SECURITY MECHANISMS FOR CONTENT DISTRIBUTION NETWORKS

by Yunhua Koglin

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

SECURITY MECHANISMS FOR CONTENT DISTRIBUTION NETWORKS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Yunhua Koglin

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2006

Purdue University

West Lafayette, Indiana

To my Dad and my Mom

No words in the world can express how much I thank you

for your love and your support

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Professor Elisa Bertino, who has been an invaluable teacher during my Ph.D. study at Purdue University. Her insightful guidance and advice have made substantial differences in my work. I cannot thank her enough for all her help and support.

I would also like to thank the other members of my committee, Professors Arif Ghafoor, Sunil Prabhakar and Ninghui Li for their time and support. Most especially, I would like to thank Professor Sonia Fahmy for reading my dissertation and providing valuable feedback.

It has been my greatest pleasure to work along with Professor Elena Ferrari and Dr. Giovanni Mella during my Ph.D. study. Their valuable advice and suggestions have helped me make great improvements in my research. I am especially impressed by their detail-oriented attitude toward research and I greatly appreciate their help.

Most of all, I would like to thank my beloved Dad, Mom, my sisters, brothers, and especially my husband Donald, for their great support and encouragement. Without their love and support, this dissertation would not exist.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Koglin, Yunhua Ph.D., Purdue University, December, 2006. Security Mechanisms for Content Distribution Networks. Major Professor: Elisa Bertino.

Securing data is becoming a crucial need for most internet-based applications. In this research, we investigate security mechanisms for content distribution networks.

We address the problem of how to ensure that data, when moving among different parties, are modified only according to the stated policies. We cast our solution in 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, allows different users to simultaneously update different portions of the same document, according to the specified access control policies. It ensures data *confidentiality* and *integrity*. 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.

We then extend our document update application into Byzantine and failure prone systems by removing the trusted party which is responsible for recovery of the document. We have developed an approach which uses a group of delegates for recovering documents. Many optimizations have been provided.

We improve previous solutions by proposing a scalable distributed protocol which uses cryptographic techniques to provide dynamic group communications, participating anonymity and completeness, and privacy on access privileges.

Other security problems such as *confidentiality* and *availability* are also investigated in the application of content-based publish/subscribe (pub/sub) systems. We propose a hierarchical event forwarding scheme which increases system availability by tolerating some broker failures. Our approach can efficiently determine the subscription groups to which an event must be delivered by exploiting locality. Moreover, we propose an efficient encryption scheme, under which a broker encrypts an event only once. The encryption key can be efficiently derived by subscribers, even though they may belong to different subscription groups.

1 INTRODUCTION

Content distribution networks (CDNs) are all those applications which support data dissemination, searching and retrieval. With the wide spread use of the internet, CDNs have been studied extensively [1–21]. Most previous research focuses on enhancing the performance of CDNs by replication. Different mechanisms (such as [22–26]) are used to deploy content replication on *trusted* cache proxies scattered around the Internet. When receiving a client request, instead of asking a content server for the requested contents, a proxy first checks if these contents are locally cached. Only when the requested contents are not cached or out of date are the contents transferred from the content server to the clients. If there is a cache hit, network bandwidth consumption can be reduced. A cache hit also reduces access latency for clients. System performance therefore improves, especially when large amounts of data are involved. Besides these improvements, caching makes the system robust by letting caching proxies to provide content distribution services when the server is down or the network is congested.

Secure content distribution has recently received more attention from both academia and industry than before, due to the increasing emphasis on security in many applications. Ensuring content security in distributed environments is challenging. For example, content may be easily modified or accessed when they are transmitted across the internet; a compromised replica may violate access control of content, or damage integrity by maliciously modifying the content. Moreover, with the emerging of various network appliances and heterogeneous client environments, there are other relevant new requirements for content services by intermediaries [6, 7] which make security enforcement difficult. For example, content from the server may need to be transformed in order to adapt to the requirements of a client's security policy, device capabilities, preferences and so forth. Therefore, several *content services* have been identified that include, but are not limited to: content trans-coding [6–8, 13], in which data is transformed from one format into another, data filtering, and value-added services, such as watermarking [10]. Other relevant services are related to personalization, according to which special-purpose proxies can tailor the contents based on user preferences,current activities, and past access history.

The 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 [11], 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.

Even though several access control mechanisms, specifically tailored to the management of web documents [1, 5, 19, 27–34], have been proposed, 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.

In this research, we address several issues to support decentralized and cooperative data modification over the Web by casting our application in XML document updates. A first requirement, that is investigated in [35] 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. The second is the development of an infrastructure and related algorithms to enforce confidentiality and integrity during the process of distributed and collaborative document updates [36,37].

Other issues such as *availability* are also important. We investigate availability problem in the context of content-based publish/subscribe (pub/sub) systems. Previous research on content-based pub/sub systems mainly focuses on the efficient matching performed by brokers. However, security issues have seldom been addressed, even though they are the basic requirements for some applications. For example, when a subscriber has registered by paying a fee for receiving events that satisfy its subscription, a broker failure should not prevent the subscriber from receiving these events on time. Moreover, other users who did not subscribe to these events should not have access to them, thus confidentiality is also need to be addressed.

1.1 Objectives of this work

The main objectives of this work are:

- to devise mechanisms to support data confidentiality and integrity, especially when data is moving around several parties according to the flow policy.
- to devise mechanisms to support data availability, cast in the application of content-based publish/subscribe systems.
- to devise mechanisms to support fault tolerance in content distribution networks.

The remainder of this dissertation is structured as follows. In the next chapter, we present a protocol which support parallel and distributed secure updates to XML documents. Chapter 3 extends our work by removing the trusted server which is responsible for recovering during the update process. We then further extend the work in Chapter 4 by proposing an protocol which uses cryptographic techniques to provide dynamic group communications, participating anonymity and completeness, and privacy on access privileges. Finally, we present our result in content-based publish/subscribe systems in Chapter 5. We do not include a separate section on related work, as such work is discussed in the chapters to which they are most appropriate.

2 AN UPDATE PROTOCOL FOR XML DOCUMENTS IN DISTRIBUTED AND COOPERATIVE SYSTEMS

In this chapter, 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.

We cast our protocol in the framework of XML [38] [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 here can be easily adapted to other hierarchical document formats.

2.1 Preliminaries

2.1.1 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

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

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.

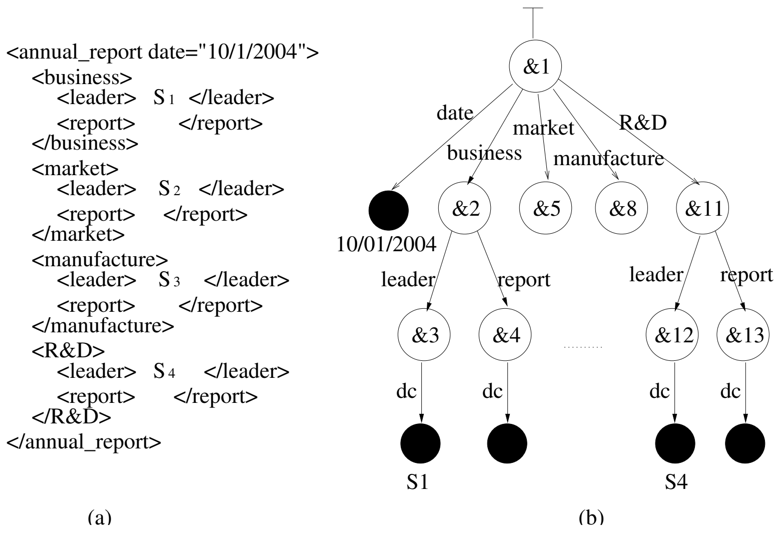
To enforce authenticity/integrity, public-key algorithms, such as RSA, are used for digitally signing the documents. We assume that DS knows the public keys of the subjects involved in the update process and that all subjects know the public key of DS. Thus the path a document must follow can also be 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.

2.1.2 Atomic elements and document regions

An XML document [38, 39] 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 [36] as illustrated by Figure 2.1.

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



```
<annual_report date="10/1/2004">
   <business>
      <leader>  S1  </leader>
      <report>      </report>
   </business>
   <market>
      <leader>  S2  </leader>
      <report>    </report>
   </market>
   <manufacture>
      <leader>  S3  </leader>
      <report>      </report>
   </manufacture>
   <R&D>
      <leader>  S4   </leader>
      <report>     </report>
   </R&D>
</annual_report>
```

Figure 2.1. (a) An example of XML document and (b) its corresponding graph representation

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:

- $ar_i = (mod,\ non\text{-}mod)$

  Access regions are split into modifiable and non-modifiable regions.

- $mod \subseteq \{1, \ldots, K\}$, $non\text{-}mod \subseteq \{1, \ldots, K\}$

  The modifiable region set and non-modifiable region set are subsets of the entire regions.

- $mod \cap non\text{-}mod = \emptyset$

  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.

2.2 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 2.2). 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.

