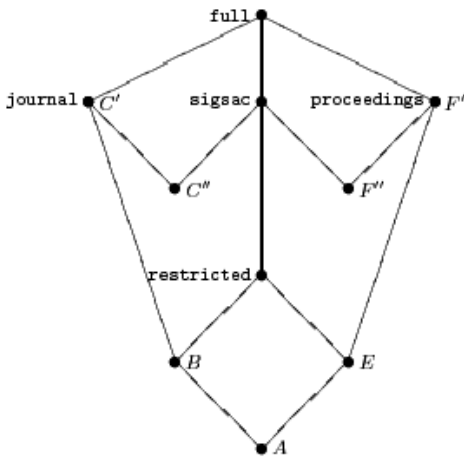


Figure 9 Including the sigsac role in the policy hierarchy

implement and even simple ones require a certain amount of effort from the end user in order to recover the plaintext. In this section we describe some alternative strategies for controlling access to XML data. We first look at the use of views, which are subsets of the original document tree, tailored for the access permitted to each role. The use of views either eliminates or significantly reduces the amount of cryptographic effort required by both publisher and subscriber. Finally, we consider the use of a technique based on the Akl-Taylor scheme that can be used for both push and pull access control models.

4.1 Security views using XSL transformations

Many document formats, such as HTML, contain information that determines both content and presentation. In contrast, XML documents determine only the logical structure of the information within the document and the information itself. Transformations can be applied to an XML document in order to render the information in different formats determined by the requirements of the end user.

An XSL transformation is used to transform an XML document into another XML document or another document format (such as HTML). A transformation can add or remove elements and attributes to or from the output file. XPath is used by the transformation to locate elements and attributes in XML documents. Hence, for each role, we can construct a role filter, an XPath Filter that describes precisely those elements of the XML document the role is authorized to view. The role filter is obtained by taking the union of the objects defined in the policy hierarchy that are less than or equal to the role. Hence, the restricted role filter is

```
/acm-catalog - //journals//body
- //proceedings/paper
```

An XSL transformation can then be written for each role, creating a security view that is distributed to the relevant subscribers. (Of course such views may be encrypted with a single key and digitally signed to prevent eavesdroppers from viewing the content and to provide an integrity check for the view received by the user.) Given that end users are likely to

prefer a plaintext version (or a ‘simple’ encrypted version) of precisely the information they subscribe to, it seems preferable to adopt this approach to distribution. Clearly, the number of views is equal to the number of roles (which will be significantly less than the number of subscribers).

4.2 Access control using Akl-Taylor without encryption

Broadcasting encrypted information to subscribers is not necessarily the only way in which users gain access to XML documents. Damiani *et al.* for example, have developed a fine-grained access control framework for controlling access to stored XML documents [6]. XACML is an XML-based language for specifying and enforcing access control policies for XML data [21]. These access control protection systems are based on the more traditional ‘pull’ model, associated with operating systems and database management systems, in which subjects request access to objects. The request is evaluated by an access control mechanism and the legitimacy of the request is determined by the authorizations that apply to the user. The access request is only granted if the user is appropriately authorized.

We observe that the use of Akl-Taylor public information to derive a view of an XML document can be used equally well for pull distribution models. However, because we are not going to use the Akl-Taylor scheme to generate encryption keys, we are able to use a more efficient representation of these security parameters.

4.2.1 Symbolic Akl-Taylor values

Let X be a poset and let $[x_1, \dots, x_n]$ be an enumeration of the elements of X . any function $f: X \rightarrow Y^n$, let $f_i(x)$ denote the i th component of $f(x)$.

Definition 7

Define $a : X \rightarrow \{0,1\}^n$, where

$$a_i(x) = \begin{cases} 0 & \text{if } x_i \leq x \\ 1 & \text{otherwise} \end{cases}$$

In other words, $a(x)$ is the symbolic prime factorization of the corresponding Akl-Taylor public value (in which the i th element in the enumeration is assigned the i th prime). If $x, y \in X$ and $x \leq y$, then $a(x) \wedge a(y) = a(y)$, where \wedge denotes the bitwise AND operation. Notice that 1, a string of n ones, has the property that $1 \wedge a(x) = a(x)$ and hence that $1 \leq x$, for all $x \in X$.⁶ For $x \in X$, we say that $a(x)$ is the SAT (symbolic Akl-Taylor) value of x .

