An Update Protocol for XML Documents in Distributed and Cooperative Systems *

Yunhua Koglin † Giovanni Mella ‡ Elisa Bertino § Elena Ferrari ¶

Abstract

Securing data is becoming a crucial need for most internet-based applications. Whereas the problem of data confidentiality has been widely investigated, the problem of how to ensure that data, when moving among different parties, are modified only according to the stated policies has been so far not deeply investigated. In this paper, we propose an approach supporting parallel and distributed secure updates to XML documents. The approach, based on the use of a security region-object parallel flow *(S-RPF) graph protocol, is particularly suited for all environments requiring cooperative updates to XML documents. It allows different users to simultaneously update different portions of the same document, according to the specified access control policies. Additionally, it supports a decentralized management of update operations in that a subject can exercise its privileges and verify the correctness of the operations performed so far on the document without interacting, in most of the cases, with the document server.*

1. Introduction

The widespread use of the Internet for exchanging and managing data has pushed the need for techniques and mechanisms that secure information when it flows across the net. *Confidentiality* and *integrity* are two main security properties that must be ensured to data or information in all those distributed cooperative applications, such as collaborative e-commerce [7], distance learning, telemedicine and e-government. Confidentiality means that data can only be accessed by subjects who are authorized by the stated access control policies. Integrity means that data can only be modified by authorized subjects. It is, however, crucial that security be achieved with reasonable performance.

Confidentiality has been widely investigated and several access control mechanisms, specifically tailored to the management of web documents [4, 5, 6], have been proposed. By contrast, the problem of integrity has not been much investigated, even though it is a common requirement in many application environments that not all parties be authorized to modify any data that is exchanged. This is one major limitation of the previous research. Another limitation is that most previous access control mechanisms heavily rely on a server to mediate access to data. We are interested in reducing the server overhead, as it is particularly important for performance; also, it is a basic requirement in some contexts, such as real-time adaptive content delivery or mobile ad-hoc networks.

Several issues need to be addressed to support decentralized and cooperative document updates over the Web. A first requirement, that we investigated in a previous paper [3] is the development of a high level language for the specification of *flow policies*, that is, policies regulating the set of subjects that must receive a document during the update process. Starting from these policies, the server can determine the path that the document must follow. The second previous contribution [1, 2] is the development of an infrastructure and related algorithms to enforce confidentiality and integrity during the process of distributed and collaborative document updates. A major limitation of our previous approach is that it does not exploit possible parallelism that is inherent in data relationships and in the access control policies.

In this paper, we address such limitation. In particular, we propose the use of a protocol that we refer to as security *region-object parallel flow* (S-RPF) graph protocol in the update process. The most innovative feature of S-RPF is that it supports parallel updates on documents, and at the same time enforces confidentiality and integrity require-

*The work of Elisa Bertino and Yunhua Koglin is supported in part by the National Science Foundation under the Project "Collaborative Research: A Comprehensive Policy - Driven Framework For Online Privacy Protection: Integrating IT, Human, Legal and Economic Perspectives", by an IBM Fellowship, and by the sponsors of CERIAS.

†Computer Science Department, Purdue University, West Lafayette, IN, USA, luy@cs.purdue.edu

‡DICO, University of Milano, Via Comelico, 39/41, 20135 Milano, Italy, mella@dico.unimi.it

§CERIAS and CS Department, Purdue University, West Lafayette, IN, USA, bertino@cerias.purdue.edu

¶DSCFM, University of Insubria, Via Valleggio, 11, 22100 Como, Italy, Elena.Ferrari@uninsubria.it

ments. Thus S-RPF ensures a high degree of efficiency. To the best of our knowledge this is the first approach which supports secure and parallel updates of documents.

We cast our protocol in the framework of XML [8] [1] because of the widespread adoption of such a standard in a large variety of application environments. Also, XML organizes data according to hierarchical nested structures thus facilitating the update parallelization. However, the techniques we present in this paper can be easily adapted to other hierarchical document formats.

The remainder of this paper is organized as follows. Section 2 provides some preliminary notions which are needed throughout the paper. Section 3 presents a general overview of our approach. Section 4 presents the server and subject protocols. Section 5 discusses the complexity of S-RPF, and compares it with a centralized system. Finally, Section 6 concludes the paper and outlines future research directions.

2. Preliminaries

2.1. Flow and access control policies

Flow policies explicitly define the order according to which subjects have to receive the document, whereas access control policies specify each subject's privileges over the document. These privileges include update and read. Update privileges allow a subject to modify, insert or delete certain portion(s) of a document. Read privileges allow a subject to browse only certain portion(s) of the document. These portions could be attribute(s), or element(s) of a document, as we will explain later.

In the following, we denote with the term Policy Base (PB) the set of flow and access control policies apply to the set of documents managed by a document server (DS). The flow path of the document among the subjects is denoted as Path = $\langle subject_0, subject_1, \ldots, subject_N, subject_{(N+1)} \rangle$, where $subject_0 = subject_{(N+1)}$ is DS. Thus we assume that the server is always the first and the last subject in the path. A subject can appear more than one time in Path and its privileges over the document may not be the same every time.

To enforce authenticity/integrity, public-key algorithms, such as RSA, are used for digitally signing the documents. We assume that DS knows the public keys of the subjects involved in the update process and that all subjects know the public key of DS. Thus the path a document must follow can also be specified in terms of the public keys of the subjects that must receive the document. More precisely, Path = $\langle pubk_0, pubk_1, \ldots, pubk_N, pubk_{(N+1)} \rangle$ denotes the path that the document must follow, where $pubk_0 = pubk_{(N+1)}$ is the public key of DS, and $pubk_i$ is the public key of the i^{th} subject in the document flow sequence.

