Applying Hierarchical and Role-Based Access Control to XML Documents

Jason Crampton

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

ABSTRACT

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. The focus of this paper is 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. Finally we consider how role-based access control can be used to enforce more complex access control policies.

Categories and Subject Descriptors

H.2.7 [**Database Management**]: Database Administration—*Security, integrity, and protection*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security, Theory

Keywords

XML, 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 subscribers. 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 at `www.isg.rhul.ac.uk/~jason/Pubs/acm_schema.xml`.) 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 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 [3, 4, 8, 10] and from standards bodies [14]. Most authorization frameworks for controlling access to XML data [3, 4, 8, 14] are essentially extensions of the protection matrix model [7]. 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 [4]. 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 mod-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ACM Workshop on Secure Web Services, October 29, 2004, Fairfax VA, USA.
Copyright 2004 ACM 1-58113-973-X/04/0010 ...$5.00.

```
<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

ify this behaviour.

The Author-$\mathcal{X}$ 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 hierarchy in such a way that a user with $k(x)$ can derive $k(y)$ for any $y \leqslant x$ [1, 6, 9, 16, 17]. Bertino *et al* have employed one such scheme to good effect [2]. 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 Suciu [10], 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. Nevertheless, their approach can be used to implement rather more complicated access control policies than are currently possible within our framework.

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 can be used to minimize the number of keys required by recipients of encrypted documents. Section 4 describes related work [2, 10] and illustrates how our approach complements these frameworks. In the following section, we describe how the techniques of Akl and Taylor can be modified to provide a method for assigning priorities to XML document elements. These priorities can then be used to generate different views for different users. In Section 6, we briefly outline why our approach in Section 3 is unlikely to be suitable for complex access control policies and sketch a method for enforcing such policies. 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 specification [22, Section 6.6.3] and is used to specify those parts of an XML document that are to be signed [20]. For this reason, XPath expressions have often been used to specify regions of a document to which access is restricted [3, 4].

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. For example, the expression /acm-catalog/journal[name="TISSEC"] selects all journal elements containing information about TISSEC. (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 [4].

2.2 XML-Signature XPath Filter 2.0

Unlike previous researchers, we will define objects using the XPath Filter transform [23]. 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 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 XPaths have been applied, the resulting node-set is used to filter the input document.

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

```
<dsig-xpath:XPath Filter="intersect">
    //journal
</dsig-xpath:XPath>
<dsig-xpath:XPath Filter="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 as a pair (op, x), where *op* is a set operation and x is an XPath expression, rather than using the more cumbersome XML syntax. The subtract operation is denoted by $-$.

2.3 Containment

Wadler [18] provides a semantics for patterns, 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 $\mathcal{S}$ and XPath expressions x and y, we will write $x \leqslant_{\mathcal{S}} y$ iff for all documents D that conform to schema $\mathcal{S}$, $x(D) \subseteq y(D)$.

The *containment problem*, determining whether $x \leqslant_{\mathcal{S}} y$, has been extensively studied in recent years [11, 12, 19]. In general the containment problem is undecidable, but there are known to be several fragments of XPath for which it is tractable. Space does not permit a more detailed analysis of these issues. Henceforth, $\mathcal{S}$ will be clear from context and we will simply write $x \leqslant y$ to denote that the set of nodes selected by x is a subset of those selected by y.

2.4 Key-based access control

Role-based access control is beginning to establish itself as the most promising access control paradigm for modern computing systems [5]. 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 how to encrypt them in order to prevent unauthorized access. In a *simple* XACP two objects in O are either disjoint or one is completely contained in the other. A *complex* XACP permits different objects to overlap. In this paper we will focus on simple XACPs and briefly discuss how complex XACPs can be implemented.

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. Objects are ordered using containment of their respective XPath expressions. Objects of an XML document are encrypted with a number of different keys based on the relative seniority of objects. A user is supplied with a master key enabling him to decrypt those objects for which he is authorized. The master key is synonymous with a role. Note that encryption can be used to explicitly deny access to certain objects – if a user doesn't have the relevant key, he cannot access those objects.

