Chapter 10

TOWARDS SECURE XML FEDERATIONS

Lingyu Wang, Duminda Wijesekera and Sushil Jajodia

Abstract The integration of isolated XML repositories has drawn more and more interest recently. In this paper, we propose *XML federations* to provide global e-services while preserving the necessary autonomy and security of each individual repository. First we show a logical architecture of XML federations, which is adapted from the common architecture of traditional federated databases according to the unique requirements of XML federations. On the basis of the architecture, we address security issues of XML federations, especially the authentication and authorization of federation users. We point out several problems in applying existing access control schemes and give our solutions.

Keywords: XML, federated database system, information integration, database security

1. Introduction

Recently, the continuing demand for information sharing has shifted interest from stand-alone XML repositories to interconnected and large-scale cooperative XML systems [8, 10]. The seamless and secure integration of isolated XML repositories presents new research challenges in many respects. In this paper, we adapt the mature techniques developed in traditional *Federated Database Systems (FDBSs)* for this purpose. FDBS has been recognized as a practical approach to integrate traditional databases while retaining the autonomy and security of each participant. An FDBS is a collection of cooperative but autonomous component database systems integrated to various degrees [17]. An *XML federation* can be regarded as a special FDBS composed of several stand-alone *component XML repositories* (or simply *components*) and a *federation* as the common interface to provide global e-services.

The coexistence of cooperation and autonomy in XML federations makes them suitable for use in extensive real-world scenarios, ranging from civilian and military coalitions to non-governmental conglomerates and from e-business providers to e-government organizations. XML federations have ad-

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35697-6_26

E. Gudes et al. (eds.), *Research Directions in Data and Applications Security*

vantages over either a collection of interoperable XML systems with no integration or a centralized XML system leaving no autonomy for each participant. For example, an XML federation formed by travel agencies, hotels, restaurants, and car rental companies provides a user with convenient all-in-one traveling e-services. In the case of interoperable XML systems, users will have to interact with each company respectively and coordinate the outcomes by themselves. On the other hand, it is unrealistic for hotels, restaurants, or car rental companies to unconditionally hand over their own business to serve travelers only.

XML federations bring many unique research challenges, among which we specially address logical architecture and access control issues. Logical architecture forms the foundation for further studies of XML federations, including the study of security-related issues. We describe an architecture based on existing practices in traditional FDBSs and the unique requirements of XML federations. We then investigate several access control schemes proposed for traditional FDBSs. We show some new issues in applying them to XML federations and give our solutions. Among issues we address are:

- When a component authenticates a federation user, how can the authentication be reliably done with all of the communications going through the untrusted federation?
- How can the federation achieve fine-grained access control with only the knowledge of the schema and how can components authorize federation users without knowing them?
- When the federation switches user identities in accessing components, how can components be convinced about what really happens if the federation can collude with federation users to gain unauthorized accesses?

The rest of the paper is organized as follows. Section 2 reviews basic concepts and related works in traditional FDBSs and access control in isolated XML repositories. Section 3 proposes a logical architecture for XML federations. Section 4 addresses the authentication and access control issues of XML federations. Finally Section 5 concludes the paper.

2. Basic Concepts and Related Work

2.1 Federated Database System

An FDBS is a collection of cooperating yet autonomous component database systems[17]. Component databases are heterogeneous in many aspects, such as data models, query languages, and access control policies. Moreover, *semantic heterogeneity* arises because the same or similar data items may have different interpretations or intended uses among component databases. According to the degree of integration, two classes of FDBSs are defined in [17],

namely, *loosely coupled FDBSs* and *tightly coupled FDBSs*. A loosely coupled FDBS is rather like a collection of inter-operable database systems. A tightly coupled FDBS creates the federation at design time and actively controls the accesses through the federation. In the rest of this paper, we assume a tightly coupled FDBS and refer to it as an FDBS unless stated otherwise.