[1] Therefore in the following we use the terms data and documents as synonyms.

2.2. Atomic elements and document regions

An XML document [8, 9] is formed by tagged elements. A tagged element may have one or more sub-elements, and one or more attributes. Elements can be nested. Because of this feature, an XML document may be represented according to a graph structure [1] as illustrated by Figure 1.

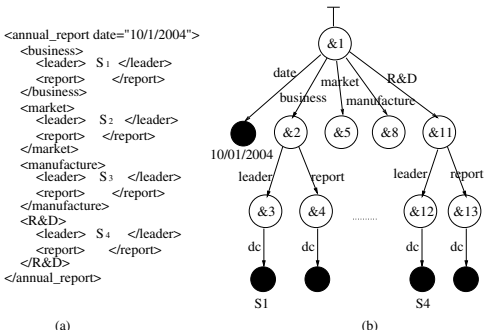

Figure 1. (a) An example of XML document and (b) its corresponding graph representation

An *atomic element* (AE) is either an attribute or the starting and ending tags of an element. An *atomic region* (AR) is a set of atomic elements to which the same access control policies apply. We assume that each region be uniquely identified.

A region can be either *modifiable* or *non-modifiable*. A region is non-modifiable by a subject if this subject can only read it. A region is modifiable by a subject if this subject possesses the authorization to modify it, according to the access control policies.

Based on the above definitions, we introduce the following notations:

Let $D = \{ae_1, ae_2, \ldots, ae_m\}$ be a document to be exchanged, consisting of a set of atomic elements each of them individually identified by an identifier. Document D is partitioned into a set of regions $\{R_1, R_2, \ldots, R_K\}$ such that each region consists of a region identifier (i) assigned by DS and of a set of atomic elements. We denotes a region as $R_i = (i, \{ae_{j_1^i}, ae_{j_2^i}, \ldots, ae_{j_r^i}\}$ where $i \in \{1, \ldots, K\}$ and for any $t \in \{j_1^i, ..., j_r^i\}$, $1 \leq t \leq m$. Atomic elements within the same region are distinct and atomic elements within disjoint regions are distinct.

Each document in our approach has an associated access control information structure (ACIS). Let D be a document, the corresponding ACIS is defined as $\{ar_0, \ldots, ar_N, ar_{(N+1)}\}$ such that:

- ar_i = (*mod, non-mod*)
  Access regions are split into modifiable and non-modifiable regions.
- *mod* $\subseteq \{1, \ldots, K\}$, *non-mod* $\subseteq \{1, \ldots, K\}$
  The modifiable region set and non-modifiable region set are subsets of the entire regions.
- *mod* $\cap$ *non-mod* $= \emptyset$
  If a region is modifiable for a subject, it cannot be in the non-modifiable set of this subject and viceversa.

All regions are considered modifiable by DS.

A *region object* O is an instance of the information in a region. A region object is associated with the region identifier, the subject who authors it, and the time when the subject authors it. Time is not a concern with respect to integrity; so we denote a region object O with a tuple $(r, pubkey)$, where $r \in \{1, \ldots, K\}$ and $pubkey$ is the public key of the subject who generates this region information. If a region R_i is authored by two different subjects, with public key of $pubk_l$ and $pubk_m$, there will be two different region objects, one is $(R_i, pubk_l)$ and another one is $(R_i, pubk_m)$, even though the information in region R_i may be the same. In XML, a region object can be expressed as an element and the tag denotes the region identifier.

All subjects participating in the update process use the same one-way hash function for integrity. When a subject $subj$ updates a region R_i, it generates one-way hash of the region object O_i it has authored. It then encrypts the hash with its private key, thereby signing this region object. The signed hash will flow together with the region object to which it corresponds. When a receiver s checks if O_i is authored by $subj$, s generates a one-way hash of O_i and decrypts the signed hash with $subj$'s public key that s received from DS in the control information. If the signed hash matches the hash value that s generated, the region object O_i is valid.

A package exchanged among subjects contains one or more region objects. Each package starts with sid which denotes that this package is for the receiver who is the i^{th} subject in the Path. Following sid there are region objects. Each region object includes an attribute of $hash$ which is the encrypted hash from the subject who authored this region object.

3. General Overview

The goal of the S-RPF protocol is to efficiently support updates in distributed and cooperative systems, and at the same time, to enforce flow and security policies.

Before starting the update process, DS determines a path P that the document must follow. DS also generates an access control information structure for each subject according to the security and flow policies for each subject (see Figure 2). From P and ACIS, DS constructs a S-RPF graph and then derives the control information (CI) for each subject from the graph. This control information specifies which regions a subject will receive and how the subject can check the integrity of each region object it receives. After DS sends out the control information for each subject, the update process starts.

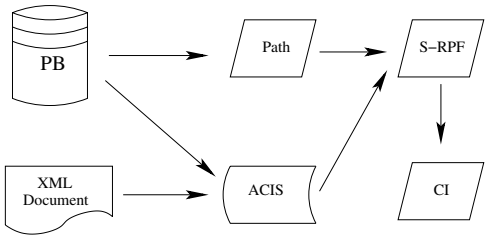

Figure 2. Document pre-procssing

During the update process, each subject decrypts the package it receives; then it uses the control information from DS to check the integrity of and to authenticate the received package. After passing these checks, the subject may execute operation(s) on region(s) of the document over which it possesses privileges. Once the update operations are completed, the subject signs the region object(s) which it is authorized to update with its private key, also in the case in which it does not alter the region information. Finally, the subject enciphers the packages according to the control information and sends them to the next receivers.