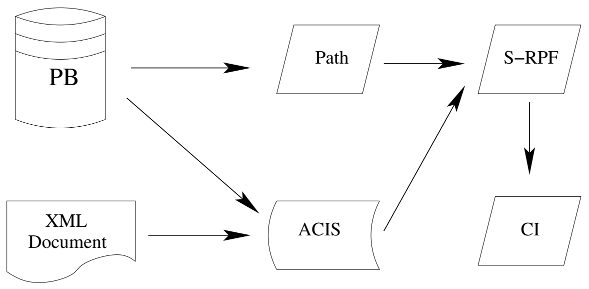


Figure 2.2. Document pre-procssing

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.

2.3 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.

2.3.1 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 2.3.6.
- 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.

2.3.2 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.

2.3.3 S-RPF construction

S-RPF is a directed graph G (see Figure 2.3(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.

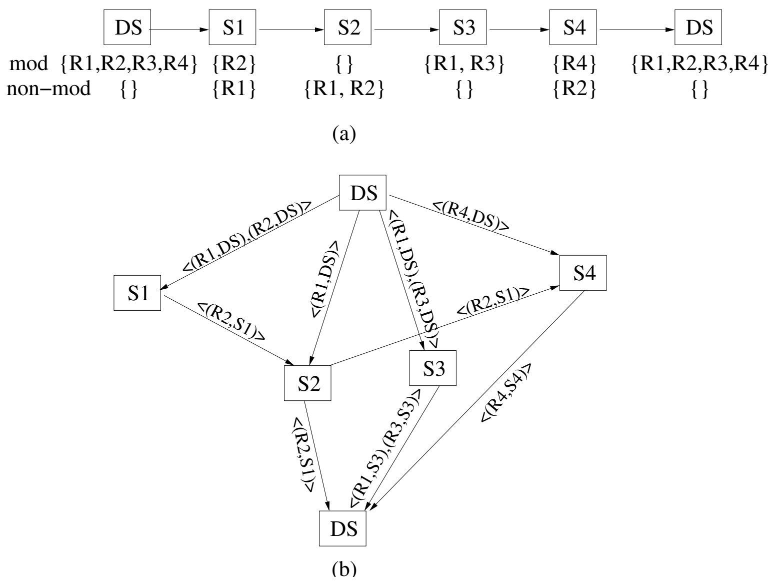


Figure 2.3. (a) An example of Path and ACIS, (b) the corresponding S-RPF graph

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 2.4.1). The algorithm in Figure 2.4 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

```
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 $r \in R.non\text{-}mod$
                 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 2.4. Algorithm for S-RPF construction

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 2.6 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

```
**Procedure** AddEdges
**Input:** Path, ACIS, $G$
**Output:** $G$

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

Figure 2.5. Procedure AddEdges

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 2.5, 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 2.5) 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 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 chapter, each region object flows among all subjects in an acylic way.

2.3.4 Control information

The Control Information (CI) contains, for each subject in the path, the corresponding incoming package templates and outgoing package templates. Figure 2.6 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 2.7). 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*

$CI = \{CI_0, CI_1, \ldots, CI_N, CI_{(N+1)}\}$ 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 2.6. Control information specification

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

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

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

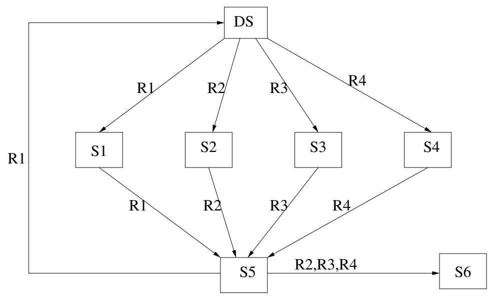


Figure 2.7. Generating control information for S5

and used during the communication. Thus a secure channel is built between a subject and DS.

2.3.5 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.

2.3.6 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.

2.4 Analysis and discussions

2.4.1 Correctness analysis

From Section 2.3.3, 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 2.4.1 *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. $\square$

Theorem 2.4.2 *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. $\square$

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.

2.4.2 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.

Next, we present the results for communication cost. We compare two cases: no recovery and recovery.

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 2.8), 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.

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 2.1. The total time needed to complete the update is formulated as $T \leq \sum_{i=1}^{i=7} T_i$.

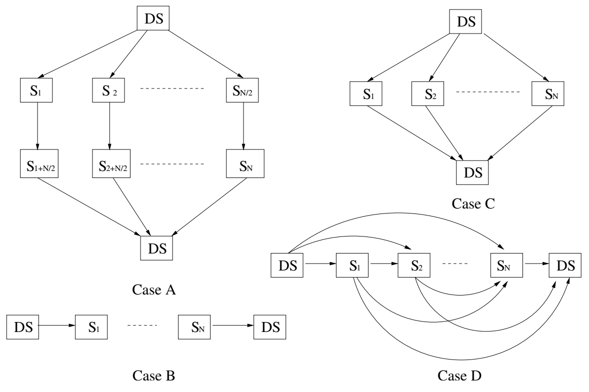


Figure 2.8. Case study for the total time to complete the update

Table 2.1
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

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 2.8 which represent a high-level parallel updates (Case A and Case C) and low-level parallel updates (Case B). Table 2.2 reports the time complexity.

   From Table 2.2, we can see that when $N > 1$ the S-RPF for Case B takes more time than Case C:

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

Table 2.2
Time analysis in the case of no recovery

	Centralized approach	S-RPF (Fig 2.8)			
		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	$N \times h$

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.

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 2.2. 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 $h \leq H$:

$$3N \times D + N \times U \geq \frac{N \times H}{2} + 4D + 3h + 2U + 2H$$
$$\implies \quad U \geq \frac{H}{2} - 3D + \frac{6H - 2D}{N - 2}$$

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 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.

2.5 Conclusion and future work

In this chapter, 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 pre-fixed. 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.

3 XML DOCUMENT UPDATES IN BYZANTINE AND FAILURE-PRONE DISTRIBUTED SYSTEMS

In this chapter, we present an approach for the cooperative updates of XML documents particularly suited for Byzantine and failure-prone distributed systems. With the term *Byzantine*, we mean a party involved in the process that does not obey the defined protocols. The proposed approach is based on a specific infrastructure that extends the one already presented in [36]. Here we introduce the possibility of specifying the path that a document must follow and of *modifying* it during the update process. We support such feature through flow paths and related policies for its specification and modification. The most important feature of the proposed system is that recovery is fully distributed as the last correct version of a corrupted document is cooperatively built by a set of subjects, called *delegates*, instead of by one trusted party. In the previous approach ([36]), the *DO* is responsible for the recovery. However, there are many situations under which the *DO* cannot do this. For example, under the principle of separation of duty, the *DO* is not allowed to access the document until it finished. Furthermore, we allow some Byzantine and failure prone delegates exist. The approach we proposed in this chapter achieves the security requirements as previous chapter, while at the same time, meets all these constraints.

3.1 Motivating example

In this section, we provide an example that motivates the need for our infrastructure, cast in the domain of pharmaceutical surveys. In such surveys, doctors from different hospitals are asked to give feedback on several drugs used for common diseases. These drugs are manufactured by different companies. As the survey results will be made public later, these companies would like to see that the feedback favors their medicines.

Participating hospitals are chosen based on an agreement with these companies. The survey document should be circulated among each of the participating hospitals. The order of circulation is fixed before the survey starts. Doctors of a participating hospital will answer the questions asked in the survey document when they receive it. Doctors may also extend the circulation path by adding their nurses, who may update certain sub-sections of the survey for their doctors. Nurses however are not permitted to alter other sub-sections. Only doctors are allowed to extend the circulation path; nurses should not do so.

In order to ensure that the survey is processed correctly, parties such as public notaries are required. These parties are chosen based on the agreement with the companies. If the survey document is corrupted by a malicious participant (for example, in order to favor a certain company, a doctor/nurse may overwrite information on the survey without authorization), the notaries are responsible for recovering the uncorrupted document. We would like to choose the fewest number of notaries possible. However, it is extremely difficult to make all companies to believe that one trusted notary exists which will execute the recovery correctly. In fact, it is possible for a notary to damage the integrity of the survey quite easily. For example, if a doctor extends the document circulation path by letting his nurse fill in parts of it, and if the nurse fills information which does not favor a particular company which patronizes the notary, the notary could delete the information filled by the nurse, as if the doctor did not extend the survey path. Therefore, it is difficult to have a single trusted notary. However, among a number of notaries, we could be confident that a certain number of them would be honest. For example, assume that 80% of the time, any given notary behaves honestly. With 10 notaries public, we expect 8 of them will behave honestly, even though at the beginning of the survey, we are not sure which ones are honest. Also, in reality, some notaries may not *always* be available for monitoring the process due to circumstances beyond their control. Waiting until every notary becomes available may delay the time for completing the survey, which in turn may delay the involved companies from executing business operations which depends on the result of such surveys (e.g., advertising campaigns).