3. SIMPLE XACPS

Given an XML schema $\mathcal{S}$, we first identify the objects of instances of $\mathcal{S}$ that are to be subject to access restrictions. These objects are specified using XPath filters and form a partial order under XPath expression containment. If x and q are two XPath filters in a simple XACP, then for any document D that conforms to $\mathcal{S}$, $x(D) \cap q(D) = \emptyset$.

3.1 An example policy

We now consider an example for the ACM catalogue. It is assumed that subscribers pay a subscription fee that provides them with commensurate access to the catalogue. 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. Figure 2 is a schematic representation of the areas of an ACM catalogue that need to be encrypted. Object A represents the whole catalogue; $B \leqslant A$ represents all journals; $C \leqslant B$ represents all journal papers; $D \leqslant B$ represents all tables of contents in journals; $E \leqslant A$ represents all conference proceedings; and $F \leqslant E$ represents all conference papers. Hence, `restricted` subscribers, for example, would be given access to object B, but not objects C and F. Table 1 summarizes the access control policy.

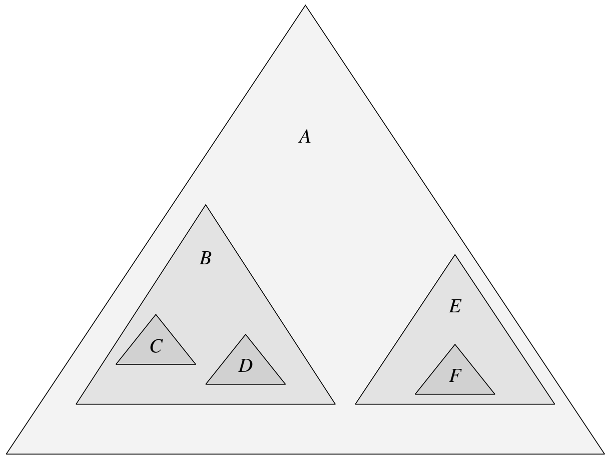


Figure 2: A schematic representation of an encrypted ACM catalogue

Subscriber	Access permitted	Access forbidden
`full`	A, B, C, D, E, F	
`restricted`	A, B, D, E	C, F
`journal`	A, B, C, D	E, F
`proceedings`	A, E, F	B, C, D

Table 1: Objects associated with different subscriber classes

3.2 Policy specification

Let $\langle O, \leqslant \rangle$ denote the partially ordered set of objects. We associate a *height* with each object o, denoted $h(o)$, using the following algorithm, which is essentially a simple modification of a breadth-first search.

- For all minimal elements, set the height to be zero
- For all other elements, set the height to be one greater than the height of its highest children

The height of an object simply measures how deeply it is nested within the set of restricted areas of the document.

Given a set of objects O, an *access control policy* is defined by a function $\delta : O \to \mathbb{Z}$, such that $0 \leqslant \delta(o) \leqslant h(o)$. $\delta(o)$ denotes the depth of encryption of o. Alternatively, we can view an access control policy as a set of policy statements (o, d), where $d \leqslant h(o)$ is the depth of the encryption.

The depth of encryption for an object indicates how many keys will be used to encrypt that object. Clearly the depth of encryption need not exceed the level of nesting of an object. However, it may be that the depth of encryption of an object may be less than that of the object in which it is contained. For example, the ACM may decide to make the table of contents of each of its journals freely available, leading to a policy statement $(D, 0)$. Alternatively, the ACM may wish to provide a free sample copy of TISSEC to all subscribers to the ACM catalogue. In this case, the policy statement will be

```
(//journal[name="TISSEC" and volume="6" and number="3"],1)
```

3.3 Policy enforcement

We can implement this policy by first encrypting elements C, D and F with keys $k(C)$, $k(D)$ and $k(F)$, respectively. We then encrypt elements B and E (including sub-elements C, D and F) with keys $k(B)$ and $k(E)$, respectively. Finally, we encrypt the whole catalogue with $k(A)$. Users with a `restricted` subscription to the catalogue service will be given keys $k(A)$, $k(B)$, $k(D)$ and $k(E)$, which enables them to recover information about abstracts, authors, etc., but not to decrypt the body of any paper. Conversely, full subscribers to the service will be given all the keys. The important question is: How do we choose the encryption keys?