4. S-RPF protocols

In this section, we illustrate the two protocols on which our approach relies, that is, the *server protocol*, executed by DS, and the *subject protocol*, which is executed by a subject upon receiving a document package. Before doing that, we state the assumptions on which they rely.

4.1. Assumptions

We make the following assumptions for XML document updates:

- The subjects participating in the updates are *cooperative*. The completion of the update depends on each subject. If one subject cheats more than twice, a receiver will notify DS and DS may broadcast that the updates failed and aborted. A recovery mechanism is detailed in Section 4.6.
- DS has access to the flow policies and to the security policies of the document. The DS is a trusted entity. It determines these policies before the update process

starts. Then these policies are enforced and are not modified during the execution of the update process.

- There is no collusion among the subjects. Each subject does not share information with other subjects.

4.2. Server protocol

The server protocol includes the following steps: (1) construct the S-RPF graph, (2) generate and send each subject its own control information, and (3) send to the first subject(s) the encrypted package(s). In the following, we illustrate all such steps.

4.3. S-RPF construction

S-RPF is a directed graph G (see Figure 3(b)), where each node represents an element in the flow path, and an arc between s_i and s_j denotes that s_j has to access a document region after s_i has accessed it. The arc is labeled with the name of the corresponding region and with the id of the last subject that modifies it.

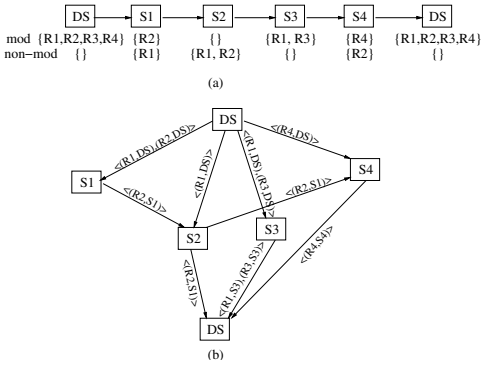

Figure 3. (a) An example of Path and ACIS, (b) the corresponding S-RPF graph

DS builds the S-RPF according to the following rule:

S-RPF Rule: Each region object, which is accessed by a subject that does not author it, flows only once out of the subject who authors it.

This rule enforces the correctness of the protocol (see Section 5.1). The algorithm in Figure 4 is used to construct the S-RPF graph. It aims at maximizing the parallelism of the process enabling the maximum number of subjects to work concurrently. This feature reduces the total amount of time required to accomplish the update process. The algorithm is organized according to the following main phases:

1. - Initialization. A node in the graph represents an element in the flow path (since a subject may appear more than once in the flow path, in the graph, $subject_i$ and $subject_j$ may be the same subject). We also store in each node the necessary information that we will use for generating control information for each node. This step initializes each node's predecessors ($pred$), successors ($succ$) and regions (reg) which this subject is authorized to access. See Figure 6 for the definitions of $pred$ and $succ$.

2. - Labeling regions. For each subject in the graph, this step labels each region that this subject is authorized to access with the public key of the subject who authored this region. We use array Reg_i to store the public key of the last subject that authored region i and a structure ar_i that contains the accessible regions for the i^{th} subject in Path.

3. - Adding edges. The procedure $AddEdges$, reported in Figure 5, updates G by inserting edges for each subject in G, according to Path and ACIS.

4. - Application of the S-RPF rule. If a region object O is to be received later by the subject *subj* who authored it, we remove it from $subj$'s incoming edges. If $subj$ only has read access to O later and needs to send O to another subject *subs*, then the predecessor which is supposed to send O back to *subj* will send O to *subs*.

Procedure *AddEdges* (Figure 5) works according to the following strategy: a subject that has modified a region R sends it to the first subsequent subject s in Path that can access (read or modify) it. If s can only read this region, it forwards the region to all subsequent subjects S in the

Algorithm Construct-RPF
Input: Path, ACIS
Output: $G = (V, E)$

1. for each $i = 0$ to $N + 1$:
   add node $subject_i$ and
   $subject_i.pred = \emptyset$
   $subject_i.suc = \emptyset$
   $subject_i.reg = \emptyset$
2. for each $i = 1$ to K :
   $Reg[i].s = Path.pubk_0$
   for each $i = 1$ to $N + 1$
   $R = ACIS.ar_i$
   for each $r \in R$
   add $(r,\ Reg[r].s)$ to $subject_i.reg$
   if $r \in R.mod$
   $Reg[r].s = subject_i.pubkey$
3. AddEdges(Path, ACIS, G)
4. for each $i = 1$ to N
   $R = ACIS.ar_i$
   for each $(r,\ s) \in subject_i.reg$
   if $s = Path.pubk_i$
   j =delete-pred(i, r)
   delete-succ(j, i, r)
   if $r \in R$*.non-mod*
   for each $su \in subject_i.succ$
   if $r \in su.reg$
   delete-succ(i, su.sid, r)
   t = delete-pred(su.sid,r)
   add-pred(j, r, su.sid)
   add-succ(su.sid, r, j)