Figure 10 illustrates the function a for the ACM catalogue policy hierarchy. We assume the following enumeration of the elements in the policy hierarchy:

$$\text{fulljournal} = [C, \text{restricted}, \text{proceedings} = [F, B, E, A]]$$

The policy hierarchy has been reproduced for ease of refer-

6. An alternative approach, based on the Harn-Lin scheme [12], is to define a function $h : X \rightarrow \{0, 1\}^n$, where the i th element of $h(x)$ is defined to be 1 if $x_i \leq x$ and 0 otherwise. If $x, y \in X$ and $x \leq y$, then $h(x) \wedge h(y) = h(x)$. We define $\text{securityLevel} = 1$ for any element that is not protected.

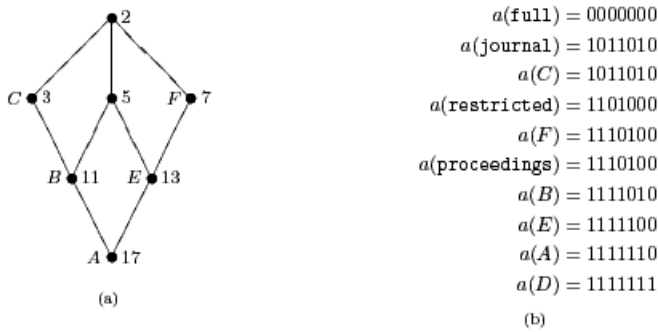


Figure 10 Constructing the function a

ence. The most significant bit corresponds to 2, while the least significant bit corresponds to 17 (the prime assigned to element A in the hierarchy). Notice that $a(D) = 1111111$, indicating that this element is not protected.

4.2.2 The security level attribute

In order to encode an XACML within an XML document, we add a new attribute to each element in the schema. The `securitylevel` attribute is defined to be a binary string of length n , where n is the cardinality of the policy hierarchy.

If the element is a protected object in the hierarchy, the value of its `securitylevel` attribute is defined to be the object's SAT value. Otherwise, it is defined to be 1. Recall that $a(x) \wedge 1 = a(x)$ for all $x \in X$. Hence any object that has `securitylevel = 1` will be 'less than' every other element and will be accessible to every role. Figure 11 illustrates the use of the `securitylevel` attribute in the ACM catalogue policy.

Henceforth we assume that every element in the XML schema has a `securitylevel` attribute.⁷ This attribute is used to encode the set of roles that are permitted to access the element.

4.2.3 Constructing views

When a requester has been identified and authenticated she is associated with a member of the role hierarchy. The access control mechanism will be responsible for identifying the requester and associating her with a SAT value using the policy hierarchy.⁸ A comparison of these attributes will determine whether access is permitted. If access is granted, a view of the document will be created using an XSL transformation parameterized by this attribute.

We can use an XML document annotated with the `securitylevel` attribute to create security views. Given a role r , we construct a 'role view' by transforming the XML document so that an element e is only included if $a(e) \wedge a(r) = a(r)$. This corresponds to 'encrypting' the element in the Akl-Taylor scheme. What it means in practice is that elements the role is not allowed to access are omitted from the transformed document.

4.2.4 Using XACML and Akl-Taylor

We will consider how our techniques can be used in XACML

7. Alternatively, we can generate a new XML document in which each element has such an attribute using an XSL transformation. This approach is less desirable because of the additional processing required.

8. The association of requesters with roles is outside the scope of this work. We anticipate that some kind of attribute-based assignment policy would be used [4, 14, 16], based on the use of digital credentials [25].

policies. Similar considerations apply when adapting our techniques to the methods of Damiani *et al.* [6]. As in the previous section, we assume that the user has been identified and associated with a SAT value.