A naïve approach to broadcasting this document (while enforcing the XACP) would be to generate six keys $k(A), \ldots, k(F)$ and give each recipient the keys required to decrypt those portions of the document he or she is authorized to read. However, in complex XML documents, this will likely 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 the concept of a master key.

Given a set of policy statements $\mathcal{P}$, we define a partial ordering on $\mathcal{P}' \subseteq \mathcal{P}$ in the following way. We first omit any policy statements of the form (o, d) from $\mathcal{P}$ if there exists another policy statement of the form $(o', d') \in \mathcal{P}$ with $o \leqslant o'$ and $d \leqslant d'$.[1] Let $(o, d), (o', d') \in \mathcal{P}'$. Then $(o, d) \leqslant (o', d')$ iff $d \leqslant d'$ and $o \leqslant o'$.

In our example, we define

$$\delta(A) = 1, \delta(B) = \delta(E) = 2, \delta(C) = \delta(D) = \delta(F) = 3.$$

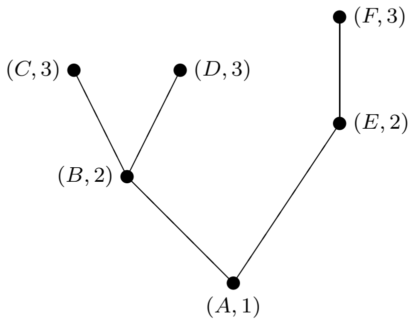
(Note that in this case $\delta(o) = h(o)$ for all o.)

3.3.1 Key hierarchies

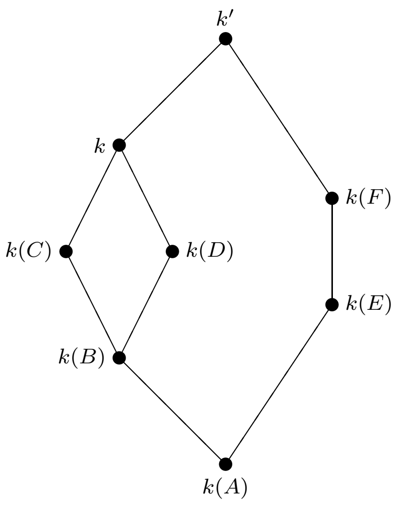
Let $k(o)$ denote the key associated with a given policy statement (o, d) in the hierarchy. Moreover, we can define master keys in the policy hierarchy. In other words, if we have a set of keys $\{k_1, \ldots, k_n\}$, we can create a master key k in the hierarchy by specifying that $k > k_i$, $1 \leqslant i \leqslant n$. A master key k has the property that it can be used to derive every subordinate key in the hierarchy. Essentially a master key is a role, granting access to a set of permissions.

[1]The rationale is that since o is contained in o' and the depth of encryption is the same, it can be ignored for the purposes of policy enforcement. This will become clearer when we explain how a policy is enforced.

Figure 3 shows the policy hierarchy and key hierarchy derived from the encrypted document shown in Figure 2. We have included two master keys k and k': k will be used to derive keys $k(A)$, $k(B)$ $k(C)$ and $k(D)$, while k' can be used to derive all other keys. The intuition is that we will give key k' to `full` subscribers to the ACM catalogue, k to `journal` subscribers and other keys to less privileged subscribers. (For clarity, we have not included a key for `restricted` subscribers in the figure, which would be a master key greater than both $k(D)$ and $k(E)$, giving access to objects A, B, D and E.)



(a) Policy hierarchy



(b) Key hierarchy

Figure 3: Policy and key hierarchies for the encrypted ACM catalogue in Figure 2; k and k' are master keys

3.3.2 *Hierarchical access control*

Akl and Taylor developed a method for assigning symmetric encryption keys to a hierarchy of security labels [1, 9]. 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 $l' \leqslant l$.

The advantage of the key hierarchy is that we can now apply such established methods for generating keys to encrypt an XML document. Keys $k(A), \ldots, k(F)$ will be used to encrypt the document and each user will be given a single key from the set $\{k(A), \ldots, k_(F), k, k'\}$ with which he can either directly decrypt the document or use it to derive appropriate keys.