Figure 4. Algorithm for S-RPF construction

Procedure AddEdges
Input: Path, ACIS, G
Output: G

1. $\overline{ACIS} = ACIS$
   $AR = \overline{ACIS}.ar_0$
   for each $r \in AR$
      for $j = 1$ to $N + 1$
         if $r \in \overline{ACIS}.ar_j.mod$
            add-pred(0, r, j)
            add-succ(j, r, 0)
            break
         if $r \in \overline{ACIS}.ar_j$*.no-mod*
            add-pred(0, r, j)
            add-succ(j, r, 0)
            $\overline{ACIS}.ar_j$*.non-mod*$= \overline{ACIS}.ar_j$*.non-mod* $\setminus \{r\}$
            continue;
2. for each $i = 1$ to N
      $AR = \overline{ACIS}.ar_i$
2.a   for each $r \in AR.mod$
         for $j = i + 1$ to $N + 1$
            if $r \in (\overline{ACIS}.ar_j.mod \cup \overline{ACIS}.ar_j$*.non-mod)*
               and $Path.pubk_j \neq Path.pubk_i$
               add-pred(i, r, j)
               add-succ(j, r, i)
               break
            if $r \in \overline{ACIS}.ar_j.mod$ and $Path.pubk_j = Path.pubk_i$
               break
            if $r \in \overline{ACIS}.ar_j$*.non-mod* and $Path.pubk_j = Path.pubk_i$
               $\overline{ACIS}.ar_j$*.non-mod* $= \overline{ACIS}.ar_j$*.non-mod* $\setminus \{r\}$
               continue;
2.b   for each $r \in AR$*.non-mod*
         for $j = i + 1$ to $N + 1$
            if $r \in \overline{ACIS}.ar_j.mod$
               add-pred(i, r, j)
               add-succ(j, r, i)
               break
            if $r \in \overline{ACIS}.ar_j$*.non-mod*
               add-pred(i, r, j)
               add-succ(j, r, i)
               $\overline{ACIS}.ar_j$*.non-mod* $= \overline{ACIS}.ar_j$*.non-mod* $\setminus \{r\}$
               continue

Figure 5. Procedure AddEdges

path that can only read R until a subject m is found that can modify R. Also m will receive from s the region. All subjects in S will not send out R to anyone. Thus the subject that has generated a region object cannot distribute different versions of the same region to different subsequent subjects because they have to receive that region object from another subject.

The main phases in the procedure $AddEdges$ are as followed:

1. - Generating the outgoing regions for DS. This phase also adds incoming region for subjects in Path. A region will be received by all the subjects that can only read that region, following DS and preceding the first subject in Path that can modify the region. Also this last subject will receive this region from DS.

2. - Generating the outgoing regions for all subjects. This phase also adds incoming regions for subjects in Path and DS. We analyze, in order, for each subject in Path the following:

2.a - Modifiable regions. A region will be received only by the first subsequent receiver that can access (read or modify) the region. As a subject may appear in Path several times, this receiver must not be the current subject. A region object O will not appear in the flow if the next receiver of O is the subject who authored it and the next receiver has update privilege over it.

2.b - Non-modifiable regions. A region will be received by all the subjects that can only read that region, following the current one and preceding the first subject in Path that can modify the region. Also this last subject will receive this region from the current subject.

If there is no element $p \in subject_x.pred$ such that $p.pid = i$, function *add-pred(i, r, x)* inserts in the set $subject_x.pred$ an element p where: (1) $p.pid = i$, (2) $p.sk = k$ and k is a symmetric key generated by DS (3) $p.reg = \langle t \rangle$ where t is the tuple in $subject_i.reg$ such that $t.r = r$. Otherwise it appends t in $p.reg$.

If there is no element $su \in subject_i.succ$ such that $su.sid = x$, function *add-succ(x, r, i)* inserts in the set $subject_i.succ$ an element su where: (1) $su.sid = x$, (2) $su.sk = k$ and $k = subject_x.pred.p.sk$, (3) $su.reg = \langle r \rangle$. Otherwise it appends r in $su.reg$.

delete-pred(i, r) function deletes r from $p.reg$ such that $p \in subject_i.pred$ and $r \in p.reg$, and returns an index $p.pid$. $subject_i$ will not expect to receive a region r from its predecessor $subject_{p.pid}$. If $p.reg = \emptyset$, then delete p from $subject_i.pred$.

delete-succ(j, i, r) function deletes r from $su.reg$ such that (1) $su \in subject_j.succ$, (2) $su.sid = i$, (3) $r \in su.reg$. $subject_j$ will not send region r to its successor $subject_i$. If $su.reg = \emptyset$, then delete su from $subject_j.succ$.

So it is possible that different subsets of all non-modifiable regions are sent to different subjects, and the same region object can be sent to different receivers by the same subject. According to the algorithm for the construction of S-RPF, a given region of the document cannot be updated by more than one subject at a time.

From above, the S-RPF graph that DS generated has the following properties:

- If no subject has access rights to a region R, then no region object O such that $O.r = R$ will appear in the flow of the S-RPF graph.
- If a region object is modified by a subject *subj*, then this region object will not flow out from *subj* and a new region object will start at *subj*.
- A region object may have several copies flowing in the graph at the same time.

- No region object flows back to the subject who authored it.
- If no subject has update rights on a region R, but at least one subject has access to this region, then a region object O, such that $O.r = R$, will start its flow at DS and its author will be DS.

From above, we can easily derive the following property:

Property 1: The flow of each region object among the subjects in the update process is acyclic.