XACML enables an authorization decision to be based on some characteristic of the subject other than its identity. The most common application of this idea is to use the subject's role. Attributes of subjects contained in the request context may be identified by the `<SubjectAttributeDesignator>` element, which contains a URN that identifies the attribute. Similarly, XACML permits attributes of the target resource (other than its identity) to be used when making an authorization decision. Attributes of the resource may be identified by the `<ResourceAttributeDesignator>` element. Finally, we may define custom functions that take subject and resource attributes as arguments and return a boolean value.

Figure 12 shows a generic XACML policy that can be used to control access to an XML document annotated with the `securitylevel` attribute. Lines 01-07 are preamble identifying relevant namespaces; lines 08-22 define a boolean value that determines whether the `securitylevel` attribute of the subject is greater than that of the requested resource; this variable is referenced in the `<Condition>` element in lines 39-41, which determine whether the rule is applicable and therefore whether access will be permitted. We have assumed the existence of a function that can compare two strings that represent `securitylevel` attributes (line 09) and the definition of the `securitylevel` attribute for subjects (line 12). Lines 10-14 return the `securitylevel` attribute of the subject, while lines 15-20 return the `securitylevel` attribute of the requested resource.

4.3 Related work

There are a number of proposals in the literature for controlling access to XML data [5, 6, 9, 11, 15, 21, 32]. Of particular relevance here are those that generate views of the XML

```
<acm-catalog ... securityLevel="1111110">
  <journal securityLevel="1111010">
    <name securityLevel="1111010">...</name>
    :
    <table-of-contents securityLevel="1111010"> ... </table-of-contents>
    <paper securityLevel="1011010">
      <title securityLevel="1011010">...</title>
      :
    </paper>
    :
  </journal>
  :
  <proceedings securityLevel="1111100">
    <conference securityLevel="1111100">
      <name securityLevel="1111100">...</name>
      :
    </conference>
    <table-of-contents securityLevel="1111100">...</table-of-contents>
    <paper securityLevel="1110100">...</paper>
    :
  </proceedings>
  :
</acm-catalog>
```

Figure 11 Including the `securityLevel` attribute in the ACM catalogue

```

<acm-catalog ... securityLevel="111110">
  <journal securityLevel="1111010">
    <name securityLevel="1111010">...</name>
    :
    <table-of-contents securityLevel="1111010"> ... </table-of-contents>
    <paper securityLevel="1011010">
      <title securityLevel="1011010">...</title>
      :
    </paper>
    :
  </journal>
  :
  <proceedings securityLevel="1111100">
    <conference securityLevel="1111100">
      <name securityLevel="1111100">...</name>
      :
    </conference>
    <table-of-contents securityLevel="1111100">...</table-of-contents>
    <paper securityLevel="1110100">...</paper>
    :
  </proceedings>
  :
</acm-catalog>

```

Figure 11 Including the securityLevel attribute in the ACM catalogue

document [9] or claim to be efficient [5, 32].

Fan *et al* augment a document type definition (DTD) with an *access specification* for each distinct group of users [9].

An access specification for group G is essentially a labelling of the edges in the directed graph representing the DTD, and indicates which parts of the document tree are available to users in G . We note that a user group has many similarities with the concept of role. Unlike our approach, which exploits the structure of the role hierarchy, the approach of Fan *et al* does not take into account any dependencies between user groups.

Yu *et al* introduce the idea of an *accessibility map*, which is essentially a labelling of the nodes of the directed graph representing an XML document [32], and provide relabelling techniques for compressing the accessibility map. Each user requires a compressed accessibility map, and hence this approach is likely to require considerable resources.

Carminati and Ferrari propose an approach designed to make the evaluation of requests to access XML data more efficient [5]. They introduce the concept of an *AC-XML document*, which essentially encodes an access control list for each element in the DTD. Moreover, their approach is based on credentials, where a user is assigned to a role based on his attributes and characteristics. They do not, however, exploit any seniority relationships between roles.

Our approach, although less mature than some of the proposals described above, provides a simple and lightweight access control mechanism for XML data, and one that can easily be incorporated into existing standards such as XACML, unlike other proposals in the literature. It exploits the power of role-based access control, thereby making policy specification and evaluation easier. Like the work of Carminati and Ferrari, our approach can also be applied at the schema level. In short, we reduce the effort required in specifying policies to a minimum.