Let X denote the key hierarchy. To initialize the scheme, the document owner performs the following steps:

(1) Choose large primes p and q and publish $n = pq$

(2) Choose $\kappa \in [2, n-1]$ such that $(\kappa, n) = 1$

(3) For each $x \in X$, choose a distinct prime $p(x)$

(4) For each $x \in X$, define and publish $\pi(x) = \prod_{y \not\leqslant x} p(y)$

(5) For each $x \in X$, compute secret key $k(x) = \kappa^{\pi(x)} \mod n$

Figure 4 illustrates how values p and π are associated with each element of the hierarchy. Hence k' is defined to be κ, k is $\kappa^{2.11.17}$, etc.

Given key $k(x)$ in the hierarchy, it is possible to derive key $k(y)$, where $k(y) \leqslant k(x)$, by computing

$$k(x)^{\pi(y)/\pi(x)} = (\kappa^{\pi(x)})^{\pi(y)/\pi(x)} = \kappa^{\pi(y)} = k(y).$$

Note that $\pi(y)$ is public information and $\pi(y)$ is divisible by $\pi(x)$ whenever $k(y) \leqslant k(x)$ by construction. Note also that it is not feasible to derive a key $k(z)$, where $k(z) \geqslant k(x)$, because this would entail computing integral roots of $\kappa \mod n$ [15]. 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].

4. RELATED WORK

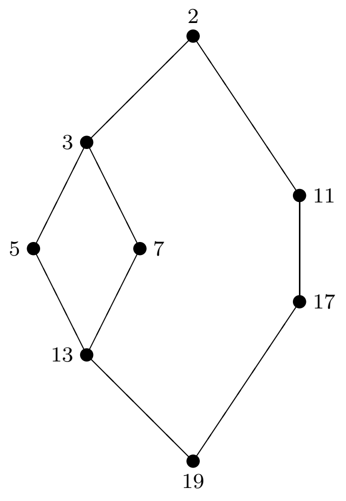
In this section we briefly describe work on two other frameworks that use cryptography to limit access to published XML documents. We also illustrate how our methods can be applied to examples from the literature, in one case leading to smaller keys and a simpler implementation of the encryption and in the other reducing the number of keys required to 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.

4.1 Author-X

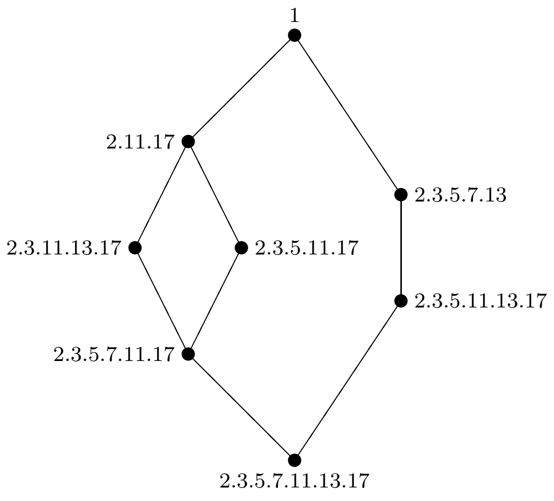
Author-$\mathcal{X}$ is an access control framework designed for XML documents by Bertino *et al* [3]. A *policy base* is a set of authorizations of the form (t, s, o, a, p), where s is a subject, o is an object (specified as an XPath expression), a is an access mode, p is a propagation flag and t specifies 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 information she is permitted to access [2].

The scheme is based on a method of Tzeng, which associates an encryption key with each element in a partially ordered set [17]. Tzeng's method is based on Harn and Lin's method for assigning keys to elements in a hierarchy [6], which is itself based on the Akl-Taylor scheme. The main difference between the Harn-Lin and Akl-Taylor schemes is



(a) $p(x)$



(b) $\pi(x)$

Figure 4: Initializing the Akl-Taylor scheme for the key hierarchy in Figure 3

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 is 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 [25]. 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 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(a) shows the access control policy.

The first element of each tuple indicates the temporal 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`). Note that `CASCADE` is the default behaviour in our approach.