Based on this feature, the S-RPF protocol could allow any static update policy. For example, during the update process a region can be modified more than once by a subject, or a region could be updated by a subject, and later on, read by the subject. Even though the original path may contain cycles among all subjects, based on the algorithm we presented in this paper, each region object flows among all subjects in an acylic way.

4.4. Control information

The Control Information (CI) contains, for each subject in the path, the corresponding incoming package templates and outgoing package templates. Figure 6 details the structure of CI.

An incoming package template contains the symmetric key for the receiver to decrypt an incoming package; it also includes the sequence of regions the incoming package will contain, and for each region the public key of the last subject who authored this region. The goal of an incoming package template is to help a receiver to verify that the package it receives is from a specified sender and to verify that the content of the package is correct up to that point. Different subjects will receive different incoming templates from DS. An outgoing package template includes the symmetric key for the sender to encrypt the package and the sequence of regions to be sent in this package, so the sender can organize a package for its successor with the correct content.

After building the S-RPF graph G, it is easy for DS to generate control information for each subject. DS just copies $G.subject_i.pred$ and $G.subject_i.succ$ to $CI_i.pred$ and $CI_i.succ$, then sends to each subject its control information.

Example 1 *Suppose that S_5 receives R_1, R_2, R_3, R_4 from S_1, S_2, S_3, and S_4, respectively and that R_1, R_2, R_3, R_4 are updated by S_1, S_2, S_3, and S_4, respectively (Figure 7). The instructions from DS to S_5 are: to read R_1 and send it to DS (no one will access R_1 anymore), to form a new package which consists of three regions, R_2, R_3 and R_4 and to send it to S_6. If Path $=\langle pubk_0, pubk_1, pubk_2, pubk_3, pubk_4, pubk_5, pubk_6, pubk_7\rangle$, where $pubk_0$ and $pubk_7$ is the public key of DS and $pubk_i$ is the public key of S_i, then the control information for S5 will be expressed as following:*

$CI_5 = (5,\ pred,\ succ)$ *where*

$CI = \{CI_0, CI_1, \ldots, CI_N, CI_{(N+1)}\}$ and
$CI_i = (i, pred, succ)$ is the control information generated for i^{th} subject in Path
$pred = \{p_{P_1}, \ldots, p_{P_i}\}$: set of incoming package templates
$p_x = (pid,\ sk_{xi},\ reg)$: an incoming template from x^{th} subject in Path, where
1. $pid = x$ and $x \in \{P_1, \ldots, P_i\}$
   $reg = \langle rs_1, \ldots, rs_{H(x)}\rangle$
   $rs_j = (r,\ s),\ j \in \{1, \ldots,\ H_{(x)}\}$
   $r \in \{1, \ldots, K\}$, s is the public key of the last P-proxy that modified r
   pid is the sender's position generated according to Path
   sk_{xi} is the symmetric key for encrypting/decrypting the package sending from $subj_x$ to $subj_i$, where $subj_t$ is the t^{th} subject in Path
2. $\forall j, w \in \{1, \ldots, H_{(x)}\}$: $j \neq w \Rightarrow rs_j.r \neq rs_w.r$
   a region must appear only once in the sequence of regions from a predecessor.
3. $\forall j, q \in \{1, \ldots, P_{(i)}\}$: $j \neq q \Rightarrow sk_{ji} \neq sk_{qi}$
   component $pred$ contains distinct predecessor subjects
4. $\forall j, q \in \{1, \ldots, P_{(i)}\}, j \neq q, x \in \{1, \ldots, H_{(j)}\}$, $y \in \{1, \ldots, H_{(q)}\} : p_j.rs_x.r \neq p_q.rs_y.r$
   an accessible region must be received only from one predecessor.

$succ = \{su_{S1}, \ldots, su_{Si}\}$: set of outgoing package templates
$su_y = (sid,\ sk_{iy},\ reg)$ this is an outgoing template, where
1. $sid = y$ and $y \in \{S_1, \ldots, S_i\}$
   sid is the position of the receiver of this package according to Path.
   sk_{iy} is the symmetric key as defined before
   $reg = \langle r_1, \ldots, r_{W(y)}\rangle$: sequence of regions sent to successor who is at the y^{th} position in Path.
   $r_f \in \{1, \ldots, K\}, f \in \{1, \ldots, W_{(y)}\}$
   $\forall j, g \in \{1, \ldots, W_{(y)}\}$: $j \neq g \Rightarrow r_j \neq r_g$
   A region must appear only once in the sequence of region objects to be sent to a successor.
2. $\forall j, x \in \{S_1, \ldots, S_i\}$: $j \neq x \Rightarrow su_j.sk_{ij} \neq su_x.sk_{ix}$
   successors are distinct.

Figure 6. Control information specification

- $pred = \{(1, sk_{15}, < (1, pubk_1) >),\ (2, sk_{25}, < (2, pubk_2) >),\ (3, sk_{35}, < (3, pubk_3) >),\ (4, sk_{45}, < (4, pubk_4 >)\}$
- $succ = \{(7, sk_{57}, < 1 >),\ (6, sk_{56}, < 2, 3, 4 >)\}$.