Access control in an FDBS is more complicated than in centralized databases due to *authorization autonomy*, which allows component databases to have control over their shared data [11, 13–15, 20]. The degree of such control divides access control in an FDBS into three classes. For *full authorization autonomy*, component databases authenticate and authorize federation users as if they are accessing component databases directly. In the other extreme, *low authorization autonomy* fully trusts and relies on the federation to authenticate and authorize federation users. The compromise between those two cases, namely *medium authorization autonomy*, provides component databases with partial control of their shared resources, using the technique called *subject switching*.

2.2 Access Control of XML Documents

Proposals for access control of XML documents can be found in [2, 5–7, 9, 12, 16]. Access control of XML documents differs from traditional access control along two dimensions, *subject* and *object*. From the subject perspective, the population of users in the context of XML documents is usually dynamic and may span a variety of locations [3]. The traditional authentication based on user identity-password pair is usually insufficient and inconvenient. Credentials such as digital certificates are used in authentication to prove both identities and attributes of users. From the object perspective, the rich structure of XML documents demands fine-grained access control capabilities. The desired granularity of access control on XML documents may range from schemata to instances and from documents to elements or attributes.

Access control for XML documents is based on *authorization rules* consisting of *subject*, *object*, *action*, *access*, and other extensions. Most proposals adopt a *view-based* approach, where an *XML view* containing only the authorized content is computed from the original XML document according to applicable authorization rules. The issues of *under-specification* or *over-specification*, which mean that either insufficient or conflicting authorization rules are specified, are addressed in [7] and [12]. The *propagation* of authorizations from an element to its descendents is supported by either a global policy [12] or specific policies defined inside access control rules [2, 7, 16]. Detailed algorithms for computing the XML view are given in [7, 12, 16]. The problem of specifying subjects with their attributes is addressed in an XML-based language, called *x*-Sec in [2]. An extension to traditional access control,

called *provisional authorization*, is proposed in [16] to add richer semantics to XML access control.

3. Logical Architecture for XML Federations

In this section, we propose a logical architecture for XML federations. This is a fundamental step towards the further understanding of various design issues in XML federations, including the security issues that we will address in the next section. The five-level architecture proposed in [17] for the traditional FDBSs needs to be adapted according to the unique requirements of XML federations. As an example, in the traditional FDBS, the local schema in each component DBS must be transformed into a common data model to eliminate or reduce the heterogeneity caused by different data models. In XML federations, this schema translation process is unnecessary since all component XML repositories share the same XML standard (although the schemata of components still need to be transformed to reduce semantic heterogeneity).

Figure 1 illustrates the logical architecture for XML federations. The architecture contains three class of modules, *data*, *schema*, and *process*. In Figure 1, the data modules are represented by ovals, schema by rectangles, and processes by unbounded text. The portion of the figure in dotted lines indicates it is optional in the architecture. Each module of the architecture is discussed next.

Component DBS (Optional) and Component XML Repository
An XML federation is applicable to both the integration of existing XML repositories and the migration of the traditional FDBS. The optional *component DBS* models the traditional databases that join an XML federation through the processes of *data mapping* and *schema mapping*. A *component XML repository* is a stand-alone XML repository to be integrated into an XML federation.

Local, Component, and Export XML Schema
A *local XML schema* is the collection of schemata used by the local XML repository to represent the structure and relationship among local XML documents. A local XML schema could simply be the collection of document type declarations (DTDs) [18]) of local XML documents. A local XML schema usually does not contain sufficient information for integration.