We assume that users will be supplied with keys based on their subscription status. We modify our policy hierarchy so that the temporal constraints are modelled as filters of the newspaper content. In practice, these filters can be represented using XPath expressions: (`wednesdays`, $\cap$, `/Newspaper`), for example, can be modelled using the expression ($\cap$, `/Newspaper[@day="wednesday"]`) (assuming that we modify the schema so that the `Newspaper` element has a `day` attribute). In fact this is a very simple policy and gives rise to the policy hierarchy shown in Figure 5(b).

o	p	π	k
(`*`, `/Newspaper//*`)	2	1	κ
(`weekends`, `/Newspaper//*`)	3	2.5.7	$\kappa^{2.5.7}$
(`*`, `//Front-page`)	5	2.3.5.11	$\kappa^{2.3.7.11}$
(`wednesdays`, `//Fin-supp//*`)	7	2.3.7.11	$\kappa^{2.3.5.11.13}$
(`sundays`, `//Lit-supp//*`)	11	2.3.5.7	$\kappa^{2.3.5.7.13}$
(`weekends`, `//Front-page`)	13	2.3.5.7.11	$\kappa^{2.3.5.7.11}$

Table 2: Assigning encryption keys to the hierarchy in Figure 5

We encrypt portions of the newspaper with the appropriate key. We then distribute the key κ to all full subscribers, the key $\kappa^{2.5.7}$ to all weekend subscribers, etc. The complexity of the keys in Table 2 compares very favourably with the keys derived by Bertino *et al* in their scheme, in which the exponent was the product of as many as 16 large primes [2, Table 3].[2]

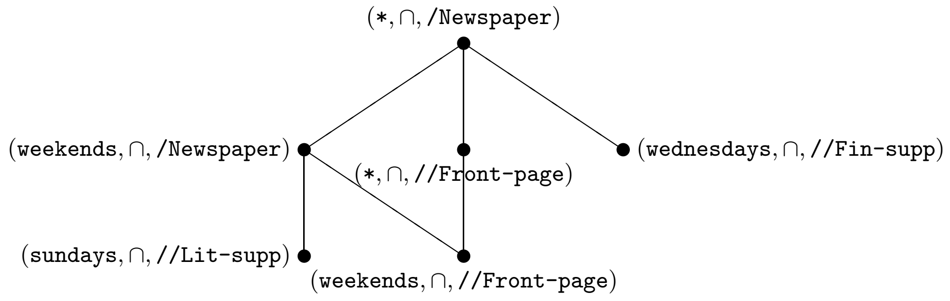
4.2 The work of Miklau and Suciu

Miklau and Suciu developed a comprehensive framework

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

((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`)

(a) Policy base


(*, ∩, /Newspaper)
(weekends, ∩, /Newspaper)
(*, ∩, //Front-page)
(wednesdays, ∩, //Fin-supp)
(sundays, ∩, //Lit-supp)
(weekends, ∩, //Front-page)

(b) Policy hierarchy

Figure 5: The policy base and policy hierarchy for the newspaper example

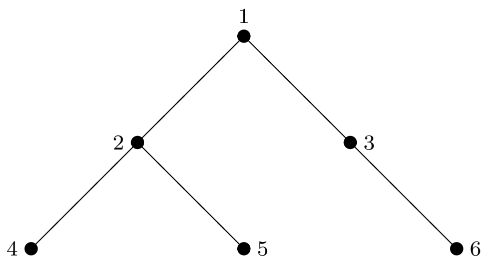
for controlling access to published XML documents [10]. Their framework includes policy specification, policy translation to primitive access control rules, the derivation of a protection scheme from those rules and cryptographic techniques for enforcing the protection scheme. They are able to specify more complex policies than have been considered in this work.

We focus here on the enforcement of a protection scheme. A *protection scheme* P [10, Section 3] is a function from the set of nodes in an XML document 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 [10, Figure 3a] is shown in Figure 6. 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 $k_1 \wedge ((k_1 \wedge k_3) \vee k_4) \wedge k_3$. Miklau and Suciu enforce the protection scheme by first *normalizing* the XML document [10, Section 3.2] 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 [24].

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. Space precludes a formal treatment of this approach; instead we illustrate the techniques involved by re-working the example in Figure 6.