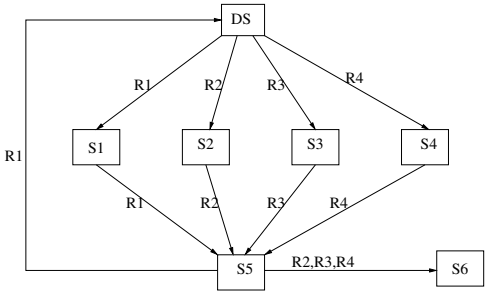

Figure 7. Generating control information for S5

Control information is signed by DS and enciphered with the recipient's public key so that only the designated subject can see the information. The designated subject can

verify that the message is from DS. Control information exchange could also be performed by opening an SSL session in which a symmetric session key is generated and used during the communication. Thus a secure channel is built between a subject and DS.

4.5. Subject protocol

During document updates, each subject executes the following steps: (i) it performs integrity check according to incoming package templates received from DS; (ii) it executes operations on the document according to its privileges; (iii) it forms packages according to outgoing package templates received from DS, and sends out these packages. We detail these steps in the following:

1. Upon receiving a package P, the receiver by using the control information CI_i, verifies (1) if there has been any transmission error; if there is any error, asks the sender to send the document again; (2) that the package has been sent by one of its predecessors. Suppose the receiver deciphers P with the symmetric key k such that $k = p_x.sk$ and $p_x \in CI_i.pred$. If $P.sid \neq CI_i.id$, the package is discarded. (3) the integrity and authorization of each region according to the incoming package template. For each R in $p_x.reg$, the receiver checks if the region object in the package starts with a region identifier equal to $R.r$. If so, the receiver generates a hash value using one-way hash function, deciphers the hash in the package with $R.s$ and checks if these two values are equal. If there is any error, it asks the sender to recover
2. The receiver performs operations on the document according to its privileges. After correctly receiving a package from each predecessor, the receiver executes its privileges on the documents. If it has update privileges on some regions, it updates the regions, calculates the hash value for each region it updated, and ciphers this value with its private key for future authorization checking.
3. The receiver generates the new package(s). For each $su \in CI_i.succ$, the receiver forms an outgoing package U such that $U.sid = su.sid$. For each $r \in su.reg$, fills *hash* and *region object* in U. After this, the subject encrypts U with $su.sk$ and sends it to the sid^{th} subject. The receiver should also keep a copy for later recovery.

4.6. Recovery protocol

If a subject receives a package which fails the verification, the subject asks the sender to recover the package. If a receiver cannot get an error-free package according to the control information twice, it will send both packages it believes are incorrect to DS and the sender.

DS then first checks if the malicious sender m of the erroneous region has only read access to this region. If not, DS decides to abort the update, because we assume that the completion of update depends on each subject correctly updating their corresponding regions. If m only has read access, DS asks all the receivers who received this region from m. If any one has a correct version, DS sends this correct version to all the senders who did not receive a corrected version from m. If no one has a correct version, DS asks the subject who authored this region to send DS a copy, DS then acts in the role of m, checking the integrity and sending to all the receivers to whom m was supposed to send this region.

5. Analysis and discussions

5.1. Correctness analysis

From Section 4.3, we can conclude that the S-RPF built by DS enforces flow policies related to an XML document. If subject S_a updates region R before subject S_b in the flow policy, the flow of R in the S-RPF built by DS will also have this order. Moreover if a subject S_c reads region R after S_a has modified it, then this order is preserved in S-RPF.

Theorem 1 *Protocol S-RPF is secure with respect to integrity.*

Proof: We need to prove that a subject m cannot update a region over which it does not have update privilege. There are two cases.

(1) m modifies a region object which is not authored by itself. In this case, integrity is enforced in the protocol by digital signature. If a region R is modified by a subject i, i will sign the hash that it calculated from R with its private key. If a subject j has read privileges on R, j will receive control information from DS, which contains an incoming template. The incoming template includes the public key of i for deciphering the hash. j will calculate the hash of the region and check the signature. m cannot modify region R before it reaches j, since m does not know i's private key.

If m receives two region objects authored by the same subject i, it cannot switch the information in these two regions. As a region object represented in XML has a region identifier in its tag.

Thus no subject can modify a region object which has not been authored by itself.

(2) A subject modifies a region object authored by itself, even though it does not has update privilege over it later. This is avoided by the S-RPF rule. Suppose region R_1 is updated by A, then flows to B for read, and then back to

A for read (A cannot update R_1 this time) and then ends at C for read. In this case, A could not send to C a region object which is different from the one it sends to B. In S-RPF graph, B will send a copy to C instead of A. S-RPF ensures that C receive the region object that A authored at the beginning. Thus the integrity of the whole document is enforced. □

Theorem 2 *Protocol S-RPF is secure with respect to confidentiality.*

Proof: We need to prove that if a subject not authorized to access a region, it can not read it. This is enforced by the use of symmetric keys to encipher/decipher a package that only designated receiver can see it. When a subject receives a package, it can use the received control information from DS to decipher the package. If a subject does not have such information, it cannot decipher the package. S-RPF generation ensures that a subject only receives the parts of the document which it is authorized to access. Thus S-RPF is secure with respect to confidentiality. □

We now discuss the amount of information which could be revealed and check if confidentiality and integrity are violated. With this approach, a receiver could partially know the access rights of its predecessor(s) or successor(s). In Example 1, S_6 knows that S_5 has access rights to at least R_2, R_3 and R_4. S_5 knows that S_6 has access rights to at least R_2, R_3 and R_4. Other than that, no other information can be derived. This will not violate confidentiality and integrity as defined previously, because these definitions concentrate on the contents of a document.