A *Component XML schema* is based on, but different from a local XML schema. Auxiliary semantics in addition to those conveyed in a local XML schema may need to be extracted when placing a component into the grand picture of an XML federation. For example, a local XML schema may need to be enhanced to eliminate semantic heterogeneity, which is not a problem until integration. Figure 2 gives a simple ex-

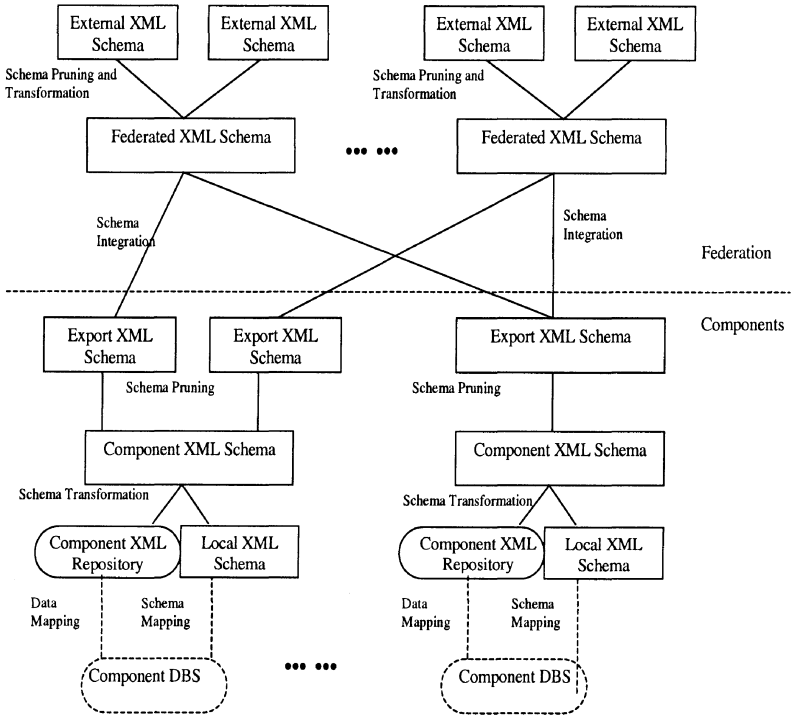

Figure 1. Logical architecture for XML federations.

```
<!ELEMENT   records  (student*)      >
<!ELEMENT   student  (name+,score?)  >
<!ELEMENT   name     #PCDATA         >
<!ELEMENT   score    #PCDATA         >
<!ATTLIST   score    maxScore  (10|100) "100">
<!ATTLIST   student  id  ID #REQUIRED>

<records>
  <student id="123456789">
    <name>Bob</name>
    <score maxScore="10">10.0</score>
  </student>
</records>
```

```
<!ELEMENT   records  (student*)      >
<!ELEMENT   student  (name+,score?)  >
<!ELEMENT   name     #PCDATA         >
<!ELEMENT   score    #PCDATA         >
<!ATTLIST   score    fractalDigit(0.1|1.0|10.0) "0.1">
<!ATTLIST   student  id  ID #REQUIRED>

<records>
  <student id="123456789">
    <name>Bob</name>
    <score fractalDigit="0.1">10.0</score>
  </student>
</records>
```

Figure 2. Semantic heterogeneity in the XML federation.

ample of semantic heterogeneity, where the same tag yields different semantics. Figure 2 shows two XML documents that belong to two different components. Both XML documents are valid according to their local schemata, which are simply DTDs in this case. However, the score

"10.0" has different interpretations due to the lack of semantics in both local schemata.

The *XML-Schema* [19] proposed by W3C suits the needs of both the local XML schema and the component XML schema. The XML-Schema has the advantage of having the functionality above or beyond DTDs and the flexibility of allowing for more extensions than DTDs. As stated by the *W3C XML-Schema Working Group*, the XML-Schema *makes explicit information which may have been implicit in the original document, such as normalized and/or default values for attributes and elements and the types of element and attribute information items.* This *explicit information* is especially important in the context of the XML federation since the issues of heterogeneity usually originate from implicit semantics missing from the local XML schema. Using the XML-Schema in XML federations also benefits from widely available software packages and standards like XSLT in transforming among schemata, because the XML-Schema is in XML format. Figure 3 gives a simple example of applying the XML-Schema in XML federations. The figure shows an exponent XML schema in XML-Schema format, which is transformed from the two local XML schemata in Figure 2. Note that the missing semantics in the local schemata have been added as the restrictions on the simple data type *decimal*, which becomes *explicit* information.