Element	**Guard formula**
1	k_1
2	$(k_1 \wedge k_3) \vee k_4$
3	true
4	k_3
5	k_4
6	k_2

Figure 6: An example of a Miklau-Suciu protection scheme

Figure 7(a) indicates which subsets of $\{k_1, k_2, k_3, k_4\}$ give rise to distinct access capabilities. For example, the set of keys $\{k_1, k_3\}$ enables a user to "unlock" elements 1, 2, 3 and 4. In contrast, the key k_2 on its own, does not permit any access to the document. Note that this table essentially enumerates an instance of the function acc_P defined by Miklau and Suciu [10, Section 3.1]. This function is used in the derivation of a protection scheme from a set of primitive rules [10, Definition 5.1].

We treat these sets of keys as roles, defining new roles where necessary, and define a new key for each role. Figure 7(b) illustrates how this is done. Notice that r_2 provides access to elements $\{1, 2, 3\}$, which is not a set of elements that appears in Figure 7(a). It is unlikely that anyone will be given the key k'_2, but it is necessary to encrypt element 2 with this key in order to enforce the protection scheme.

Finally, in Figure 7(c), we illustrate the role hierarchy. Roles are ordered by set inclusion on the associated sets of elements. Hence $r_1 \leqslant r_2$, for example, because $\{1, 3\} \subseteq \{1, 2, 3\}$. The numbers in brackets indicate the primes numbers associated with each element in the hierarchy. These are used to derive the Akl-Taylor key values shown in Figure 7(b).

Encryption of the document can be performed directly using the keys in Figure 7(b) and the same techniques used by Miklau and Suciu. Element 4, for example, is encrypted first with k'_3, then with k'_2 and finally with k'_1. Note that no normalization is required. We also note that the normalized document has an additional 6 elements and requires 4 additional keys.

5. VIEWS USING AKL-TAYLOR

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 distributing XML documents to subscribers. We note that there is an alternative way of implementing this distribution model that is likely to be far simpler for end users as it relies on little or no encryption.

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. Therefore, having assigned a public value to each of the nodes in the permission hierarchy, instead of generating encryption keys, we generate a *view* of the XML document using an XSLT transformation [21].

The new document is identical to the original, except that every element in the transformed document includes a priority attribute. This attribute is assigned the value of the public information associated with that element if it appears in the permission hierarchy (and 0 otherwise). Indeed, we can modify the underlying XML schema so that the every element has a required `priority` attribute, which, by default is 0.

For each value i in the set of public information, an XSLT transform [21] parameterized by i can then generate separate views of the "prioritized" document, only including an element in the view if the value of the priority attribute of the element is divisible by i. (Note that 0 is divisible by i for all i. Hence priority 0 elements, which are not subject to any access restrictions, are included in every view.) These views are then distributed to the appropriate subscribers. Of course such views may be encrypted and digitally signed to prevent eavesdroppers from viewing the content and to ensure the integrity of the view received by the user.

We note that this implies that the number of views is equal to the number of keys, which in turn equals the number of nodes in the permission hierarchy. Given that end users are likely to prefer a plaintext version of precisely the information they subscribe to, it seems preferable to adopt this approach to distribution.

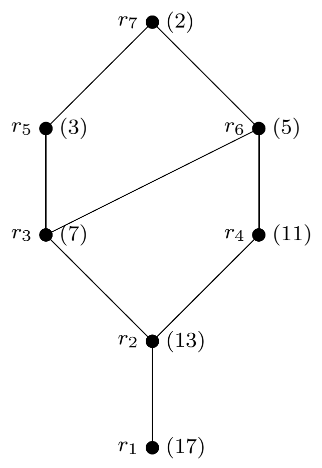
Broadcasting encrypted information to subscribers is not necessarily the only way in which users need to access XML documents. Damiani *et al* have developed a fine-grained access control framework for controlling access to stored XML documents [4]. This framework is based on the more traditional "pull" model, associated with operating systems and database management systems, in which subjects request ac-

Keys	Elements
$\{k_1\}$	$\{1, 3\}$
$\{k_1, k_3\}$	$\{1, 2, 3, 4\}$
$\{k_1, k_4\}$	$\{1, 2, 3, 5\}$
$\{k_1, k_2\}$	$\{1, 2, 3, 4, 6\}$
$\{k_1, k_3, k_4\}$	$\{1, 2, 3, 4, 5\}$
$\{k_1, k_2, k_3, k_4\}$	$\{1, 2, 3, 4, 5, 6\}$