5.2. Complexity analysis

We now analyze the complexity with respect to temporal complexity and communication complexity. The latter is evaluated in terms of number of exchanged messages. We also compare our approach against a centralized approach.

In particular, under a centralized approach, DS sends each subject in Path a package containing only the contents of a document to which the subject has access privileges. After executing operations on it, the subject sends back to DS only the parts that it has updated. When DS correctly receives it, that is, there are no transmission errors, DS sends another package to the next subject in Path. Otherwise, DS sends the subject the package again and asks for recovery. A centralized system accomplishes the same function as our protocol. It also uses symmetric key to allow DS securely communicate with each subject. However, in a centralized approach, no hash function is needed and subjects do not need to sign the region objects they authored, since DS communicates with each subject securely and knows each subject's access control information structure.

There are two types of errors that require a recovery. They are as following:

1. Subject-will-recover error: This includes transmission errors, and any other errors occuring in a centralized approach that require DS to ask a subject recovery.
2. Malicious-subject-intentional error: A malicious subject illegally modifies a region object and refuses to send the correct version to a receiver.

Only the first type of errors can happen in the centralized approach. In S-RPF, DS is needed for the second type error recovery.

In the following analysis, all communications before the start of the updates are ignored. As in a centralized system, DS also needs to communicate with all subjects to set up secure communication channels before starting the update process. In order to simplify our analysis, we will not consider the size of hash value in a package.

We compare the following cases for communication cost:

1. No recovery:

   In this case, the total number of packages PK and total size of messages M are as following:

   - for the centralized approach
     - $PK = 2N$
     - $M = \sum_{i=1}^{i=N} A_i + \sum_{i=1}^{i=N} U_i$
   - for S-RPF protocol
     - $PK = \sum_{i=1}^{i=N} Pr_{s_i} + Pr_{DS}$
     - $M = \sum_{i=1}^{i=N} A_i + u$

   Where:

   N is the number of elements in the path, not including DS;

   A_i is the size of the package that DS sends to $subject_i$ in centralized approach;

   U_i is the size of the package $subject_i$ sends back to DS. This package only contains updated regions by $subject_i$.

   Pr_i is the number of predecessors of $subject_i$ in S-RPF graph.

   u is the sum of the size of packages DS received in S-RPF graph. ($u \leq \sum_{i=1}^{i=N} U_i$)

2. Recovery:

   If the recovery has to be executed because of the first type of error, the extra packages caused by the recovery in the centralized system is equal to that in S-RPF. As in S-RPF, a receiver will act the same as DS in the centralized system, asking the sender to recover.

If the recovery has to be executed because of the second type of error, S-RPF will incur extra cost which will not appear in the centralized approach. In this case, DS in S-RPF may need to ask up to $N-2$ subjects for a correct version.

From above, we can see that when all subjects are cooperative and a region is updated often (for example, Case B in Figure 8), S-RPF reduces the number of packages (in case B, only $N+1$ packages) and the total size of messages (in case B, $\sum_{i=1}^{i=N} A_i + U_N$). However, S-RPF could also possible generate $O(N^2)$ packages. The total number of packages in S-RPF is equal to the number of edges in S-RPF graph. In congested networks and uncooperative systems, S-RPF may not perform better than the centralized approach.

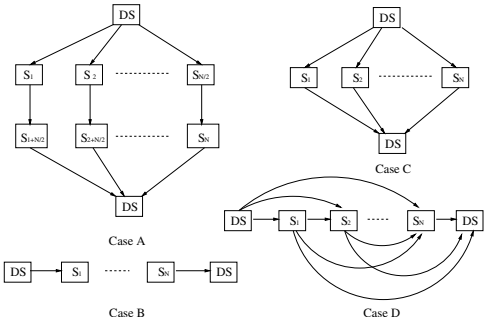

Figure 8. Case study for the total time to complete the update

Next, we analyze the efficiency of the protocol by comparing the time needed to complete the update. The parameters we used in analysis are listed in Table 1. The total time needed to complete the update is formulated as $T \leq \sum_{i=1}^{i=7} T_i$.

1. No recovery. We can easily estimate the time for the centralized system. For the S-RPF, the time varies. We study the cases in Figure 8 which represent a high-level parallel updates (Case A and Case C) and low-level parallel updates (Case B). Table 2 reports the time complexity.

   From Table 2, we can see that when $N > 1$ the S-RPF for Case B takes more time than Case C:

   $$T_B - T_C = (N-1)(h+U)$$

   Also, Case B takes more time than Case A:

   $$T_B - T_A = \frac{(N-2)(D+E+H+2h+2U)}{2}$$

   The best time for S-RPF to complete the update is hard to find. For example, $T_C - T_A = \frac{(N-2)(D+E+H)}{2} - (h+U)$. If the average time for an object executing operation takes longer time than the time of $\frac{N(D+E+H)}{2}$, then Case C is better than Case A. If N is large and the average time for a subject finishing operations is fast, then Case A can be better than Case C.