The *Export XML schema* allows components to conveniently customize their shared information. Each export XML schema is a subset of the component XML schema obtained through the process of schema pruning. By using multiple export schemata, a component XML repository may share only part of its resources or different resources for different applications in a federation.

Federated and External XML Schemas

The *Federated XML schema* integrates export schemata from participant component XML repositories. Multiple federated XML schemata can be defined in an XML federation corresponding to a different class of applications. *Each external XML schema* is a subset of a federated XML schema. Since a federated XML schema is usually large and complex in structure, it is convenient to define different external XML schemata for different groups of federation services or users. Compared to multiple federated XML schemata, an external XML schema is an effort towards finer grained schemata, which can be regarded as a *view* defined for each federated XML schema. External XML schemata are obtained by pruning of the federated XML schema.

Data and Schema Mappings

Data and schema mappings convert the data and schema of traditional

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="records">
      <xs:complexType >
        <xs:sequence>
          <xs:element name="student" type="StudentType"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="StudentType>
        <xs:sequence>
          <xs:element name="name" type="xs:string"/>
          <xs:element name="score" type="xs:score" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="id" type="xsd:ID" use="required"/>
      </xs:complexType>
    </xs:element>
    <xs:simpleType name="score">
      <xs:restriction base="xs:decimal">
        <xs:totalDigits value='3'/>
        <xs:fractionDigits value='1'/>
        <xs:minInclusive value='0.0'/>
        <xs:maxInclusive value='100.0'/>
      </xs:restriction>
    <xs:simpleType>
</xs:schema>
```

Figure 3. Using XML schema in XML federations.

databases to those of XML repositories. Very often, databases of different data models already exist before the need for integration arises; hence, data and schema mappings are essential steps towards integrating them into XML federations. Some techniques for data and schema mappings can be found in [4].

Pruning, Transformation, and Integration of Schema

The pruning and transformation of schema are used to generate component XML schema, export XML schema, and external XML schema. The process can be implemented with proprietary methods or standard XSLT processors. However, the implementation must be based on the semantics of component XML repositories. While the automation of those processes is necessary, human intervention is still essential due to the lack of semantics in most schemata.

The schema integration techniques in [1] could be adapted for application in XML federations. The standard XSLT transformation can be used for schema integration if the XML-Schema is used for export XML schemata. As stated in [17], the common format of an export schema is critical for schema integration, which must be capable of representing rich semantics. The integration process is indeed guided by those seman-

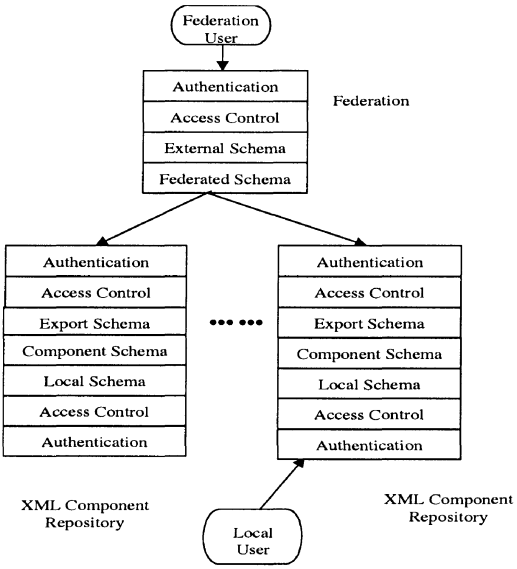

Figure 4. Access control scenario in XML federations.

tics rather than the syntax of schemata. Emerging standards, including the XML-Schema, are promising candidates for this purpose.

4. Access Control in XML Federations

In this section, we address the access control problem in XML federations on the basis of the logic architecture presented above. The access control scenario in XML federations is shown in Figure 4.

We state our assumptions as follows. We only consider federation users accessing the federation, since access control of local users in components is the same as that discussed in Section 10.2. When a human acts as both a federation user and a local user at the same time, issues such as the *inference problem* (sensitive information may be derived by combining the query results from both the federation and its components) may arise. Those issues are out of the scope of this paper. We assume federation bases its query processing on a schema. When some data of components are replicated in a federation, its access control is more like the local data in its components and is not addressed here. We assume an *administrative paradigm* in our study since most works on access control of XML documents do so. We do not consider conflicting authorization rules between the federation and components [13]. We assume that XML federations may collect personal information through authentication and keep them as *user profiles*. We also assume that each component trusts