Another concern is to ensure that the survey is unbiased. Doctors or nurses may be influenced by the answers filled in by people from different hospitals. Thus information provided by doctors or nurses of different hospitals should be kept confidential. Additionally, a doctor or nurse may fill in some information that violates the security and privacy policies of their hospital. Therefore, after they finish answering the survey, the administrative staff of the hospital should check if there are any such violations. If so, they should be able to remove them. However, administrative staff should should not access information provided by other hospitals.

3.2 Related work

An overview of research work and commercial products related to XML security can be found in [40]. Most of the proposals deal with confidentiality issues and do not consider the problem of controlled document updates. Even though we are not aware of other proposals to which our system can be directly compared, related work includes: proposals concerning the update of XML documents [41–43]; group communication techniques and the fault tolerance problem in distributed systems [44–46]; and proposals to manage the illegal behaviour of Byzantine subjects ([47–51]). In general, approaches dealing with updates of XML documents do not deal with security [41,43] or rely on centralized approaches [42]. Therefore they are not suitable for highly decentralized approaches, as the one considered in this chapter. The distributed nature of the collaborative update of XML documents presented in this chapter implies, as a requirement, the use of group communication techniques. A survey of the main group communication specifications is given in [44]. Our protocols take into account the fault tolerance problem inherent in the asynchronous and failure-prone nature of distributed systems. Our design has been heavily influenced by protocols proposed in [45, 46]. The main difference between the previous approaches [44–46] and ours is these approaches are based on the notion of *Views* of a communicating group. That is, messages must be exchanged only between members of the current view. Thus a stop of the communication is generated whenever the view changes according to the insertion of a new member or the exit of a current one. We adopt a group management specification which requires considering at each instant the initial entire communicating group chosen by the DO, that is, $\mathcal{D}G$. Another feature of our protocols is that it provides methods to mitigate malicious behaviors of a limited number of Byzantines. The Byzantine problem has been extensively investigated ([47–51]). Most approaches are based on the specification of conditions according to which it is possible to detect malicious behaviors of Byzantines and to continue without affecting the global computation. We borrow from the above mentioned proposals the idea of adding a number of redundant subjects in a communicating group in order to prevent the supposed number of Byzantine subjects in the group from affecting the communication protocol with their behavior. Moreover, we borrow the idea that when dealing with a set of entities containing some Byzantine ones, each entity must receive a number of messages determined according to the estimated number of Byzantines, to allow the protocols to correctly progress.

3.3 Specification languages

Before we present the specification languages that we have developed to support the collaborative and distributed XML document updating, we first describe the example survey document that will be used.

Example 2 *The survey document (Figure 3.1) is named "Medicine Effect". For simplicity, the survey concerns medicines M1 to M10 which are manufactured by drug companies "C1" to "C10" respectively. The subjects who update this document are doctors from hospitals "H1" to "H50". The doctors should give the positive and negative effects for each medicine, the number of times they prescribed it, and the* *overall efficacy rating of the medicine. Doctors may also extend the document flow path by permitting their nurses to update the document.*

```
Survey name = "Survey.xml" note="Medicine Effect">
 <Medicine name = "M1" company = "C1">
     <Doctor name = "Tony" hospital = "H1">
        <Positive> fill in <\Positive>
        <Negative> fill in <\Negative>
        <Num_of_use> N/A  <\Num_of_use>
        <Overall rate = ".." recommend=".."/>
     <\Doctor>
     <Doctor name = "Don" hospital = "H2">
         ....
     <\Doctor>
     ...
 <\Medicine>
 <Medicine name = "M2" company = "C2">
         ....
 <\Medicine>
 ....
<\Survey>
```

Figure 3.1. An example of XML document

Access control policies are encoded using the $\mathcal{X}$-sec language [52]. The term *Policy Base* ($\mathcal{PB}$) denotes the XML file encoding access control policies that apply to the *DO*'s XML documents.[1] The *Policy Base* is specified according to the $\mathcal{X}$-Sec *Policy Base* template shown in Figure 3.2(a). Note currently, our *Policy Base* template does not support *add_element* and *add_attribute* privileges. We plan to include them in future work.

Example 3 *Figure 3.2(b) shows a $\mathcal{PB}$ referring to the XML document in Figure 3.1. According to the policies in Figure 3.2(b) administrative employees can browse all information filled by doctors or nurses of their hospital. They can also delete information contained in Num_of_use elements for security reason. A doctor or nurse can only update information.*

[1]We assume that each policy is uniquely identified by an identifier, generated by the system when the policy is specified.

```
<!DOCTYPE policy_base[
  <!ELEMENT policy_base
(policy_spec*)>
  <!ELEMENT policy_spec
EMPTY>
  <!ATTLIST policy_spec
      pid ID
      cred_expr CDATA
#REQUIRED
      target CDATA
#REQUIRED
      path CDATA #IMPLIED
      priv (update_attr
          | delete_attr
          | delete_elemt
          | view
          | navigate
          | browse_all)
#REQUIRED
      prop (CASCADE
          | FIRST_LEVEL
          | NO_PROP)
#REQUIRED>
]>
```

(a)

```
<policy_base>
  <policy_spec pid=‘P1’
cred_expr=‘//Type[@Role=‘Admin’ AND
      host=‘H1’]’
      target=‘Survey.xml’ path=‘//Doctor[@host=’H1’]’
      priv=‘browse_all’ prop=‘CASCADE’/ >
  <policy_spec pid=‘P2’
cred_expr=‘//Type[@Role=‘Doctor’ AND
      host=‘H1’]’ target=‘Survey.xml’
      path=‘//Doctor[@host=’H1’]’
      priv=‘update’ prop=‘NO_PROP’/ >
  <policy_spec pid=‘P3’
cred_expr=‘//Type[@Role=‘Doctor’ AND
      host=‘H2’]’ target=‘Survey.xml’
      path=‘//Doctor[@host=’H2’]’
      priv=‘update’ prop=‘CASCADE’/ >
  <policy_spec pid=‘P4’
cred_expr=‘//Type[@Role=‘Nurse’ AND
      host=‘H1’]’ target=‘Survey.xml’
      path=‘//Doctor[@host=’H1’]’ priv=‘update’
      prop=‘CASCADE’/ >
  <policy_spec pid=‘P5’
cred_expr=‘//Type[@Role=‘Admin’ AND
      host=‘H1’]’ target=‘Survey.xml’
      path=‘//Doctor[@host=’H1’]/Num_of_use’
      priv=‘delete_elemt’ prop=‘CASCADE’/ >
  ...
</policy_base>
```

(b)

Figure 3.2. (a) The $\mathcal{X}$-Sec *Policy Base* template and (b) an example of *Policy Base*

Subjects, to which an access control policy applies, are specified by means of *credentials*, encoded in XML using $\mathcal{X}$-Sec [52]. Examples of $\mathcal{X}$-Sec credentials are presented in Figure 3.3.

```
<H_staff>                                   <H_staff>
  <Name> Ann </Name>                          <Name> Cathy </Name>
  <Eid> 112 </Eid>                            <Eid> 235 </Eid>
  <Type Role="Admin" host ="H1"\>            <Type Role="Nurse" host = "H2"\>
<\H_staff>                                  <\H_staff>
<H_staff>                                   <H_staff>
  <Name> Brian </Name>                        <Name> Don </Name>
  <Eid> 110 </Eid>                            <Eid> 253 </Eid>
  <Type Role="Doctor" host ="H1"\>           <Type Role="Doctor" host = "H2"\>
<\H_staff>                                  <\H_staff>
```

Figure 3.3. Examples of $\mathcal{X}$-Sec credentials

A *flow policy* denotes the sequence of subjects that must receive the document. This sequence can be fully specified at the beginning of the update process, or partially

```
<Fpa ...>
 <ReceiverSpec Id = "1">
   <ReceiverProfile Id="2">
     <CredSpec Id = "3">
       //Type[@Role="Doctor" AND @host="H1"]
     </CredSpec>
     <ExtSpec Id="4">subpath</ExtSpec>
   </ReceiverProfile>
 </ReceiverSpec>
 <ReceiverSpec Id = "5">
   <ReceiverProfile Id="6">
     <CredSpec Id = "7">
       //Type[@Role="Admin" AND @host="H1"]
     </CredSpec>
     <ExtSpec Id="8">nosubpath</ExtSpec>
   </ReceiverProfile>
 </ReceiverSpec>
 </ReceiverSpec>
 <ReceiverSpec Id = "9">
   <ReceiverProfile Id="10">
     <CredSpec Id = "11">
       //Type[@Role="Doctor" AND @host="H2"]
     </CredSpec>
     <ExtSpec Id="12">subpath</ExtSpec>
   </ReceiverProfile>
 </ReceiverSpec>
  ...
<\Fpa>
```

Figure 3.4. A possible flow policy

specified when the process starts and then modified and extended by authorized subjects. A flow policy contains some *receiver specifications*, that is, properties that have to be verified by the receivers. Each receiver specification contains one or more alternative *receiver profiles*. A receiver satisfies a receiver specification if it satisfies at least one of the receiver profiles contained in that specification. Receiver profiles consist of a *credential expression*, that is, a condition specified against credentials by means of XPath [53]. 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*.

