Flow Policies: Specification and Enforcement

E. Bertino
CERIAS
Purdue University
bertino@cerias.purdue.edu

E. Ferrari
DSCFM
University of Insubria
Elena.Ferrari@uninsubria.it

G. Mella
DICO
University of Milano
mella@dico.unimi.it

Abstract

This paper deals with the problem of secure cooperative updates for XML documents in distributed systems. In particular, we introduce the basic notions underlying a flow language by using which a user can specify the flow that a given XML document has to follow within a group of cooperating subjects. A key feature of the flow language is to be based on the notion of subject credentials. In addition, we describe a policy language to specify special-purpose authorizations allowing selected subjects to modify or extend a given document flow. Finally, we briefly describe the protocols for verifying that the path followed by a document in a collaborative group agrees with the specified flow and to verify that modifications on a given flow are in accordance with the specified authorizations.

1 Introduction

The exchange of documents on the Web, in particular XML documents [5], in the framework of collaborative and distributed applications [6] involving different parties, such as subjects belonging to different organizations, requires a proper infrastructure. In particular, confidentiality and integrity must be preserved for documents flowing among different parties making also sure that only authorized subjects be able to modify the documents. An approach to achieve such goals is based on encrypting the document contents and on generating some special-purpose control information, that are used by a subject to locally check the integrity of the document portions for which it possesses at least an authorization [1]. The encrypted document and the corresponding control information form the so called *package*. Another key requirement, that, up to now, has not been widely investigated, when dealing with distributed and collaborative applications, is the support for the specification of *document flow policies*, that is, policies regulating the set of subjects (hereafter called *collaborative group*), that must receive a package during the update process. We believe that this is an important requirement since document updates in many organizations must be governed by specific policies that reflect the internal rules of the organizations. Many issues need to be addressed for achieving this goal. First, a flow policy specification requires the development of a high level specification language. The main features of this language must be the possibility of generating totally or partially specified lists of subjects that will have to receive a document in a distributed and cooperative update process and the possibility of specifying which subjects can extend a flow policy by adding new receivers. We also believe that the language must support flexible ways of qualifying subjects, based on the notion of credentials. Secondly, the updates to the original document and to the original flow policy must be regulated by proper access control policies and modification control rules, respectively. These policies and rules must be specified by the subjects, called here and in what follows *originators*, that have generated the documents and flow policies. Thus, the originators have to specify who can modify which portions of a particular document or flow policy and which privileges can be exercised over them. Finally, a proper infrastructure is required for the decentralized enforcement of the stated policies and rules.

In this paper, we propose an approach to these issues. Our approach is based on the generation of some *control information*, that are attached to the document and updated as the document flows, and that make a subject able to locally check the correctness of the path followed by the document till that point and of the modifications performed over it. We first present the language we have developed for specifying flow policies and modification control rules. Then, we present the architecture and related protocols that we have developed for supporting distributed and collaborative update processes for XML documents (see Figure 1). The architecture includes a Parser, that analyzes the document and the corresponding flow policy. It applies the access control policies to the document,