a federation for data confidentiality and integrity, that is, once the requested data is provided by components, a federation will honestly process and present the final outcome only to the user who requests it. However, malicious federation users may collude with a federation to obtain unauthorized accesses to components under the identity of normal users.

The main issues of access control in XML federations can be described by the following questions about the authentication and authorization of a federation user in response to his/her queries: *Who authenticates the federation user? Who authorizes him/her? How could the authentication and authorization be reliably done?* The answers to the first two questions lead to several access control schemes proposed in [14]. However, applying those schemes to XML federations causes some problems that are either ignored in the literature or specially pertain to this case. To address those problems, first we need some notations for the conciseness of our discussions.

XML Federation We use the pair $< F, C_i >$ for an XML federation consisting of n components, where F is the federation and each $C_i (i = 1, 2, \ldots, n)$ is a component. We use FU (or FU^j) as a federation user and CU_i (or CU_i^j) as a local user accessing C_i. $profile(X)$ is the set of profiles (collection of properties about a user) of a set of users X satisfying $profile(\emptyset) = \emptyset$, where $X \in \{\{FU^j\}, \{CU_i^j\}, \emptyset\}$. q is a query posed by any FU on F, and q_i is the portion of q that should be processed by C_i according to the federated schema.

Authentication $auth(X, Y)$ denotes authentication. $X \in \{F, C_i\}$ is the authenticator, and $Y \in \{\{FU^j\}, \{CU_i^j\}\}$ is the set of users being authenticated. $auth(X, Y)$ returns $Y' \subseteq Y$, where the authentication succeeds for any $y \in Y'$ and fails for all $y \in Y \setminus Y'$. $switch(FU, C_i, q)$ is the process of *subject switching,* which returns $\{CU_i^j\}$.

Authorization R_F and R_i are the authorization rules of F and C_i, respectively. $V(X)$ returns an XML file containing the information requested by query X, where $X \in \{q, q_i\}$. We use $\sum V_i$ to represent the view integration process at F, where each V_i is an XML file returned by C_i. $P(profile(X), V, R)$ is the process of view pruning[1] for the purpose of authorization, where $X \in \{\{FU^j\}, \{CU_i^j\}, \emptyset\}$ is the set of users who initiate the request, V is the XML file to be pruned, and R is the authorization rules. The process returns the pruned XML file. We require $P(\emptyset, V, R) = \emptyset$ for any V and R, and $P(profile(X), V, \emptyset) = V$ for any X.

[1] We represent the processes of view-computing and view-pruning separately, while they are sometimes integrated as one process.

Scheme 1: Federation Authentication - Component Authentication and Authorization

In this scheme, FU is first authenticated by F, and is further authenticated and authorized by each C_i. This is elaborated with the following formula:

$V = \sum_i P(profile(auth(F, \{FU\}) \cap auth(C_i, \{FU\})), V(q_i), R_i)$

This scheme corresponds to *local authentication* in a traditional FDBS. As stated in [14], this scheme preserves *full authorization autonomy* since each C_i knows who is accessing which part of its resources. On the other hand, it was considered cumbersome because each FU may have to go through multiple authentications $auth(C_i, \{FU\})$ for every q it submits. However, there are actually more problems with this scheme than just multiple authentications. How would C_i be convinced that $auth(C_i, \{FU\})$ is reliable when C_i and FU have to interact through the untrusted F? For example, F may store any credential submitted by FU^1 and replay it later for another malicious FU^2. It is also possible that F sends q_i to C_i on behalf of the authenticated FU^1, while q is actually submitted by FU^2. Our solution to these problems is illustrated as follows. Note that we use $[X]k$ for the public-key encryption of X using k, $\{X\}k$ for the digital signature of X using k, and $h(X)$ for the hash of X (various standards are available for encryption, signature, and hash, which are not discussed here).