Example 4 *Figure 3.4 shows a flow policy associated with the document in Figure 3.1. It specifies that the first receiver must be a doctor in hospital "H1" and the second receiver must be the administrative employee of "H1". Only doctors can extend the flow policy by inserting a new sub flow policy. Thus a doctor may let his/her nurses update the information.*

Modifications to flow policies are governed by flow modification rules, which state which subjects can modify a flow policy. Like credentials and access control policies, flow modification rules are encoded using $\mathcal{X}$-Sec. We denote with the term Rule Base ($\mathcal{RB}$) an XML file encoding a set of flow modification rules. This specification language is very similar to that used to specify access control policies, thus we omit the formal presentation of such a language.

3.4 Control information

In this section, we introduce the control information needed by subjects to check document content integrity, and to correctly exercise their modification rights on the document content. Before presenting document control information, we have to introduce some preliminary concepts.

3.4.1 Preliminary definitions

Our approach to ensure confidentiality is based on encryption techniques. All the document portions to which the same policies apply are encrypted with the same encryption key. Each subject that has an authorization over some portions of a document receives all and only the keys needed to decrypt those portions. Further details about this encryption method are available in [54]. In particular, the encryption of a document consists of two main phases: the first, referred to as *marking phase*, marks all document portions with a label containing a list of access control policy identifiers, whereas the second, referred to as *encryption phase*, encrypts all document portions according to the strategy explained above.

This leads to the definition of *document atomic element*, which is the basic portion of an XML document that is individually encrypted.

Definition 3.4.1 (Document Atomic Element). *Let d be an XML document. The set $DocAE(d)$ of document atomic elements of d is defined as follows: 1) for* *each element e in d, and for each attribute*[2] *a in e:* $e.a \in DocAE(d)$;[3] *2) for each element e in d,* $e.tags \in DocAE(d)$.

Note that the reason why we can encrypt the start and end tag of an element with a different key wrt the one used for its content and attributes is that we support attribute-level access control policies. Therefore, elements belonging to an XML document *d* give rise to two or three non-contiguous atomic components in the encrypted document, depending on their type. By contrast, an attribute always corresponds to a single atomic element (that is, the attribute name and its value, or only the value for data content). For this reason, each encrypted document atomic element *docae* has associated a position information that specifies where *docae*'s components are located into *d*.

Example 5 *Examples of atomic elements in the XML document in Figure 3.1 are:*

a) 'name = 'D1"*: the first attribute of the* `Doctor` *element;*

b) 'N/A'*: the data content of the* `Num_of_use` *element;*

c) '<Overall', '/ >'*: the two components of the empty-element* `Overall`*;*

d) '<Doctor', '>', '</Doctor>'*: the three components of the start and end tag of* `Doctor` *element.*

The set of atomic elements encrypted with the same key is called a *document region*. We assume that each document region is uniquely identified by an identifier.

The *DO* of an XML document and/or of a flow policy generates a set of signed certificates, containing information concerning the privileges a subject can exercise over the document and/or flow policy, according to its $\mathcal{PB}$ and $\mathcal{RB}$. Certificates generated for XML documents are called *document modification certificates*, whereas those for flow policies are called *flow policy modification certificates*. These certificates are used by a subject who has modified a document/flow policy portion, to prove its right of modifying that portion to the subsequent receivers of the package.

[2]For simplicity, we consider the data content associated with an element as an attribute, denoted as "`dc`".

[3]Here and in what follows we use the dot notation to denote a component of a given structure.

We do not provide certificates for `add_element` and `add_attribute` privileges because new inserted nodes should be labeled according to the stated *DO*'s access control policies, thus requiring an additional centralized marking phase.

Definition 3.4.2 (Document Modification Certificate). *Let* d *be an XML document managed by the* DO *and let* $\mathcal{PB}$ *be its policy base. Let* $Auth\_P(d) \subseteq \mathcal{PB}$ *be the set of authoring access control policies that apply to* d, *and let* acp *be a policy in* $Auth\_P(d)$. *Let* Sbj_PK*(*acp*) be the set of public keys of subjects authorized to modify* d *according to* acp. *A document modification certificate* dmc, *generated according to* acp, *is a tuple (*cert_id, doc_id, priv, sbj_pk, obj, signature*), where:* cert_id *is the certificate identifier that univocally identifies a document modification certificate among those generated by the* DO*;* doc_id *is the identifier of* d*;* priv *is the privilege contained in* acp*;* sbj_pk $\in$ Sbj_PK*(*acp*);* obj *is one or a set of document regions where sbj has privilege* priv *over them according to the* acp*;* signature *is the digital signature of* DO *over the certificate.*

Example 6 *Consider user Ann, an administrator belonging to the hospital H1. Consider moreover the document in Figure 3.1 and the access control policies in Figure 3.2(b). Furthermore, we assume that:* R_1 *is the identifier of the document region corresponding to: //Doctor[hospital =“H1”]/Num_of_use, whereas* R_2 *is the region containing the document atomic elements corresponding to //Doctor[hospital =“H1”]/Positive. Then, (10, Survey, delete_attr,* PK_{Ann}*,* R_1*, signature)* [4] *is a valid certificate. Since according to the DO's access control policies, Ann is authorized to delete the atomic elements belonging to* R_1*. By contrast, (22, Survey, delete_attr,* PK_{Ann}*,* R_2*, signature) is not a valid certificate, since Ann can only view the atomic elements belonging to* R_2*.*

We omit the description of the flow policy modification certificates, since they are very similar to the document modification certificates.

[4] With the notation PK_s we denote the public key associated with subject s.

3.4.2 Document control information

Table 3.1
Modification declaration structure

Notation	**Structure**	**Semantics**
`ReceiverSpec`	(..., `DocDecl`, ...)	Single receiver specification information inserted by the corresponding receiver
`DocDecl`	(`Doc-UpAttr-Decl`, `Doc-DelAttr-Decl`, `Doc-DelEl-Decl`)	Modification declaration inserted by a receiver when it modifies the document
`Doc-UpAttr-Decl`	set of *r_id*	document region ids declared as updated
`Doc-DelAttr-Decl`	set of (*r_id*, `del-docae`)	Declaration inserted by the receiver when it deletes some attributes of region *r_id*
`del-docae`	set of *docae_id*	set of document atomic elements (attributes) declared as deleted by the receiver
`Doc-DelEl-Decl`	set of (*doc-root_id*, `del-reg`)	deletion declaration of some sub-trees root at *doc-root_id*
`del-reg`	set of *r_id*	set of region ids involved in the deletion

The update of the document requires the insertion in the flow policy attachment some *modification declarations*, having the structure reported in Table 3.1. Moreover, at the end of the document update, s_c must insert in the document, for each modification operation executed on the document, some control information. This guarantees subsequent receivers that s_c possesses the privilege required to execute that operation, and, in case the privilege is `update_attr`, it must compute a new signature on the updated content.

Each document atomic element is marked with a label containing the set of access control policies that apply to it. We can distinguish two main categories of document atomic elements, according to the privileges of those policies: *non-deletable atomic elements* and *deletable atomic elements*. Since a deletable element requires the computation of additional control information wrt a non-deletable one, in this way, we can minimize the amount of control information to be computed and inserted in the document package. Examples of deletable atomic elements are attributes to which at least an access control policy with the `delete_attr` privilege applies or attributes and tags to which at least an access control policy with a `delete_elemt` privilege applies. Table 3.2 shows control data structures associated with both the categories, whereas Table 3.3 presents the components of the structures introduced in Table 3.2.

Table 3.2
Control data structures for document atomic elements

Name	**Notation**	**Structure**	**Semantics**
Control structure for non-deletable document atomic elements	*NDAE_LIST*	list of T_{NDAE}, one for each non-deletable document atomic element of d belonging to a particular region *r_id*	Control information associated with the non-deletable document atomic elements of d belonging to a particular region *r_id*
Control tuple for non-deletable document atomic element	T_{NDAE}	(*docae_id*, *position*, *data*)	Information corresponding to a non deletable atomic element *docae* of a document d
Control structure for deletable document atomic elements	*DAE_LIST*	list of T_{DAE}, one for each deletable document atomic element of d belonging to a particular region *r_id*	Control information associated with the deletable document atomic elements of d belonging to a particular region *r_id*
Control tuple for deletable document atomic element	T_{DAE}	(*docae_id*, *position*, *data*, *h_docae*)	Information corresponding to a deletable document atomic element *docae* of a document d

Table 3.3
Components of the control data structures for document atomic elements

Component	**Semantics**
docae_id	identifier of the document atomic element *docae*
position	value that specifies where *docae*'s components are located in the document
data	encrypted *docae*'s content
h_docae	hash value computed over the *data* component

Similarly, document regions generated by the document marking can be divided in *non-modifiable* and *modifiable regions.* A region is non-modifiable if all policies that apply to it contain only browsing privileges (i.e., `view`, `navigate`, and `browse_all`); a region is modifiable otherwise. Table 3.4 presents the control data structures for non-modifiable regions, whereas Table 3.5 illustrates the semantics of components presented in Table 3.4. We use character ($*$) to denote the string concatenation operator, whereas we use the notation ($\sum^{*}_{x \in ListX} x$) to denote the concatenation of all the elements belonging to $ListX$, in the order in which they are listed. Modifiable regions can be further classified into five sub-categories, according to the different authoring privileges contained in the access control policies that apply to them. This distinction is made for efficiency purposes. Indeed, in this way, we maximize the amount of content statically protected by specific control information and we also reduce the total amount of control information needed to allow integrity check of a region, thus reducing the time required for the integrity check procedure executed by the protocols.