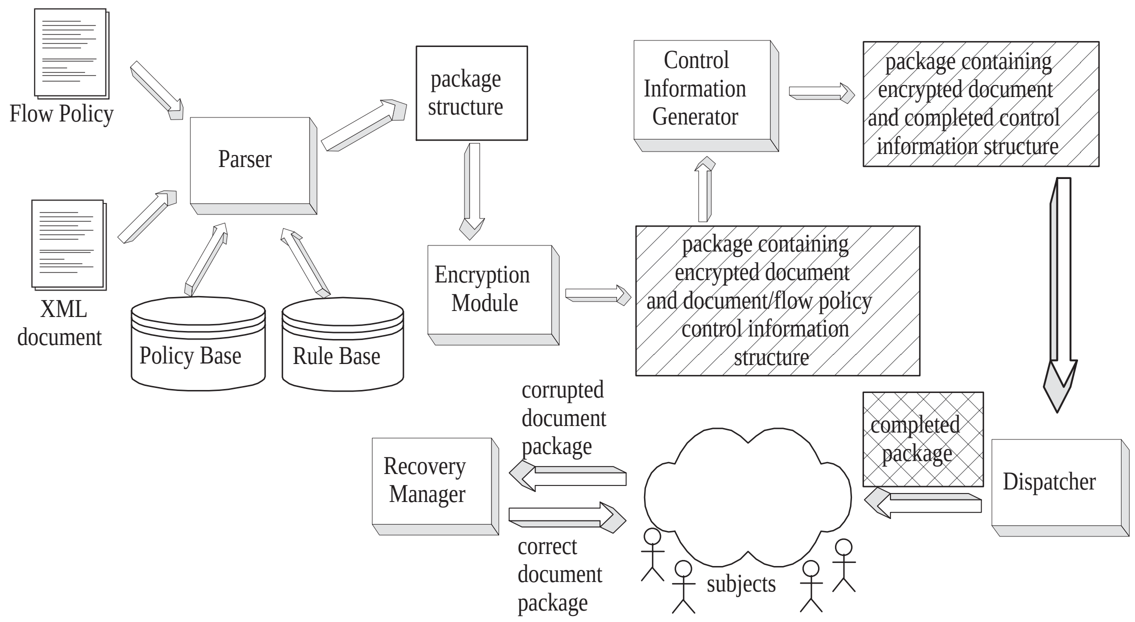
Flow Policy
XML document
Parser
Policy Base
Rule Base
package structure
Encryption Module
package containing encrypted document and document/flow policy control information structure
Control Information Generator
package containing encrypted document and completed control information structure
Dispatcher
completed package
subjects
corrupted document package
correct document package
Recovery Manager

Figure 1: Distributed update of documents: overall schema

grouping together the portions to which the same set of access control policies apply. Then, it applies the modification control rules to the flow policy portions grouping them according to the set of rules that apply to those portions. As a result of this first step we have the package structure containing the document and the flow policy portions grouped according to the above-mentioned strategy. Then, the Encryption module, another relevant component of our architecture, encrypts the document portions using a different key for each generated group, for confidentiality purposes. Then, the Control Information Generator generates some control information for both the document and the flow policy. Finally the encrypted document, the flow policy and all the control information are inserted in the package by the Dispatcher module. This module is also in charge of sending the package to the first chosen subject in the collaborative group. The Recovery module receives recovery requests from the subjects and sends back the last correct version of the received corrupted package.

Since in a previous paper [2] we have already presented an approach to distributed and collaborative updates of XML documents, in this paper we focus on flow policy management.

The approach we present is based on the following assumptions: 1) all the originators are considered trusted; 2) a package can be sent to only one subsequent receiver; 3) no loss of messages, that is each message sent to a subject is certainly received.

To the best of our knowledge, the work reported in this paper is the first addressing the problem of flow policy management in distributed systems. This work is part of the Author-$\mathcal{X}$ project [3], whose aim is to provide a comprehensive system for the protection of XML documents.

The remainder of this paper is organized as follows. Section 2 presents the flow policy modification model on which our approach relies, introducing also the flow policy specification language. Section 3 presents the control information required by our approach for the integrity check of the flow policy content, whereas Section 4 introduces the information inserted in the package by the subjects that have modified the document/flow policy content. Finally, Section 5 presents the protocols at the originator and receiver sides, whereas Section 6 concludes the paper.

2 Flow Policy Modification Control Model

A modification control model is necessary to specify who can modify which portions of a flow policy and how it can do that. Before presenting our flow policy modification control model we need to better introduce the concept of flow policy. A flow policy contains the sequence of subjects that must receive the package with which it is associated. This sequence can be fully specified in advance, at the beginning of the update process, or partially specified when the process starts and then modified and extended by authorized subjects. A flow policy attachment does not necessarily contain the univocally specified list of receivers, but it can contain some *receiver specifications*, that is, sets of properties that have to be verified by the receivers. Each receiver specification can also contain some alternative *receiver profiles*, specified by using an XML-based language called $\mathcal{X}$-Sec [4]. A receiver is considered *legal* for a particular receiver specification if it satisfies at least one of the receiver profiles that belong to that specification. Our flow policy specification language enables also an originator to grant a receiver the permission of extending a flow policy by inserting a *sub-flow policy*. This is supported by associating the value `subpath`, that enables an insertion, or `nosubpath`, that denies an insertion, with a receiver profile. This value is denoted as an *extension specification.*