5. CONCLUSION

We have described a new cryptographic approach for enforcing access control policies on published XML documents. Our approach exploits the inherent hierarchical nature of XML documents and employs role-based ideas to derive keys or views for different users. We believe that our approach offers a scalable and natural approach to access control for XML documents, and also inter-operates well with existing frameworks.

Nevertheless, a considerable amount of research needs to be done. Of immediate interest is a more thorough investigation and formal treatment of the interplay between our framework and that of Miklau and Suciu. We believe that the expressive policy specification and transformation framework of Miklau and Suciu combined with our techniques from role-based and hierarchical access control could prove to be a powerful mechanism for protecting access to published XML documents.

We discussed the problems that arise in enforcing more complex access control policies and proposed an alternative approach using views. These views are generated either using an XPath filter for each role or by annotating an XML document with a *securitylevel* attribute that encodes the relative seniority of each element in the policy hierarchy. This attribute can be compared with a similar attribute for the requesting entity enabling an access decision to be made. Our approach is considerably simpler than existing approaches and fully exploits the hierarchical nature of XML documents, the use of XML schema (or DTDs) and role hierarchies – the first mechanism to do this. However, our work in this area is still at an early stage. We need to undertake additional work to determine the limitations of our approach and whether alternative techniques in the literature support a wider range of access control policies. It would also be interesting to see whether the techniques used by Yu *et al* for compressing the accessibility map can be applied to our approach, thereby reducing storage and maintenance overheads ever further.

```

01 <Policy
02   xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:cd:04"
03   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
04   xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:cd:04
05                       http://docs.oasis-open.org/xacml
06                       /access_control-xacml-2.0-policy-schema-od-04.xsd"
07   PolicyId="urn:isg:rhul:ac:uk:acm-catalog:securityLevelPolicy">
08 <VariableDefinition VariableId="compare:securityLevel:attributes">
09 <Apply FunctionId="urn:isg:rhul:ac:uk:acm-catalog:function:string-compare">
10 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
11 <SubjectAttributeDesignator
12   AttributeId="urn:isg:rhul:ac:uk:acm-catalog:attribute:securityLevel"
13   DataType="http://www.w3.org/2001/XMLSchema#string"/>
14 </Apply>
15 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
16 <AttributeSelector
17   RequestContextPath="//xacml-context:Resource/xacml-context:ResourceContent
18                       //[@securityLevel]/text()"
19   DataType="http://www.w3.org/2001/XMLSchema#string"/>
20 </Apply>
21 </Apply>
22 </VariableDefinition>
23 <Target/>
24 <Rule Effect="Permit">
25 <Target>
26 <AnySubject/>
27 <AnyResource/>
28 <Action>
29 <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
30 <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
31   read
32 </AttributeValue>
33 <ActionAttributeDesignator
34   AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
35   DataType="http://www.w3.org/2001/XMLSchema#string"/>
36 </ActionMatch>
37 </Action>
38 </Target>
39 <Condition>
40 <VariableReference VariableId="compare:securityLevel:attributes">
41 </Condition>
42 </Rule>
43 </Policy>

```

Figure 12 Sample XACML policy

Both XACML and the framework of Damiani *et al.* allow policies in which the authorization information is inconsistent, in the sense that a request may be both granted and denied. In particular, policies that include negative as well as positive authorizations can lead to conflicting decisions.

A particular example occurs when an authorization at the schema level permits access to an element but an authorization at the document level denies access. Such policies require a conflict resolution strategy, such as 'deny-overrides' algorithm defined in the XACML standard. We have not yet addressed these issues. This is certainly something that will concern us in our future work.

ACKNOWLEDGEMENTS

A preliminary version of this paper appeared in the Proceedings of the 2004 ACM Workshop on Secure Web Services.