Table 3.4
Control data structures for non-modifiable document regions

Name	**Notation**	**Structure**	**Semantics**
Control structure for non-modifiable document regions	NMR	list of T_{NMR}, one for each non-modifiable region of d	Information used by a subject to verify integrity of non-modifiable document regions of d
Control tuple for non-modifiable document regions	T_{NMR}	(*r_id*, *NDAE_LIST*, *h_nmr_static*)	Information corresponding to a specific non-modifiable document region *r_id* of d

Table 3.5
Components of the control data structures for non-modifiable document regions

Component	**Semantics**
r_id	identifier of a non-modifiable document region of a document d
h_nmr_static	hash value computed over *NDAE_LIST* belonging to *r_id*: $H(\sum^{*}_{t \in NMR[r\_id].NDAE\_LIST} t.docae\_id * t.position * t.data)$

Table 3.6 presents these five sub-categories, giving for each sub-category the corresponding authoring privileges. For example, the set of authoring privileges contained in the access control policies that apply to a region classified as $PDUR$ must be equal to {`update_attr, delete_attr`}.

Without lack of generality, in the following we focus only on fully deletable and updatable regions ($FDUR$), because they are the modifiable regions on which the whole set of authoring privileges supported by our model can be exercised. Thus, they represent the most general and complex modifiable region sub-category. According to this assumption, Table 3.7 presents control data structures for $FDUR$ regions only,

Table 3.6
Modifiable region classification

Sub-category	**Notation**	**Privileges**
Updatable regions	*UR*	{update_attr}
Partially deletable regions	*PDR*	{delete_attr}
Fully deletable regions	*FDR*	{delete_elemt} or {delete_elemt delete_attr}
Partially deletable and updatable regions	*PDUR*	{update_attr, delete_attr}
Fully deletable and updatable regions	*FDUR*	{update_attr, delete_elemt} or {update_attr, delete_elemt delete_attr}

Table 3.7
Control data structures for modifiable document regions

Name	**Notation**	**Structure**	**Semantics**
Control structure for document modifiable regions	*DMR*	(*UR*, *PDR*, *FDR*, *PDUR*, *FDUR*, *delete_elmt_cert*)	Information used to verify correctness of document modifiable regions
...	...	...	...
Control structure for fully deletable and updatable regions	*FDUR*	list of T_{FDUR}, one for each fully deletable and updatable region	Information used by a subject to verify integrity of *FDUR* regions
Control tuple for fully deletable and updatable regions	T_{FDUR}	(*r_id*, *DAE_LIST*, *h_fdur_static*, *sig_fdudocae*, *update_cert*, *delete_attr_cert*)	Information corresponding to a specific *FDUR* region *r_id*

whereas Table 3.8 presents the components of the introduced data structures. With reference to Table 3.8, the *delete_elmt_cert* component contains the certificates with *delete_elemt* privilege, inserted by subjects when they exercised their modification rights, that apply to disjoint set of atomic elements, thus defined `non-overlapping` certificates.

The signature generated by the last subject that has modified the content of a modifiable region is computed on the components *h_docae* associated with the atomic elements belonging to that region and not on their content (*data* component). Such signature is used to check the integrity of the region. That is, we need to check the integrity of the components *h_docae* and then check the correspondence between each *h_docae* component and the corresponding atomic element content (component

Table 3.8
Components of the control data structures for $FDUR$

Component	**Meaning and formal specification**
delete_elmt_cert	it contains `non-overlapping` authoring certificates, with `delete_elemt` privilege inserted by the subjects that have executed a deletion over document regions
r_id	identifier of a modifiable document region
h_fdur_static	hash value computed by *DO* over *docae_id*, *position* and *h_docae* of the document atomic elements that are tags and also over *docae_id* and *position* components of the document atomic elements that are attributes listed in *DAE_LIST* belonging to *r_id*: $\mathrm{H}(\ (\sum^{*}_{t \in FDUR[r\_id].DAE\_LIST, type(t)=tags}\ t.docae\_id * t.position * t.h\_docae) * (\sum^{*}_{t \in FDUR[r\_id].DAE\_LIST, type(t)=attribute}\ t.docae\_id * t.position)\ )$
sig_fdudocae	digital signature computed over the *h_docae* component of all the document atomic elements that are attributes listed in *DAE_LIST* belonging to *r_id* by the last subject (s_{last}) that has modified the region and whose modification declaration is contained in the receiver specification identified by the information: (*fpa-id*, *ver*, *rs-id*, *orig*), where *fpa-id* is a *fpa* identifier, *ver* is a *fpa* version, *rs-id* is a receiver specification identifier and *orig* is a *fpa* originator $\mathrm{S}_{s_{last}}(\ (\sum^{*}_{t \in FDUR[r\_id].DAE\_LIST, type(t)=attribute}\ t.h\_docae) * fpa\text{-}id * ver * rs\text{-}id * orig\ )$
update_cert	it contains the authoring certificate with `update_attr` privilege inserted in a region *r_id* by the last subject that has updated that region
delete_attr_cert	it contains the authoring certificate with `delete_attr` privilege inserted in a region *r_id* by the last subject that has deleted at least one attribute of that region

data), only for those elements not declared as deleted. The same must be done for modification operations executed on the flow policy.

3.5 General system overview

Parties involved in the collaborative update process are: a *Cooperative Group*, denoted as $\mathcal{CG}$, a *Delegates Group*, denoted as $\mathcal{DG}$, and the *DO*. Subjects belonging to $\mathcal{CG}$ are chosen by the *DO* at the beginning of the process. They are the only ones that can be chosen to be the receivers of the XML document. $\mathcal{CG}$ can contain an unlimited number of Byzantine subjects. $\mathcal{DG}$ is a set of subjects also chosen by the *DO* at the beginning of the update process. They are responsible for checking the flow policy integrity (Fpa-Checking) at each step of the process and, whenever required by a subject in $\mathcal{CG}$, to execute document content recovery. The set of delegates is partitioned into three subsets: the set of *Byzantine delegates* ($\mathcal{B}$, with $|\mathcal{B}| \geq 0$), the set of *operative delegates* ($\mathcal{OP}$, with $|\mathcal{OP}| \geq 0$), and the set of *down delegates* ($\mathcal{D}$, with $|\mathcal{D}| \geq 0$). More precisely, operative delegates obey the protocol and are reachable by the subjects, whereas down delegates are unreachable. A down delegate can become operative again, whereas an operative delegate goes down whenever a failure occurs. Note, for each delegate in $\mathcal{DG}$, no one can tell whether it belongs to $\mathcal{B}$, $\mathcal{D}$, or $\mathcal{OP}$ at the beginning of the update.

The DO is the subject who generates the XML document package (Doc_{DO}) to be updated and the associated flow policy attachment (Fpa_{DO}). This subject also specifies the set of access control policies that apply to Doc_{DO} and the set of flow modification rules that apply to Fpa_{DO}.

During the update process, the document generated by the DO is sent to a first chosen subject in $\mathcal{CG}$. Each subject s_c (except the first one) in $\mathcal{CG}$ performs the document integrity checking when it receives it, according to the control information in the document it received from the previous subject, and the Fpa it received from the delegates. Whenever an error occurs to the document content, the subject contacts all delegates in $\mathcal{DG}$ to start a recovery. At the end of this recovery, the subject obtains the last correct version of the document and can thus update the document according to its modification rights. During recovery, delegates interact with subjects in $\mathcal{CG}$ to obtain the last correct version of the document and build the recovered document to be sent to the requester.

After subject s_c executes its operations on the document and/or Fpa according to the privileges it possesses, it will insert certificates which can be verified by the later subjects. s_c then sends the Fpa to $\mathcal{DG}$ for Fpa_Checking. That is, s_c should have $\mathcal{Q}$ signatures from delegates which approve the current version of Fpa (as s_c may modify the Fpa). Before s_c sends the document to the next subject, it will make all operative delegates' states stable (this is called Change-Delegates-State). That is, s_c will send to all delegates Fpa which is signed by at least $\mathcal{Q}$ delegates. Upon receiving the message, each operative delegate will forward this message to other delegates if the message is correct (Fpa is signed by at least $\mathcal{Q}$ delegates). Thus all operative delegates will have the same Fpa. Also, if s_c requested recovery of the document, it also needs to send the recovered document version which is signed by at least $\mathcal{Q}$ delegates. The purpose of Change-Delegates-State is to ensure that all operative delegates have the same information. That is, they have the same Fpa, recovered document version, etc. Finally, s_c sends the document to the next subject according to the flow policy attachment content. All delegates will send the approved Fpa to the next subject.