1. $FU \rightarrow F, F \rightarrow C_i$: request for authentication
2. $F \leftarrow C_i$: random number r_i
3. $FU \leftarrow F$: digital certificate of C_i containing its public key k_i, r_i
4. $FU \rightarrow F, F \rightarrow C_i$: [credential of FU, r_i, session key $k_s]k_i$
5. $FU \rightarrow F$: $q, \{h(q)\}k_s$
6. $F \rightarrow C_i$: $q, q_i, \{h(q)\}k_s$

In the above procedure, steps 1 to 4 are executed when FU initiates a new session, and steps 5 and 6 are executed whenever FU submits a new query q. We briefly explain how this scheme works. First, the confidentiality of the credential of FU is achieved through the encryption with k_i in step 4, which means F cannot learn this credential from the messages it receives. Second, the uniqueness of the credential during each session is achieved by including r_i in step 4, that is, F is not able to store and replay the credential for the authentication of other users. Finally, because each following query q submitted by FU in the same session is digitally signed with k_s, which is only known to FU and C_i, the integrity of q is achieved in that F cannot alter q or replace it with another query (submitted by other users). Some details are not shown in the procedure. Each session ends through either the time-out enforced by C_i or a termination request submitted by FU. Query q and the partial query results returned by each C_i are still sent to F in the clear, which is essential for F to process them. The client software module is required to automate the process

of authentication, and its integrity during distribution must be guaranteed. Although our solution shares some ideas of public key-based network protocols like SSL, those protocols do not directly apply to our problem (they are used to secure the network communications).

While Scheme 1 might be useful for the case that C_i does not trust F for authentication and authorization, each C_i must keep track of every FU. This may not always be feasible considering that the population of FU is very dynamic in XML federations. In the following two schemes, authentication is only done by F and C_i trusts F for authentication.

Scheme 2: Federation Authentication - Component Authorization

In this scheme, FU is authenticated by F and authorized by each C_i respectively. This is summarized as:

$V = \sum_i P(profile(auth(F, \{FU\})), V(q_i), R_i)$

This scheme provides each C_i with authorization autonomy and releases it from the burden of keeping track of each FU. The problem we need to address is how C_i could authorize FU without knowing all details about it.

Our solution is *profile-based authorization.* We describe the basic idea without giving all the details. We require each authorization rule R_i to be defined on the basis of the attributes included in user profiles, instead of user identities. After F authenticates FU, it forwards $profile(auth(F, \{FU\})$ to C_i. C_i then extracts the attributes required for authorization from the profile. A profile could be formatted as a special XML file so its processing can be done through XSLT processors. Profile-based authorization is more suitable for XML federations than for traditional FDBSs, because most existing practices of access control for XML documents base authorization on roles, organizations, locations, or other attributes, which collectively form a user profile.

This scheme is useful and efficient for the cases where coarse authorization granularity of subjects is acceptable to components. Otherwise, if authorizations are given to each user differently, then this scheme does not apply. The following scheme shifts this authorization burden to the federation.

Scheme 3: Federation Authentication and Authorization