(a) The access provided by different sets of keys

Role	Elements	Key
r_1	$\{1, 3\}$	$k'_1 = \kappa^{2.3.5.7.11.13}$
r_2	$\{1, 2, 3\}$	$k'_2 = \kappa^{2.3.5.7.11}$
r_3	$\{1, 2, 3, 4\}$	$k'_3 = \kappa^{2.3.5.11}$
r_4	$\{1, 2, 3, 5\}$	$k'_4 = \kappa^{2.3.5.7}$
r_5	$\{1, 2, 3, 4, 6\}$	$k'_5 = \kappa^{2.5.11}$
r_6	$\{1, 2, 3, 4, 5\}$	$k'_6 = \kappa^{2.3}$
r_7	$\{1, 2, 3, 4, 5, 6\}$	$k'_7 = \kappa$

(b) A hierarchical set of keys



(c) The role hierarchy

Figure 7: The Miklau-Suciu example using hierarchical and role-based access control

cess 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 mechanism returns a view of the XML document corresponding to those regions of the document that the user is entitled to view.

We observe that this alternative use of Akl-Taylor public information to derive a view of an XML document can be used equally well for pull distribution models. When a requester has been identified and authenticated he can be associated with a member of the policy hierarchy. The public information of this policy is then used as input to an XSLT transformation to derive an appropriate view to be returned to the requester. We believe that this approach is likely to scale better than the approach of Damiani *et al* because it is based on role-based concepts rather than the protection matrix model.

6. COMPLEX XACPS

6.1 Using encryption

We now consider the case where $o, o' \in O$ such that o and o' partially overlap. Again, policies of this sort occur very frequently. Consider a new type of subscription `sigsac`, which allows the user full access to all publications related to security (such as TISSEC and the proceedings of CCS and SACMAT).[3] While $o(\texttt{sigsac})$, the regions associated with the `sigsac` subscription, are contained within $o(\texttt{full})$, $o(\texttt{sigsac})$ partially overlaps $o(\texttt{restricted})$. In particular, if we encrypt $o(\texttt{sigsac})$ with some key, then `restricted` subscribers will not be able to view those parts of TISSEC that they are authorized to view.

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 what the depth of encryption should be for that intersection. 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 depth. In the case of our example, consider the following possibilities:

- $\delta(o(\texttt{sigsac}) \cap o(\texttt{restricted})) = \delta(o(\texttt{restricted}))$

  In this case, we have removed part of the `sigsac` object and `sigsac` subscribers would need to be given the key for `restricted` subscribers. This is likely to be the solution that would be adopted in the case of this example. Subscription services are usually "layered", with subscribers adding more features to their profile. In this scenario, all users pay for the restricted service and then add subscriptions to special interest groups and other additional features.

- $\delta(o(\texttt{sigsac}) \cap o(\texttt{restricted}) = \delta(o(\texttt{sigsac}))$

  In this case, we have removed part of the `restricted` object and assuming that `restricted` subscribers will not be given the key to decrypt all SIGSAC publications, `restricted` subscribers will no longer be able to access summary details for journals such as TISSEC. It is unlikely that this will be a satisfactory solution in our example.

- $\delta(o(\texttt{sigsac}) \cap o(\texttt{restricted}) > \max\{\delta(o(\texttt{sigsac})), \delta(o(\texttt{restricted}))\}$

  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 either of the above solutions are not appropriate. Note that we saw a trivial example of this type of approach in Section 4.1, when it was necessary to create a separate object (`weekends`, `//Front-page`).

In general, it would appear that using cryptography to implement complex access control policies will be problematic, although resolving some of these problems lies in careful policy specification and policy design. In the next section we consider how a direct application of role-based access control could provide a solution for complex access control requirements.

6.2 Using a role-based approach and views

For more complex policies, we believe that role-based access control can provide a solution. Instead of encrypting documents, we will associate regions of the document with roles and generate different views of the document for different roles.