After the last subject in the Fpa sends the document to the DO, if the document is correct, the DO sends a message to end the process. Otherwise, the DO requests the $\mathcal{DG}$ to recover the document in order to get the last correct version of the document.

Before we describe our protocols in detail, we first present our assumptions of the system and how to set the parameters.

3.5.1 Assumptions

Our approach relies on a set of assumptions. First, we assume that the DO, each delegate and each subject involved in the update process possesses a private key and that all the other parties know or can retrieve the public key of each other. The DO is in charge of informing, at the beginning, all subjects and delegates of which users compose $\mathcal{CG}$ and $\mathcal{DG}$. Moreover, we assume that there is a finite upper bound on message transmission time. This means that if an honest party sends a message to another honest and reachable party, the message is received by a fixed amount of time ($MTTIME$). Each sent message is always signed by the sender for integrity and authentication purposes. To avoid deadlocks caused by the malicious behavior of a Byzantine subject s_c, a *Rollback* procedure is executed to replace s_c with another subject after a fixed amount of time by the last change of state executed by an operative delegate.[5]

[5]In the protocols specified in this paper, we do not address this issue, that it is assumed to be a parallel process auditing the delegate behaviour and starting its task when needed.

3.5.2 Protocol parameters setting

At the beginning of the update process, the DO has to set two parameters: b and d, that respectively represent the maximum number of Byzantine delegates that do not affect the protocol, and the maximum number of down delegates that do not delay the protocol. The parameter b may be set by the system with the default vale of 0, or may be set by the DO. Value d is set as $\lceil c \cdot f \rceil$ where f is an estimated average number of failures proposed by the system and c is the correction parameter set by the DO, the default value of which is 1.

Another important parameter op, which is the number of operative delegates, is strictly related to Quorum $\mathcal{Q}$, which is the minimum number of delegates required for making progress. The relationship between op and $\mathcal{Q}$ is:

$$\begin{aligned} &op \geq \mathcal{Q} && \text{(constraint 1)} \\ &b + (op + d)/2 < \mathcal{Q} && \text{(constraint 2)} \end{aligned}$$

Constraint 1 states that op must be greater than or equal to $\mathcal{Q}$, because in case Byzantine delegates do not answer a request, only operative delegates will be able to sign a message content for which the protocol requires at least $\mathcal{Q}$ valid signatures.

Constraint 2 enforces that a Byzantine cannot able to obtain two sets of valid signatures of cardinality at least equal to $\mathcal{Q}$ for two different messages of the same type, or for two messages of the same priority, exchanged in the same step, when there is the maximum number of Byzantine delegates and no down delegate.[6] The minimum value of op that assures all the above requirements is $(2b + d + 1)$ and the corresponding value for $\mathcal{Q}$ is $(2b + d + 1)$ too.

3.6 Distributed and cooperative update process protocols

Our approach relies on a suite of protocols, namely: the protocol executed by the DO (*Document Originator Protocol*); the protocol executed by the subjects in $\mathcal{CG}$

[6]More details about message types/priorities are presented in Section 3.6.

(*Subject Protocol*); and the protocol executed by the operative delegates (*Delegate Protocol*). First, we explain some terminologies and data structures we used in these protocols.

3.6.1 Terminology and structures

We call $State^{x}_{dl}$ the state associated with a delegate $dl \in \mathcal{OP}$ at step x of the process. We call *step* all the operations/interactions executed by a subject $s_c \in \mathcal{CG}$ and delegates, from the reception of the document and flow policy attachment by s_c, to the delivery of the updated document to the *next receiver* ($s_{next} \in \mathcal{CG}$). The complete cooperative and distributed update process thus consists of a set of steps. $State^{x}_{dl}$ which is stored in the local storage of dl, contains following components that can be possibly updated step by step: a Document (Doc), a flow policy attachment (Fpa), a structure containing the invalid modification document declarations ($IMDD$), a structure used during the recovery that indicates when the last recovery occurred for each region ($LSRR$), a vector of progressive numbers used to protect against replay attacks ($N_{\mathcal{IP}}$, where $\mathcal{IP} = \mathcal{CG} \cup \mathcal{DG} \cup \{DO\}$). For a delegate $dl \in \mathcal{OP}$, a step x ends and the subsequent one $(x+1)$ begins when dl makes $State^{x+1}_{dl}$ *stable*, that is, the values of modifiable information contained in $State^{x}_{dl}$ are replaced with the new ones, according to the information contained in the last correct message sent by s_c to all delegates.

Next, we explain some data structures used by a delegate in detail.

- $IMDD$ (*Invalid Modification Document Declarations*) This structure contains invalid declarations inserted during each recovery (see example 3.6.1). Invalid declarations are stored in $IMDD$ according to the subject that has inserted them in Fpa, and according to the type of operation associated with it (update_attr/delete_attr/delete_elmt privilege). A subject is identified in $IMDD$ through the information that specifies its position in Fpa at the time of insertion of that declaration in Fpa itself.

Example 7 *Don did not extend the survey path to his nurse Cathy. However, Cathy modified the attribute rate filled by Don and inserted in the document control information and Fpa her modification declarations. When the document passed to Lynn, who is the administrative staff of the hospital, she found the document corrupted. Lynn asked notaries for a recovery. Notaries will recover the document by undoing Cathy's modifications, and put Cathy's modification declarations into $IMDD$.*

- $LSRR$ (*Last Saved Region Recovery*) This structure is used by a delegate during the recovery. It stores, for each modifiable region, the information that identifies the subject in Fpa that has generated the last detected as corrupted version of the document wrt that region. During a recovery, only subjects that have declared some modifications on a region to be recovered and that appear in Fpa in a position greater than that stored in $LSRR$ for that region, will be contacted to obtain the most recent correct region content. Since previous recovery has stored the most recent correct region content wrt the declarations in Fpa inserted by subjects in a position less than that stored in $LSRR$ for that region, the recovery process will use it to recover the region if it does not receive a more recent correct region content by the contacted subjects.

- $N_{\mathcal{IP}}$ This structure is a vector of progressive numbers, initially sets to all zeros, one for each party involved in the process. It is used to keep track of the number of steps a subject or the DO has taken part in, and how many times a delegate has requested an *agreement*. Whenever a subject/DO participates in a step, the corresponding progressive number is increased. The indication of the receiver in the messages sent by a delegate, together with the insertion of the value stored in $N_{\mathcal{IP}}$, which corresponds to that receiver, prevents Byzantine subjects and/or Byzantine delegates from replaying messages exchanged in a step x during a step y, with $y > x$.

- *Queue* This structure stores all the received messages. During a generic step x, a delegate needs a strategy to choose among all the received messages the next one to process. This strategy is called *received messages scheduling policy* and it is applied each time a message has been completely processed or when the time assigned to a process that processes a message ends. This policy collects among all the messages in *Queue* only the messages valid according to $State^x$, and the values of the previous introduced variables. Then, it selects from this set the messages with higher priority and in case of more than one message, the message received first.

- *state* This variable contains a string which indicates the state in which the delegate is. For example, the value *norec* for this variable indicates that the delegate has not yet requested a recovery or it is completing a step without the need of a recovery.

- *requests* This variable is an integer, indicates the number of processed requests wrt the value of variable *state*. If no recovery has been requested, variable *requests* reaches at most value 1, whereas it reaches value 2 in presence of recovery. Variable *state* and *requests* are used to avoid replay attacks within the same step.

The Subject Protocol also makes use of variable *state* to represent the action the subject is executing or has just executed. A subject also use a structure $N_{\mathcal{CG}}$ which is similar to $N_{\mathcal{IP}}$, to keep track of the number of steps all subjects have taken part in. Whenever a subject s sends a message, this message contains the progressive number associated with s ($N_{\mathcal{CG}}[s]$). This information is used by a receiver to discard old messages.

Parties involved in a cooperative and distributed update process communicate by exchanging messages. Table 3.9 gives more details about the exchanged messages. In the table, messages are presented in terms of type, sender, receiver(s), and their complete structure and semantics. Only messages received by a delegate have associated a priority. This is because only Down Delegate and Delegate Protocols use this information to choose the next message to process. Figure 3.5 shows the message exchanged between the involved parties.