In this scheme, each FU is authenticated and authorized by F. The authorization is enforced by F through either pruning the partial results before integrating them or pruning the integrated result. This is summarized as:

$V = \sum_i P(profile(auth(F, \{FU\})), V(q_i), R_F)$

or

$V = P(profile(auth(F, \{FU\})), \sum_i V(q_i), R_F)$

This scheme corresponds to the case of *low authorization autonomy* in traditional FDBSs. In [14], it is regarded as being able to benefit each component DBS with the finer access control of the federation. However, one issue arises when applying it to XML federations. As stated in Section 10.2, fine-grained authorization is inherently necessary for XML documents. According

to the logical architecture we proposed, F only knows the schemata of shared resources, which may be insufficient for it to enforce any instance-level authorizations. One solution is for F to import fine-grained authorization rules from C_i. However, having fine-grained rules without having the associated fine-grained data is still problematic. For example, assume that some instances of an element in C_i are not intended to be shared with F under a default open policy. In this case, the access should not be granted to any FU even though there are no explicit prohibition rules for those instances. Enabling F to enforce such types of authorization rules may lead to a substantial replication of the data of C_i in F, which is in turn not acceptable due to the redundancy and potential inconsistency of data. In summary, this scheme will be valid only when the desired authorization granularity is feasible for F to enforce.

Scheme 4: Federation Authentication and Authorization with Subject Switching

In this scheme, each FU is authenticated and authorized by F with subject switching [20]. Each switched CU_i^j is authenticated and authorized by C_i. This is summarized as:
$V = P(profile(auth(F, \{FU\})), \sum_i P(profile(auth(C_i, switch(FU, C_i, q))), V(q_i), R_i), R_F)$

Like Scheme 2, this scheme is a compromise between no trust and complete trust of F for authentication and authorization. However, this scheme is unique in that it provides authorization autonomy on the basis of *accountability* (i.e., the ability of F to testify about what happened) of subject switching. As long as accountability is ensured, C_i retains control of the resources it shares with F. The advantage compared to Scheme 3 is that no FU can be granted the data that is not intended to be shared by C_i, even if F malfunctions in subject switching. Compared to Scheme 1, C_i does not have the burden of knowing every FU until it needs to audit the history of subject switching.

Unfortunately, the critical problem of this scheme, the accountability of subject switching, is not considered by the literature. Here we give our solution, namely, *delayed local authentication*, as follows.

1. $FU \rightarrow F, F \rightarrow C_i$: request for authentication
2. $F \leftarrow C_i$: random number r_i
3. $FU \leftarrow F$: digital certificate of C_i containing its public key k_i, r_i
4. $FU \rightarrow F, F$ keeps in log file Log_f : [credential of FU, r_i, session key k_s, timestamp $t_a]k_i$
5. $FU \rightarrow F, F$ keeps in Log_f : $q, \{h(q),$ timestamp $t_1\}k_s$
6. $F \rightarrow C_i : \{CU_i\} = switch(FU, C_i, q)$, q, q_i
7. C_i keeps in log file $Log_i : \{CU_i\}$, q, q_i, timestamp t_2

Our solution is similar to the procedure in Scheme 1 except that C_i does not authenticate each FU; instead, the credential information is logged by F in

Log_f, as does each digitally signed q. When F switches to a local user CU_i and sends q_i to C_i, C_i authenticates CU_i and logs the requests. Accountability is achieved because F cannot create a record for FU in Log_f without the real credential of FU. If F attempts to request accesses for CU_2 under the identity of CU_1 with subject switching, it will succeed for the time being. However, either the signature of queries $\{h(q), t_1\}k_s$ or the associated authentication records for FU_1 to obtain k_s will be missing in Log_f. C_i may later discover this violation by comparing Log_f and Log_i. Timestamps are logged to ensure a session is correctly timed out by the client (C_i has no knowledge of sessions). Note that although F could potentially delete or alter Log_f, this action only exacerbates its situation in auditing.