Example 1 *An example of flow policy is the following:*

```
<{rs1,(rp1,"//vice_manager[@Department="R&D"]",
  subpath), (rp2, "//secretary[@Department="R&D"]",
  nosubpath)}, {rs2,(rp3,"//accountant[@level="third"]",
  subpath)}, {rs3,(rp4,"//company_director", subpath)}>
```

which specifies that the first receiver must be a vice manager or a secretary of the "R&D" Department; the second receiver must be a third level accountant; whereas the third receiver must be a company director. Moreover, the flow policy specifies that whereas vice managers, accountants and company directors are entitled to insert a new sub-policy into this flow policy, secretaries are not enabled to do so.

`{rs1,(rp1,"//vice_manager[@Department="R&D"]",subpath), (rp2,"//secretary[@Department="R&D"]", nosubpath)}` *is an example of receiver specification consisting of two receiver profiles:*

`(rp1,"//vice_manager[@Department="R&D"]",subpath)` *and* `(rp2, "//secretary[@Department="R&D"]", nosubpath)`. ○

Our modification control model consists of a set of modification control rules specified in terms of *subjects* that can modify a flow policy, a *privilege* that can be exercised by authorized subjects, an *object* on which the privilege can be exercised, and some *propagation options*, to reduce the number of rules to be specified.

Subjects. In our model subjects are qualified by means of conditions specified against *credentials*. Figure 2 shows an example of XML credentials specified according to $\mathcal{X}$-Sec [4]. Each subject has one or more credentials, issued by different Certification Authorities (CAs). Conditions specified on credentials are denoted as *credential expressions*, and are specified by means of an XPath-based language [7].

```
<vice_manager cid="50">
  <name>
    <Fname> Jim </Fname>
    <lname > Mason </lname>
  </name>
  <age> 52 </age>
  <department> R&D </department>
  <salary> 9,000 </salary>
  <category > Top Executive </category>
</vice_manager>
```

```
<secretary cid="12", vice_manager="50">
  <name>
    <Fname> Alice </Fname>
    <lname > Brown </lname>
  </name>
  <age> 38 </age>
  <department> R&D </department>
  <salary> 2,000 </salary>
  <level > third </level>
  <duty > vice_manager secretary </duty>
</secretary>
```

Figure 2: Examples of XML credentials

Privileges. The privileges supported by our model are `delete` and `update`. The first privilege can be exercised over one or more receiver specifications or over one or more receiver profiles; whereas the second can be exercised only over the credential expressions that describe the receiver profiles or the extension specifications.

Objects. Objects to which a modification control rule applies can be receiver specifications, receiver profiles, credential expressions, and extension specifications, according to the privilege specified in the rule, as discussed above.

Propagation options. Finally, our model supports the definition of two propagation options: `PROP` and `NO_PROP`. The propagation option `PROP` causes the application of a rule to the specified object and to all the objects that compose it, whereas `NO_PROP` causes the application of a rule to the specified object only. The set of rules specified for the flow policies generated by a flow policy's originator are locally stored into a repository called *Rule Base* ($\mathcal{RB}$).

Example 2 *Examples of rules referred to the flow policy of Example 1 are the following:*

```
<(r_id1,delete,//vice_manager[@cid="50"],rp3,NO_PROP),
  (r_id2,update,//vice_manager[@cid="50"],rp4,PROP)>
```

The first rule allows vice_manager Jim (see Figure 2) to delete the receiver profile contained in the second receiver specification, whereas the second rule allows the same subject to update the credential expression and the extension specification associated with the receiver *profile contained in the third receiver specification.* ○

3 Flow Policy Control Information

To correctly enforce the modification of the flow policy portions and to allow a receiver to locally check the integrity of those portions, we need to associate with a flow policy some control information, referred to as *flow policy attachment.* Flow policy portions are grouped together according to the set of rules that apply to them forming some *regions.* All the flow policy portions to which no rule applies belong to a unique *non-modifiable region*, whereas the other portions belong to *modifiable* regions. Control information associated with the non-modifiable region consists of a hash value computed over all the flow policy portions that belong to that region. To protect the authenticity of this information the flow policy's originator encrypts this hash value with its private key.