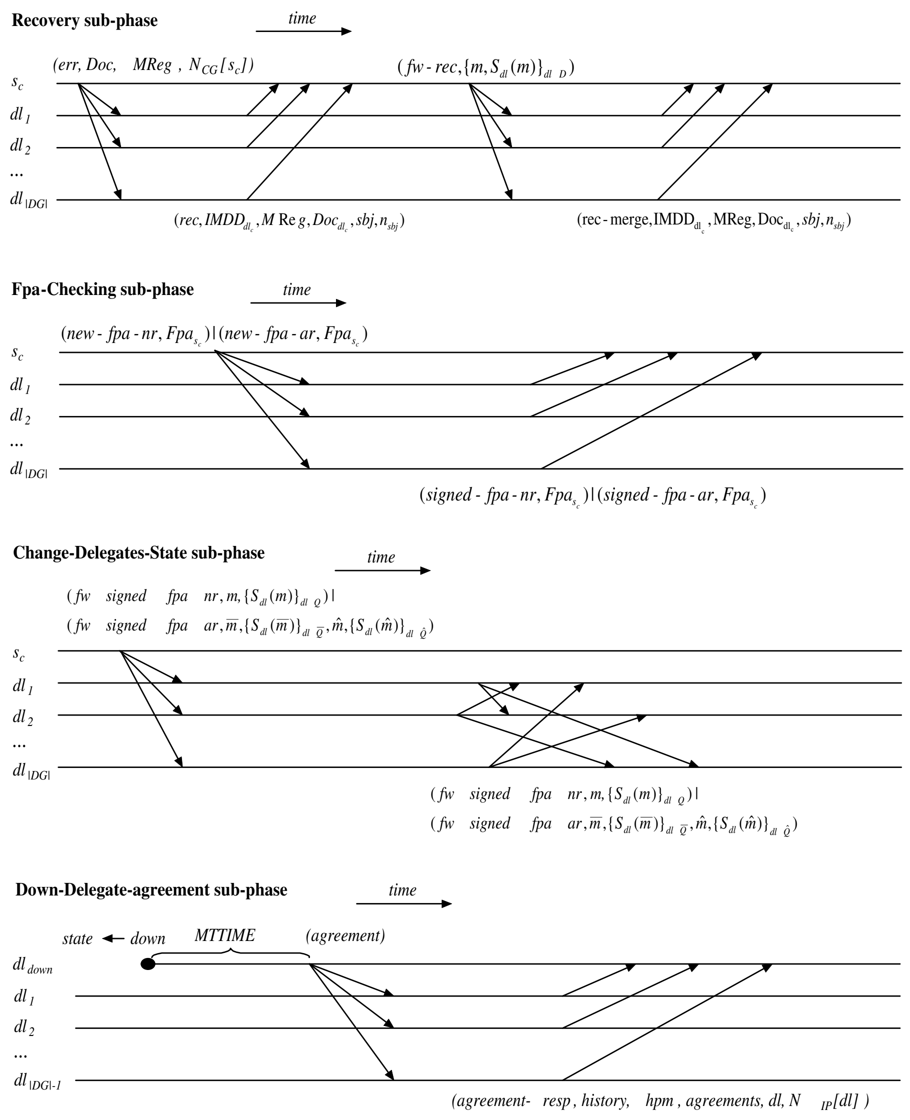

Recovery sub-phase
time
(err, Doc, MReg , $N_{CG}[s_c]$)
$(fw\text{-}rec, \{m, S_{dl}(m)\}_{dl \in D})$
s_c
dl_1
dl_2
...
$dl_{|DG|}$
$(rec, IMDD_{dl_c}, MReg, Doc_{dl_c}, sbj, n_{sbj})$
$(\text{rec-merge}, \text{IMDD}_{dl_c}, \text{MReg}, \text{Doc}_{dl_c}, sbj, n_{sbj})$
Fpa-Checking sub-phase
time
$(new\text{-}fpa\text{-}nr, Fpa_{s_c}) | (new\text{-}fpa\text{-}ar, Fpa_{s_c})$
$(signed\text{-}fpa\text{-}nr, Fpa_{s_c}) | (signed\text{-}fpa\text{-}ar, Fpa_{s_c})$
Change-Delegates-State sub-phase
time
$(fw\ signed\ fpa\ nr, m, \{S_{dl}(m)\}_{dl \in Q}) |$
$(fw\ signed\ fpa\ ar, \overline{m}, \{S_{dl}(\overline{m})\}_{dl \in \overline{Q}}, \hat{m}, \{S_{dl}(\hat{m})\}_{dl \in \hat{Q}})$
$(fw\ signed\ fpa\ nr, m, \{S_{dl}(m)\}_{dl \in Q}) |$
$(fw\ signed\ fpa\ ar, \overline{m}, \{S_{dl}(\overline{m})\}_{dl \in \overline{Q}}, \hat{m}, \{S_{dl}(\hat{m})\}_{dl \in \hat{Q}})$
Down-Delegate-agreement sub-phase
time
state ← down
MTTIME
(agreement)
dl_{down}
$dl_{|DG|-1}$
(agreement- resp, history, hpm, agreements, dl, $N_{IP}[dl]$)

Figure 3.5. Messages exchange

Table 3.9
Messages

P	Type	Sender	Rcvr(s)	Content and Semantics
-	*init-dg*	DO	$\mathcal{DG}$	($init\text{-}dg$, d_id, Doc_{DO}, Fpa_{DO}, $\mathcal{CG}$, $\mathcal{DG}$) Message sent by the DO to all delegates containing the original version of the document, the initial flow policy attachment and the set of delegates and subjects involved in the process
-	*init-cg*	DO	$\mathcal{CG}$	($init\text{-}cg$, d_id, $regkeys_s, docmodcert_s, fpmodcert_s, \mathcal{CG}, \mathcal{DG}$) Message sent by the DO to each subject s containing s's decryption keys, document/flow policy certificates and the set of subjects and delegates
0	*agreement*	$dl \in \mathcal{D}$	$\mathcal{DG}$	($agreement$) Message sent by a down delegate to all delegates in $\mathcal{DG}$ to receive information needed to reach the same state of the operative delegates
-	*agreement-resp*	$\mathcal{DG}$	$dl \in \mathcal{D}$	($agreement\text{-}resp$, $history, hpm, agreements, dl, N_{\mathcal{IP}}[dl]$) Message sent by delegates to a down delegate containing the history of all previous steps, in terms of messages of type *fw-signed-fpa-nr* or *fw-signed-fpa-ar* that cause the step change ($history$), their last processed message (hpm), all the received but not still processed agreement messages ($agreements$) and information required to prevent other delegates to replay this message (a progressive number and the public key of the down delegate receiver)
1	*err*	$s_c \in \mathcal{CG} \mid DO$	$\mathcal{DG}$	(err, $m, S_{sbj}(m), MReg, N_{\mathcal{CG}}[s_c]$) Message sent by the current subject/DO to all delegates when an error occurs to the document content to collect $(b+1)$ recovery versions
-	*rec*	$\mathcal{DG}$	$s_c \in \mathcal{CG} \mid DO$	(rec, $IMDD_{dl_c}, MReg, Doc_{dl_c}, sbj, n_{sbj}$) Message sent by delegates to the current subject/DO containing the result of their recovery: a Doc and the updated IMDD structure
2	*fw-rec*	$s_c \in \mathcal{CG}$	$\mathcal{DG}$	($fw\text{-}rec$, $\{m, S_{dl}(m)\}_{dl \in D}$) Message sent by the current subject to all delegates to receive the last correct document version wrt its accessible modifiable regions, obtained unifying the $\|D\| = (b+1)$ forwarded recovery versions
-	*rec-merge*	$\mathcal{DG}$	$s_c \in \mathcal{CG}$	($rec\text{-}merge$, $IMDD_{dl_c}, MReg, Doc_{dl_c}, sbj, n_{sbj}$) Message sent by delegates to the current subject containing a Doc, union of the $(b+1)$ received recoveries, and the IMDD structure, updated according to the $(b+1)$ received recoveries
3	*new-fpa-nr*	$s_c \in \mathcal{CG}$	$\mathcal{DG}$	($new\text{-}fpa\text{-}nr$,Fpa_{s_c}) Message sent by the current subject to all delegates to propose a new Fpa to be made stable, in absence of recovery
-	*signed-fpa-nr*	$\mathcal{DG}$	$s_c \in \mathcal{CG}$	($signed\text{-}fpa\text{-}nr$, Fpa) Message sent by delegates to the current subject if the proposed Fpa is correct, in absence of recovery
3	*new-fpa-ar*	$s_c \in \mathcal{CG}$	$\mathcal{DG}$	($new\text{-}fpa\text{-}ar$,Fpa_{s_c}) Message sent by the current subject to all delegates to propose a new Fpa to be made stable, after a recovery
-	*signed-fpa-ar*	$\mathcal{DG}$	$s_c \in \mathcal{CG}$	($signed\text{-}fpa\text{-}ar$, Fpa) Message sent by delegates to the current subject if the proposed Fpa is correct, after a recovery
4	*fw-signed-fpa-nr*	$s_c \in \mathcal{CG} \mid \mathcal{DG}$	$\mathcal{DG}$	($fw\text{-}signed\text{-}fpa\text{-}nr$, $m, \{S_{dl}(m)\}_{dl \in Q}$) Message sent by the current subject to all delegates and then forwarded by delegates to each other delegate to make stable the Fpa contained in m and previously signed by $\|Q\| = \mathcal{Q}$ delegates
4	*fw-signed-fpa-ar*	$s_c \in \mathcal{CG} \mid \mathcal{DG}$	$\mathcal{DG}$	($fw\text{-}signed\text{-}fpa\text{-}ar$, $\overline{m}, \{S_{dl}(\overline{m})\}_{dl \in \overline{Q}}, \widehat{m}, \{S_{dl}(\widehat{m})\}_{dl \in \widehat{Q}}$) Message sent by the current subject to all delegates and then forwarded by delegates to each other delegate to make stable the Doc contained in $\widehat{m}$ and the Fpa contained in $\overline{m}$ and previously signed by $\|\overline{Q}\| = \|\widehat{Q}\| = \mathcal{Q}$ delegates
5	*end*	DO	$\mathcal{DG} \cup \mathcal{CG}$	(end, d_id) end the update process

3.6.2 DO protocol

The DO chooses $\mathcal{CG}$ and $\mathcal{DG}$, and distributes to all delegates information (Doc_{DO}, Fpa_{DO}, $\mathcal{CG}$, $\mathcal{DG}$). Then, it distributes the decryption keys and document/flow policy modification certificates to the corresponding subjects in $\mathcal{CG}$. Finally, the DO sends to the first subject the DO's version of the document to be updated (Doc_{DO}) and the DO's version of the associated flow policy attachment (Fpa_{DO}).