Table 1. Notations for efficiency analysis

DS	T_1	Total time for deciphering and enciphering packages
	T_2	Total time for calculating hash values and encrypting them
	T_3	Total time for integrity check of received packages
Subjects	T_4	Total time for deciphering and enciphering packages
	T_5	Total time for integrity checking
	T_6	Total time for executing operations (read, update)
	T_7	Total time for calculating hash values and encrypting them
N		Number of subjects in the path, not including DS
E		Average time for enciphering a package
D		Average time for deciphering a package
H		Average time for checking integrity of regions in a received package
U		Average time for a subject in Path executing operations(read/update)
h		Average time for a subject or DS in Path calculating the hash values for the region objects that it authored and encrypting them

   The worst case for S-RPF is when DS sends a package to each subject and each subject sends a package to everyone following it (Case D). However, the time to complete the update is far less than $\sum_{i=1}^{i=7} T_i$ in Table 2. As S_1 is deciphering the package and executing integrity checking, all other subjects following it will also check integrity of the packages they received from DS. So the worst time of the S-PRF is

   $$T \leq \frac{E \times N^2 + 3N \times E}{2} + (N+1)(D+h+H) + N \times U$$

   The time difference between the centralized approach with the S-RPF Case B is $N(D+E-h-H) - (D+E+H+h)$. Since $D \approx E$ and $h \approx H$ in Case B, it can be simplified as $N(2D-2H) - 2D - 2H$. This means that, if deciphering a package takes similar time as integrity checking a package, then the centralized approach has similar time as S-RPF case B.

   Next we compare a centralized approach with S-RPF Case A, where subjects can execute parallel operations on the document. Since normally $D \approx E$ and $h \leq H$:

   $$3N \times D + N \times U \geq \frac{N \times H}{2} + 4D + 3h + 2U + 2H$$

   $$\Longrightarrow \; U \geq \frac{H}{2} - 3D + \frac{6H - 2D}{N-2}$$

   Under the situation that $D \geq \frac{2H}{N-2}$, when the average time for a subject executing operation is longer than half time of integrity checking, then S-RPF in this case

Table 2. Time analysis in the case of no recovery

	Centralized approach	S-RPF (Fig 8) Case A	Case B	Case C	Case D
T_1	$N \times (D+E)$	$\frac{N \times (D+E)}{2}$	$D+E$	$N \times (D+E)$	$N \times (D+E)$
T_2	0	h	h	h	h
T_3	0	$\frac{N \times H}{2}$	H	$N \times H$	$N \times H$
T_4	$N \times (D+E)$	$2(D+E)$	$N \times (D+E)$	$D+E$	$\frac{N \times (1+N)(D+E)}{2}$
T_5	0	$2H$	$N \times H$	H	$\frac{N \times (N+1)H}{2}$
T_6	$N \times U$	$2U$	$N \times U$	U	$N \times U$
T_7	0	$2h$	$N \times h$	h	$N \times h$

requires less time to complete update than the centralized approach.

2. Recovery: If a recovery has to be executed because of the first type of error, for the centralized system, the extra time is $2D + 2E$; for n recoveries, the extra time increases linearly, that is, $n(2E + 2D)$. For the S-RPF protocol, the extra time varies. It depends on the S-RPF graph computed by DS and the location of the recovery. It may even not increase the total time due to the parallel operations among all participants.

   If the recovery has to be executed because of the second type of error, no extra time is required for the centralized approach. For S-RPF, the additional incurred time varies. If the number of subjects involved in the recovery is very small, then the overall completion time may not increase. If many subjects are involved in the recovery, the extra time may increase substantially.

Since encipher and decipher operations can be very fast, while human interactions are in most cases involved in the update, S-RPF can complete the update faster than centralized approach if subjects are cooperative. When more subjects are involved, even if $U \leq H$, S-RPF could be still more efficient than centralized systems.

6. Conclusion and future work

In this paper, we have proposed a protocol for distributed document update in cooperative systems.The protocol enforces both flow and security policies of a document and simultaneous updates on different parts of a document can be executed. In a cooperative system, when several subjects update a large document, S-RPF can reduce the time to complete the update, especially when human beings are involved in update process. If the recovery is not due to malicious subjects, the frequency of recovery to be executed by DS is low. However, if a malicious subject is detected, the recovery can be expensive.

Flow policies and access control policies can be static or dynamic. Subjects involved in static flow policies will not change and their order of receiving a document is prefixed. In static access control policies, each subject's privilege over a document will not change during the update process. By contrast, in dynamic flow policies, a subject may join in or drop out of during the update. The privilege of a subject over a document may also change. This protocol applies to static flow and access control policies. It can also be extended to certain dynamic security policies; however, due to space limits, we do not detail such extensions here. Future work includes to test our protocol's performance in real systems.

References

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

[2] E. Bertino, G. Correndo, E. Ferrari and G. Mella, "An Infrastructure for Managing Secure Update Operations on XML Data", in SACMAT'03, Como, Italy, 2003.

[3] E. Bertino, E. Ferrari and G. Mella, "An XML-based Approach to Document Flow Verification", in Proceedings of 7th International Conference on Information Security(ISC04), Palo Alto, CA, USA, 2004.

[4] C. Pollmann. The XML Security Page. Available at http://www.nue.et-inf.uni-siegen.de/ geuerpollmann/xml_security.html.

[5] W. Fan, C. Chan and M. Garofalakis, "Secure XML Querying with Security Views", in SIGMOD 2004, Paris, France, 2004.

[6] G. Miklau and D. Suciu, "Controlling Access to Published Data Using Cryptography", in Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003.

[7] B. Thuraisingham, A. Gupta, E. Bertino and E. Ferrari, "Collaborative Commerce and Knowledge Management", in Knowledge and Process Management, 9(1):43-53(2002).

[8] Extensible Markup Language (XML). Available at: http://www.w3.org/XML/.

[9] W3C XML Schema. Available at: http://www.w3.org/XML/Schema.