Before presenting the control information associated with modifiable regions we have to introduce the concept of *flow policy modification certificate.* A flow policy modification certificate is generated by the flow policy's originator according to the rules in $\mathcal{RB}$. Given a modification control rule *mcr* belonging to $\mathcal{RB}$ and a subject *sbj* to which *mcr* applies a flow policy modification certificate, generated for *sbj* according to *mcr*, contains the following information: the *sbj*'s public key (`sbj-pubkey`), the privilege contained in *mcr* (`priv`), and the set of regions and corresponding flow policy portions, to which *mcr* applies. Each certificate is signed by the flow policy's originator with its private key for authentication purpose and distributed to the subject to which it belongs to.

Modifiable regions can be classified in three different categories: *updatable*, *deletable*, and *updatable-deletable* regions. Updatable regions contain flow policy portions to which only rules containing the update privilege apply. Deletable regions contain flow policy portions to which only rules containing the delete privilege apply. Whereas updatable-deletable regions contain flow policy portions to which rules containing both delete and update privilege apply. Control information associated with a modifiable region is different according to the corresponding category. For example, a signature is computed by the last subject that has updated an updatable region *ur* on the flow policy portions belonging to *ur* itself and inserted in the flow policy attachment. A modifiable region also contains a set of flow policy modification certificates used by the subsequent receivers to verify that the modifications executed till that point over the region are correct wrt the specified policies and rules.

4 Receiver Declarations

A subject that has received a package can exercise on the document and/or on the flow policy portions the privileges contained in the document/flow policy modification certificates that it has received by the originators. The j^{th} receiver of a package must satisfy the j^{th} receiver specification in the flow policy to be a valid receiver. To allow subsequent receivers to check that it is really a valid j^{th} receiver, it has to insert within the j^{th} receiver specification *rs* one of its credentials that satisfies at least one of the credential expressions associated with the alternative receiver profiles belonging to *rs.* Moreover, it has to specify the identity of the next receiver. Finally, it can also insert two types of declarations within *rs*: a *document modification declaration* and/or a *flow policy modification declaration.* The former declaration contains the list of all modification operations executed over document portions. Each declared operation is given in terms of the executed privilege and list of modified document portions. A corresponding certificate must be inserted in the modified region in the proper region control component. The latter declaration is similar to the former one, indeed also in this case it contains the list of modification operations executed over the flow policy. A corresponding flow policy modification certificate must be inserted in the proper region. Before sending the updated package to the next receiver it has also to sign the whole *rs*'s content for authentication and integrity purposes. All the declarations stored in the flow policy form the so called *modification history* of both the document and the flow policy.

To prevent a subject *sbj* that has received several times the same package from sending an old package version to another subject, each receiver sends its current version of the flow policy attachment to all the other subjects involved in the distributed and collaborative update process.

Example 3 *Figure 3 shows the package received by the first subject involved in a distributed and collaborative process and the updated package sent by this subject to the subsequent one. In particular, the first subject deletes a receiver profile belonging to the third receiver specification, according to the flow policy modification certificate (fpmc1) that it has received by the proper flow policy originator. It also inserts in the flow policy attachment the required information before signing the content of its corresponding receiver specification (i.e., the first one) and then it inserts in the package the used certificate in the proper flow policy region (fpaR2).* ○