We define a permission to be an XPath filter. We associate each permission with a role and assign users to roles based on identification, authentication and credential discovery (using SAML assertions, for example [13]). The `sigsac` role could be assigned the permissions

$$(\cap, \texttt{//journal[name="TISSEC"]}),$$
$$(\cup, \texttt{//proceedings[name="CCS"]}),$$
$$(\cup, \texttt{//proceedings[name="SACMAT"]}),$$

whereas the `restricted` role is assigned the permissions

$$(\cap, \texttt{//journal}),$$
$$(-, \texttt{//journal//body}),$$
$$(\cup, \texttt{//proceedings}),$$
$$(-, \texttt{//proceedings//body}).$$

Views of the catalogue are generated for each role. One simple way of implementing this scheme is to encrypt the region with a key associated with the role and to encrypt the remainder of the document with some other key. Space does not permit a detailed examination of this approach, which will be the subject of our future work.

7. CONCLUDING REMARKS

We have described a new approach to access control policies for 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 which also inter-operates well with existing frameworks.

[3]SIGSAC is the ACM special interest group on security, audit and control.

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 must also address some of the more complex policies envisaged by researchers. For example, we need to investigate how to implement policies in which objects overlap. We believe that encrypting documents in such situations will lead to a proliferation of keys. We anticipate that the technique of assigning a priority or role identifier to each element of the document (as described in Section 6) is likely to prove more successful.

Damiani *et al*, amongst others, have considered policies in which authorization information conflicts. We have not yet addressed these types of policies. We believe that their approach of labelling the document prior to generating a user view could be merged with our technique for assigning priorities to produce a comprehensive, scalable access control mechanism with conflict resolution capabilities.

Finally, I would like to thank the anonymous referees for their helpful comments and particularly for drawing my attention to the work of Miklau and Suciu.

8. 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] 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.

[3] Bertino, E., Castano, S., and Ferrari, E. Author-$\mathcal{X}$: A comprehensive system for securing XML documents. *IEEE Internet Computing 5*, 3 (2001), 21–31.

[4] 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.

[5] Ferraiolo, D., Kuhn, D., and Chandramouli, S. *Role-Based Access Control*. Artech House, Boston, Massachussetts, 2003.

[6] Harn, L., and Lin, H. A cryptographic key generation scheme for multilevel data security. *Computers and Security 9*, 6 (1990), 539–546.

[7] Harrison, M., Ruzzo, W., and Ullman, J. Protection in operating systems. *Communications of the ACM 19*, 8 (1976), 461–471.

[8] Kudo, M., and Hada, S. XML document security based on provisional authorization. In *Proceedings of the 7th ACM conference on Computer and communications security* (2000), pp. 87–96.

[9] 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.

[10] 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.

[11] Miklau, G., and Suciu, D. Containment and equivalence for a fragment of XPath. *Journal of the ACM 51*, 1 (2004), 2–45.

[12] 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.

[13] OASIS. *Assertions and Protocols for the OASIS Security Assertion Markup Language*, 2003. OASIS Committee Specification: E. Maler, P. Mishra and R. Philpott (editors).

[14] OASIS. *eXtensible Access Control Markup Language (XACML) Version 1.1*, 2003. OASIS Committee Specification: S. Godik and T. Moses (editors).

[15] 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.

[16] Ray, I., Ray, I., and Narasimhamurthi, N. A cryptographic solution to implement access control in a hierarchy and more. In *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies* (2002), pp. 65–73.

[17] 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.

[18] Wadler, P. A formal semantics of patterns in XSLT and XPath. *Markup Languages: Theory and Practice 2*, 2 (2000), 183–202.

[19] Wood, P. Containment for XPath fragments under DTD constraints. In *Proceedings of 9th International Conference on Database Theory (ICDT 2003)* (2003), pp. 300–314.

[20] World Wide Web Consortium. *XML Path Language (XPath) Version 1.0*, 1999. J. Clark and S. DeRose (editors).

[21] World Wide Web Consortium. *XSL Transformations (XSLT) Version 1.0*, 1999. J. Clark (editor).

[22] World Wide Web Consortium. *XML-Signature Syntax and Processing*, 2002. W3C Recommendation, D. Eastlake, J. Reagle and D. Solo (authors).

[23] World Wide Web Consortium. *XML-Signature XPath Filter 2.0*, 2002. W3C Recommendation, J. Boyer, M. Hughes and J. Reagle (authors).

[24] World Wide Web Consortium. *XML Encryption Syntax and Processing*, 2003. W3C Recommendation, D. Eastlake and J. Reagle (editors).

[25] 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.