REFERENCES

- 1 Akl, S., and Taylor, P. Cryptographic solution to a problem of access control in a hierarchy. *ACM Transactions on Computer Systems* 1, 3 (1983), 239–248.
- 2 Bell, D., and LaPadula, L. Secure computer systems: *Mathematical foundations*. Tech. Rep. MTR-2547, Volume I, Mitre Corporation, Bedford, Massachusetts, 1973.
- 3 Bertino, E., Carminati, B. and Ferrari, E. A temporal key management scheme for secure broadcasting of XML documents. In *Proceedings of the 8th ACM Conference on Computer and Communications Security* (2002), pp. 31–40.
- 4 Bertino, E., Castano, S. and Ferrari, E. Author-X: A comprehensive system for securing XML documents. *IEEE Internet Computing* 5, 3 (2001), 21–31.
- 5 Carminati, B. and Ferrari, E. AC-XML documents: Improving the performance of a web access control module. In *Proceedings of 10th ACM Symposium on Access Control Models and Technologies* (2005), pp. 67–76.
- 6 Damiani, E., De Capitani di Vimercati, S., Paraboschi, S. and Samarati, P. A fine-grained access control system for XML documents. *ACM Transactions on Information and System Security* 5, 2 (2002), 169–202.
- 7 Davey, B. and Priestley, H. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, United Kingdom, 1990.
- 8 Denning, D. A lattice model of secure information flow. *Communications of the ACM* 19, 5 (1976), 236–243.
- 9 Fan, W., Chan, C.-Y. and Garofalakis, M. Secure XML querying with security views. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (2004), pp. 587–598.
- 10 Ferraiolo, D., Kuhn, D. and Chandramouli, S. *Role-Based Access Control*. Artech House, Boston, Massachusetts, 2003.
- 11 Gabillon, A. and Bruno, E. Regulating access to XML documents. In *Proceedings of Fifteenth Annual IFIP WG 11.3 Working Conference on Database Security* (2001), pp. 15–18.
- 12 Harn, L. and Lin, H. A cryptographic key generation scheme for multilevel data security. *Computers and Security* 9, 6 (1990), 539–546.
- 13 Harrison, M., Ruzzo, W. and Ullman, J. Protection in operating systems. *Communications of the ACM* 19, 8 (1976), 461–471.
- 14 Herzberg, A., Mass, Y., Mihaeli, J., Naor, D. and Ravid, Y. Access control meets PKI, or: Assigning roles to strangers. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy* (2000), pp. 2–14.
- 15 Kudo, M. and Hada, S. XML document security based on pro-

- visional authorization. In *Proceedings of the 7th ACM Conference on Computer and Communications Security* (2000), pp. 87–96.
- 16 **Li, N., Mitchell, J. and Winsborough, W.** Design of a role-based trust-management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy* (2002), pp. 114–130.
 - 17 **MacKinnon, S., Taylor, P., Meijer, H. and Akl, S.** An optimal algorithm for assigning cryptographic keys to control access in a hierarchy. *IEEE Transactions on Computers* C-34, 9 (1985), 797–802.
 - 18 **Miklau, G. and Suciu, D.** Controlling access to published data using cryptography. In *Proceedings of 29th International Conference on Very Large Data Bases (VLDB 2003)* (2003), pp. 898–909.
 - 19 **Miklau, G. and Suciu, D.** Containment and equivalence for a fragment of XPath. *Journal of the ACM* 51, 1 (2004), 2–45.
 - 20 **Neven, F. and Schwentick, T.** XPath containment in the presence of disjunction, DTDs, and variables. In *Proceedings of 9th International Conference on Database Theory (ICDT 2003)* (2003), pp. 315–329.
 - 21 **OASIS.** eXtensible Access Control Markup Language (XACML) Version 2.0, 2005. OASIS Committee Specification (T. Moses, editor).
 - 22 **Rabin, M.** Digitalized signatures and public-key functions as intractable as factorization. *Tech. Rep. TR-212*, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1979.
 - 23 **Tzeng, W.-G.** A time-bound cryptographic key assignment scheme for access control in a hierarchy. *IEEE Transactions on Knowledge and Data Engineering* 14, 1 (2002), 182–188.
 - 24 **Wadler, P.** A formal semantics of patterns in XSLT and XPath. *Markup Languages: Theory and Practice 2*, 2 (2000), 183–202.
 - 25 **Winslett, M., Ching, N., Jones, V. and Slepchin, I.** Using digital credentials on the world-wide web. *Journal of Computer Security* 5 (1997), 255–267.
 - 26 **Wood, P.** Containment for XPath fragments under DTD constraints. *Proceedings of 9th International Conference on Database Theory (ICDT 2003)* (2003), pp. 300–314.
 - 27 **World Wide Web Consortium.** XML Path Language (XPath) Version 1.0, 1999. J. Clark and S. DeRose (editors).
 - 28 **World Wide Web Consortium.** XML-Signature Syntax and Processing, 2002. W3C Recommendation, D. Eastlake, J. Reagle and D. Solo (authors).
 - 29 **World Wide Web Consortium.** XML-Signature XPath Filter 2.0, 2002. W3C Recommendation, J. Boyer, M. Hughes and J. Reagle (authors).
 - 30 **World Wide Web Consortium.** XML Encryption Syntax and Processing, 2003. W3C Recommendation, D. Eastlake and J. Reagle (editors).
 - 31 **Yi, X. and Ye, Y.** Security of Tzeng’s time-bound key assignment scheme for access control in a hierarchy. *IEEE Transactions on Knowledge and Data Engineering* 15, 4 (2003), 1054–1055.
 - 32 **Yu, T., Srivastava, D., Lakshmanan, L. and Jagadish, H.** A compressed accessibility map for XML. *ACM Transactions on Database Systems* 29, 2 (2004), 363–402.