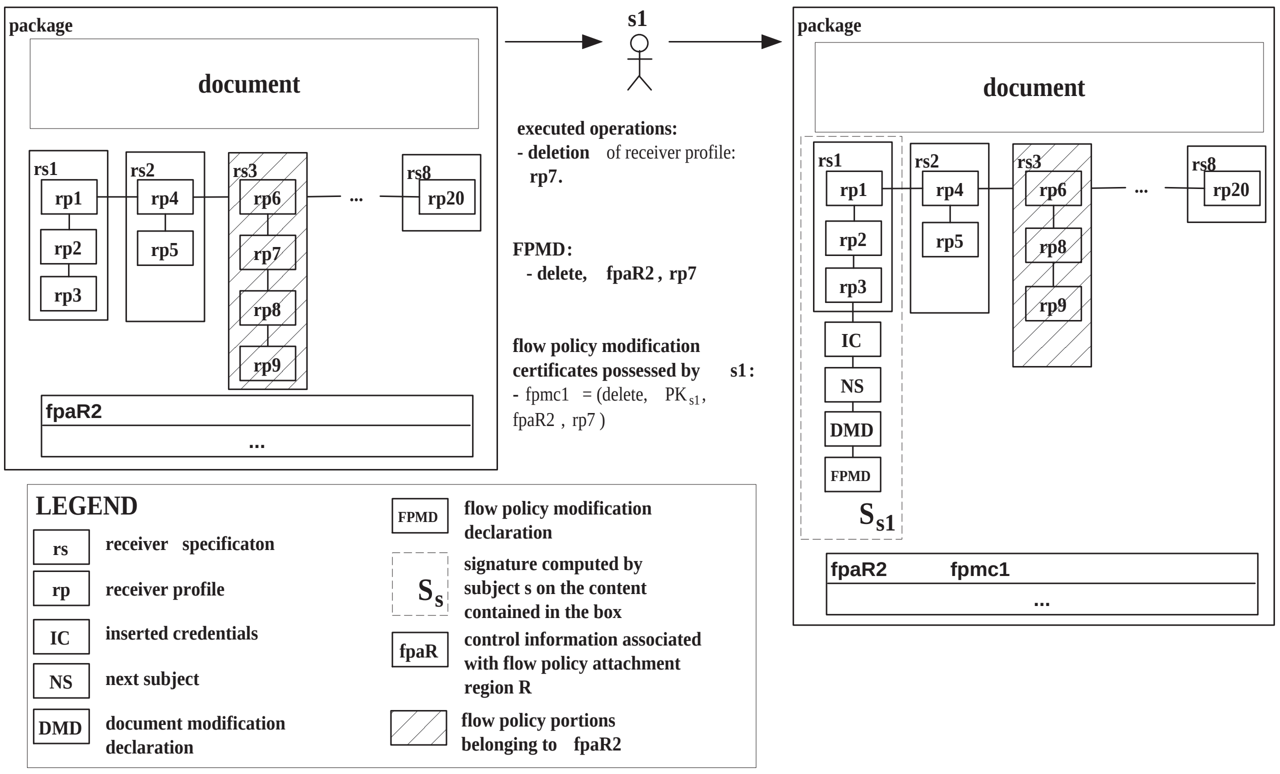
package
document
rs1
rp1
rp2
rp3
rs2
rp4
rp5
rs3
rp6
rp7
rp8
rp9
...
rs8
rp20
fpaR2
...
s1
executed operations:
- deletion of receiver profile: rp7.
FPMD:
- delete, fpaR2, rp7
flow policy modification certificates possessed by s1:
- fpmc1 = (delete, PKs1, fpaR2, rp7)
package
document
rs1
rp1
rp2
rp3
IC
NS
DMD
FPMD
Ss1
rs2
rp4
rp5
rs3
rp6
rp8
rp9
...
rs8
rp20
fpaR2 fpmc1
...
LEGEND
rs receiver specificaton
rp receiver profile
IC inserted credentials
NS next subject
DMD document modification declaration
FPMD flow policy modification declaration
Ss signature computed by subject s on the content contained in the box
fpaR control information associated with flow policy attachment region R
flow policy portions belonging to fpaR2

Figure 3: Package received by a subject and then sent after execution of some modifications

5 Protocols

In this section we present the protocols executed by a receiver to check the integrity of a package and to exercise privileges on the document or flow policies, and the protocols executed by the originators to recover a corrupted package version.

5.1 Receiver Protocol

A subject *sbj* upon receiving a package executes the integrity check procedure to detect possible corruptions to the package content. First, the flow policy and the document are analyzed. Since a subject has received by the subjects that have already received the package the corresponding flow policy attachments and it has saved the correct one with the highest number of receivers, it is enabled to check if the flow policy contained in the received package contains or not all those receivers that have notified their package reception. If this correspondence is not verified an error occurs and *sbj* sends a flow policy recovery request to the originator of the flow policy (or sub-flow policy) that contains the sender of the received package. The same activity is executed if during the flow policy integrity check an error occurs. Moreover, if an error occurs during the document integrity check process a recovery request is sent to the document's originator to obtain a package that contains the last correct version of the corrupted regions. The integrity verification is based, in both cases, on the check that the current content is the result of the execution of all and only the operations declared in the flow policies by the previous receivers starting from the original content and that such modification operations are legal wrt the policies and the rules stored respectively in the repository called *Policy Base*($\mathcal{PB}$) and $\mathcal{RB}$ of the originators.