5. Conclusions

In this paper, we described a logical architecture for integrating isolated XML repositories into XML federations. We investigated the access control issues of XML federations. We pointed out the new issues in applying existing access control schemes to XML federations and gave our solutions. Future work includes implementing a prototype of an XML federation and further investigation of various security issues in XML federations.

Acknowledgments

This work was partially supported by the National Science Foundation under grant CCR-0113515.

References

[1] Batini, C., Lenzerini, M. and Navathe, S. (1986). A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364.

[2] Bertino, E., Castano, S., Ferrari, E. and Mesiti, M. (2000). Specifying and enforcing access control policies for XML documents sources. *World Wide Web Journal*, 3(3):139–151.

[3] Bonatti, P. and Samarati, P. (2000). Regulating service access and information release on the web. In *Proceedings of the Seventh ACM Conference on Computer and Communications Security*, pages 134–143.

[4] Collins, S., Navathe, S. and Mark, L. (to appear). XML schema mappings for heterogeneous database access. *Information and Software Technology (Special Issue on Objects)*.

[5] Damiani, E., De Capitani di Vimercati, S., Paraboschi, S. and Samarati, P. (2000a). Design and implementation of an access control processor for

XML documents. In *Proceedings of the Ninth International World Wide Web Conference.*

[6] Damiani, E., De Capitani di Vimercati, S., Paraboschi, S. and Samarati, P. (2000b). Regulating access to semistructured information on the web. In *Proceedings of the IFIP-TC11 International Conference on Information Security.*

[7] Damiani, E., De Capitani di Vimercati, S., Paraboschi, S. and Samarati, P. (2000c). Securing XML documents. In *Proceedings of the International Conference on Extending Database Technology*, pages 121–135.

[8] Damiani, E., De Capitani di Vimercati, S., Paraboschi, S. and Samarati, P. (2001). Fine grained access control for SOAP e-services. In *Proceedings of the Tenth International Conference on the World Wide Web*, pages 504–513.

[9] Damiani, E., De Capitani di Vimercati, S., Paraboschi, S. and Samarati, P. (2002a). A fine-grained access control system for xml documents. *ACM Transactions on Information and System Security*, 5(2):169–202.

[10] Damiani, E., De Capitani di Vimercati, S., Paraboschi, S. and Samarati, P. (2002b). Securing soap e-services. *International Journal of Information Security*, 1(2):100–115.

[11] Dawson, S., Samarati, P., De Capitani di Vimercati, S., Lincoln, P., Wiederhold, G., Bilello, M., Akella, J. and Tan, Y. (2000). Secure access wrapper: Mediating security between heterogeneous databases. In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX).*

[12] Gabillon, A. and Bruno, E. (2001). Regulating access to XML documents. In *Proceedings of the Fifteenth IFIP WG11.3 Conference on Database and Applications Security*, pages 311–328.

[13] Gudes, E. and Olivier, M. (1998). Security policies in replicated and autonomous databases. In *Proceedings of the Twelfth IFIP WG 11.3 Conference on Database Security*, pages 93–107.

[14] Jonscher, D. and Dittrich, K. (1994). An approach for building secure database federations. In *Proceedings of the Twentieth VLDB Conference*, pages 24–35.

[15] Jonscher, D. and Dittrich, K. (1995). Argos - a configurable access control system for interoperable environments. In *Proceedings of the IFIP WG 11.3 Workshop on Database Security*, pages 43–60.

[16] Kudo, M. and Hada, S. (2000). XML documents security based on provisional authorization. In *Proceedings of the Seventh ACM Conference on Computer and Communications Security*, pages 87–96.

[17] Sheth, A. and Larson, J. (1990). Federated database system for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236.

[18] WWW Consortium (2000). Extensible markup language (XML) 1.0. Available at www.w3.org/TR /REC-xml.

[19] WWW Consortium (2001). XML schema 1.1. Available at www.w3.org/ XML/ Schema.

[20] Yang, J., Wijesekera, D., and Jajodia, S. (2001). Subject switching algorithms for access control in federated databases. In *Proceedings of the Fifteenth IFIP WG11.3 Conference on Database and Applications Security*, pages 199–204.