Appendix A Schema for the ACM catalogue

```

<x:schema>
  <x:element name="acm-catalog">
    <x:complexType>
      <x:attribute name="issue-date" type="xs:string"/>
      <x:attribute name="issue-number" type="xs:string"/>
      <x:element name="journal" maxOccurs="unbounded">
        <x:complexType>
          <x:element name="name" type="xs:string" maxOccurs="1"/>
          <x:element name="date" type="xs:date" maxOccurs="1"/>
          <x:element name="volume" type="xs:string" maxOccurs="1"/>
          <x:element name="number" type="xs:string" maxOccurs="1"/>
          <x:element name="table-of-contents" type="xmtocType" maxOccurs="1"/>
          <x:element name="paper" type="acmpaperType" maxOccurs="unbounded"/>
        <x:/complexType>
      <x:/element>
      <x:element name="proceedings" maxOccurs="unbounded">
        <x:complexType>
          <x:element name="conference" minOccurs="1" maxOccurs="1">
            <x:complexType>
              <x:element name="name" type="xs:string" maxOccurs="1"/>
              <x:element name="year" type="xs:string" maxOccurs="1"/>
              <x:element name="location" type="xs:string" maxOccurs="1"/>
            <x:/complexType>
          <x:/element>
          <x:element name="table-of-contents" type="xmtocType" maxOccurs="1"/>
          <x:element name="paper" type="acmpaperType" maxOccurs="unbounded"/>
        <x:/complexType>
      <x:/element>
    <x:/complexType>
  <x:/element>

  <x:complexType name="xmtocType">
    <x:element name="item" type="xmtocitemType" maxOccurs="unbounded"/>
  <x:/complexType>

  <x:complexType name="xmtocitemType">
    <x:element name="toc-entry" type="xs:string" maxOccurs="1"/>
    <x:element name="page-number" type="xs:string" maxOccurs="1"/>
  <x:/complexType>

  <x:complexType name="acmpaperType">
    <x:element name="title" type="xs:string"/>
    <x:element name="pages" type="xs:string"/>
    <x:element name="author" type="xs:string"/>
    <x:element name="abstract" type="xs:string"/>
    <x:element name="body" type="xs:string"/>
    <x:element name="referencec">
      <x:complexType>
        <x:element name="reference" type="xmreferenceType" maxOccurs="unbounded"/>
      <x:/complexType>
    <x:/element>
  <x:/complexType>

  <x:complexType name="xmreferenceType">
    <x:element name="citekey" type="xs:string"/>
    <x:element name="author" type="xs:string"/>
    <x:element name="title" type="xs:string"/>
    <x:element name="booktitle" type="xs:string"/>
    <x:element name="pages" type="xs:string"/>
    <x:element name="year" type="xs:string"/>
  <x:/complexType>
</x:schema>

```