5.2 Originator Protocol

An originator in our approach is in charge of generating documents and/or flow policies, all the corresponding control information and all the needed certificates. It has also to take care of sending the decryption keys associated with document regions, document modification certificates, and flow policy modification certificates to the proper subjects. Moreover, an originator has to manage recovery requests received by subjects involved in the update process. If it receives a flow policy recovery request regarding the delivery of an old version of a package or concerning an error affecting a region in a (sub-)flow policy, the originator sends a message to each subject stored in the flow policy attachment that it has locally saved, to obtain a package containing a correct flow policy version, starting from the last subject in the flow policy and going backwards. The process ends when the originator receives a package satisfying the required property. Once the package has been recovered, as last step, the originator sends to the subject *sbj* from which it has received that package the request of sending that package to a different receiver. The originator also notifies all the other subjects that *sbj* will send the rebuilt package to a receiver subject that it chooses. Note that such a chosen receiver must be different from the subject to which the package had been sent before by *sbj*. If the originator receives a document recovery request for a document that it has generated, it sends a message to each subject stored in the flow policy attachment that it has locally saved to obtain a package containing the last correct version of one or more corrupted regions, starting from the last subject in the flow policy and going backwards. The process ends when the originator collects from the received packages the last correct version of all the corrupted regions. Then, it inserts in the package received by the subject *s* who sent the recovery request all these correct region versions and it sends back this updated package to *s*. Finally, the originator notifies all the other subjects the set of document modification declarations that must no longer be considered for document integrity check.

6 Concluding Remarks

In this paper we have presented how a distributed and collaborative update process can support specification and modification of flow policies. In particular we have shown the flow policy modification control model that regulates who can modify a flow policy or its portions and the infrastructure needed to enforce a correct exercise of the privileges supported by our model. We plan to extend this work along two main directions. First, a feature that we intend to address is the possibility, for a receiver, of entering, in the flow policy attachment, only the required information needed to guarantee that it is a valid receiver, instead of a whole credential. Secondly, we plan to implement a prototype system and test the performance and the overhead implied by our solution.

References

[1] E.Bertino, E.Ferrari, G.Mella, "A Framework for Distributed and Cooperative Updates of XML Documents", *Proc. of the 16th Annual IFIP WG 11.3, Working Conference on Data and Application Security*, Cambridge, UK, July 2002.

[2] E.Bertino, E.Ferrari, G.Mella, "An Approach to Cooperative Updates of XML Documents in Distributed Systems", *Technical Report*, DICO, University of Milano, 2003.

[3] E. Bertino, B. Carminati, E. Ferrari, G. Mella, "Author-X - A System for Secure Dissemination and Update of XML Documents", *Proc. of Third International Workshop on Databases in Networked Information Systems (DNIS 2003)*, Aizu, Japan, September 2003.

[4] E. Bertino, S. Castano, E. Ferrari, "On Specifying Security Policies for Web Documents with an XML-based Language". *Proc. of the ACM Symposium on Access Control Models and Technologies (SACMAT'2001)*, Fairfax, VA, May 2001.

[5] World Wide Web Consortium. Extensible Markup Language (XML) 1.0, (Second Edition) 2000. Available at `http://www.w3.org/TR/2000/REC-xml-20001006`.

[6] B.Thuraisingham, A. Gupta, E.Bertino, E.Ferrari, "Collaborative Commerce and Knowledge Management Across Borders", *Knowledge and Process Management*, Vol.9, No. 1, pp. 43-53, January 2002.

[7] World Wide Web Consortium. XML Path Language (Xpath), 1.0, 1999. Available at: `http://www.w3.org/TR/1999/REC-xpath-19991116`.