At the end of update, the DO receives from the last receiver subject (sbj) in Fpa a message m containing the document (Doc_{sbj}) and a message signed by $(2b + d + 1)$ delegates containing Fpa and $IMDD$. It checks the document integrity and, if an error occurs, it sends to all delegates an error message for recovery. Each delegate $dl \in \mathcal{DG}$ generates a document recovery version by contacting subjects in $\mathcal{CG}$ and then sends a message containing its version to the DO. The DO accepts the first $(b+1)$ messages from the delegates and then composes them to obtain the last correct document version. [7] The original document is thus replaced by this new document. At this point, the DO sends a message (end, d_id) to all delegates and subjects, to end the cooperative update process concerning document with identifier equal to d_id.

3.6.3 Subject protocol

When subject $s_c \in \mathcal{CG}$ receives a message m from a subject $sbj \in \mathcal{CG}$, and at least $b + 1$ messages from different delegates such that the structure of Fpa and $IMDD$ are all the same from these messages of the delegates, it can check if m contains a corrupted document according to the received $IMDD$ and Fpa. As we will see in the delegate protocol, an operative delegate will not send Fpa and $IMDD$ to a subject unless at least $(2b + d + 1)$ delegates approves this Fpa and $IMDD$.

If there is no error in m, s_c executes the operation on the document. Otherwise, it sends to all delegates a recovery message to obtain the last correct document version wrt the set of regions it can access.

[7] More details about the recovery functionality are presented in Section 3.7.

s_c accepts the recovery reply messages from $(b+1)$ delegates and then puts these messages in a message $\overline{m}$ and sends $\overline{m}$ to all delegates. Next, s_c waits to receive recovery messages from at least $(2b+d+1)$ delegates. The recovery results from these $(2b+d+1)$ delegates must be the same. Then this finishes the recovery and s_c can start operations on the correct document version.

After executing the operation on the document and/or Fpa, s_c has to send its updated Fpa to all delegates in order to be checked and signed (This is called Fpa_Checking). If no recovery has been requested by s_c during the step, s_c sends a message (*new-fpa-nr*,Fpa_{s_c}) to all delegates; a message (*new-fpa-ar*,Fpa_{s_c}) is sent otherwise. s_c has to send to each delegate a message whose type depends on whether it has requested a recovery or not in the step. A delegate verifies if the updated Fpa is correct, that is, if s_c has the certificates which authorize it to update the Fpa and it inserted these certificates in the Fpa. If so, the delegate signs the message containing the Fpa and sends it back. Example 8 illustrates the importance of Fpa_Checking.

Example 8 *When Don extends the flow path by letting his nurse Cathy update the survey, he should get enough signatures (at least of $\mathcal{Q}$) from delegates by requesting Fpa_Checking for the new proposed flow path. Subsequent modification made by Cathy will be valid if she inserts her credentials (see Figure 3.3). However, if Don does not extend the flow path, any modification made by Cathy will be invalid, even though she inserts her credentials. Fpa_Checking is also important to prevent Byzantine delegates from damaging the integrity of the update. As previously mentioned in Section 3.1, if a Byzantine delegate deleted what Cathy filled in because the information does not favor the drug company he favors, he must have a message from Don who proposed the flow path without extension. However, he could not get such message.*

When s_c receives $(2b+d+1)$ such signatures from delegates, it is ready to change delegate state. That is, before sending the document to the next subject, s_c has to notify all delegates the document recovery version and corresponding $IMDD$ structure generated during the recovery, if any, and the correct Fpa proposed. s_c does this by sending a message $\widetilde{m}$ to all delegate. If no recovery happened in the step, $\widetilde{m}$ is of type *fw-signed-fpa-nr* and it contains the new Fpa and $(2b + d + 1)$ signatures of it from delegates. Otherwise, $\widetilde{m}$ is of type of *fw-signed-fpa-ar* and it contains: 1) message $\overline{m}$ of type *fw-signed-fpa-ar* containing the new Fpa; 2) $(2b + d + 1)$ delegate signatures on $\overline{m}$; 3) message $\widehat{m}$ of type *rec-merge* containing the document recovery version and associated IMDD structure; 4) $(2b+d+1)$ delegate signatures computed on $\widehat{m}$.

After s_c sends $\widetilde{m}$, it sends the document to the next subject according to the Fpa. s_c needs to wait until receiving a message from the DO indicating that the update process ends. Before the update process end, some delegates may contact s_c for recovering the document.

As we will see in the delegate protocol, a subject cannot generate a valid message of type *fw-signed-fpa-nr* and another valid message of type *fw-signed-fpa-ar* in the same step, because the protocols prevent this subject from collecting $\mathcal{Q}$ signatures for a message of type *new-fpa-nr* and $\mathcal{Q}$ signatures for a message of type *new-fpa-ar*. Indeed, in the same step, a delegate does not accept messages with the same priority.

Example 9 *After Don extended the path and got Fpa_Checking, the document passed to Cathy. Cathy filled in parts of the survey. Later on, a Byzantine delegate cannot undo Cathy's update, even by colluding with Don. This is because at least $\mathcal{Q}$ delegates should sign Fpa. Don cannot get $\mathcal{Q}$ delegates sign a different Fpa.*

3.6.4 Delegate protocol

The initial *state* of a delegate is *norec*. If a delegate *dl* receives a recovery message from *sbj* who is in $\mathcal{CG}$, *dl* checks the following before doing any recovery: 1) according to the Fpa stored in the current state, *sbj* should be the subject doing update process now. As we will show later, all operative delegate always have the same and correct Fpa at each step, due to the *State* consistancy; 2) the request associated with the current step is 0; 3) current *state* variable is *norec*; 4) the document contained in the recovery message is signed by a previous subject who is before sbj in the Fpa; 5) this is not a replay attack, according to the information stored in $N_{\mathcal{IP}}$. If there is any error, dl just ignores the message. Otherwise, it generates a document recovery version and the corresponding $IMDD$ updated structure by contacting subjects in $\mathcal{CG}$ and then sends to sbj a message containing the generated information.

If dl receives from sbj a message m of type *fw-rec*, it will check the following: 1) variable $state$ is rec; 2) $sbj \in \mathcal{CG}$; 3) variable $requests$ is 0; 4) according to the Fpa, sbj is the current subject who is updating the document; 5) there are $(b+1)$ recovery messages signed by different delegates. If there is no error, dl sets $requests = 1$ and m as the hpm; then it generates a merge version of Doc and $IMDD$ and sends it to sbj.

When dl receives a message m from $sbj \in \mathcal{CG}$ for Fpa_Checking, it checks the following. If the message m is of type *new-fpa-nr*, $state$ variable must be $norec$ and $requests$ must be equal to 0, as this is the first request from sbj in this step. If the message m is of type *new-fpa-ar*, $state$ variable must be rec and $requests$ must be equal to 1. Also, from the Fpa stored in dl's $State$, sbj should be the current subject requesting for Fpa_Checking. If all above are satisfied, dl increases the variable $requests$ by 1 and set m as the hpm. dl then checks the integrity of the proposed Fpa from sbj. If no error in the proposed Fpa, dl sends a signed message of type *signed-fpa-nr* or type *signed-fpa-nr* to sbj, depending whether a recovery happened or not in this step.

When dl receives a message m of type *fw-signed-fpa-nr* or *fw-signed-fpa-ar* from sbj for commit the step, it checks the following. 1) In the case that m is of type *fw-signed-fpa-nr*, them m must contain a message $\overline{m}$ of type *signed-fpa-nr* and $(2b+d+1)$ delegate signatures on $\overline{m}$. This indicates that at least $(2b+d+1)$ delegates agree with the Fpa proposed in $\overline{m}$. 2) In the case that m is of type *fw-signed-fpa-ar*, them m must contain a message $\overline{m}$ of type *signed-fpa-ar* and $(2b+d+1)$ delegate signatures on $\overline{m}$. Also, m must contain a message $\widetilde{m}$ of type *rec-merge* which contains a recovered version of document and the corresponding $IMDD$, and $(2b+d+1)$

delegate signatures on $\widetilde{m}$. If the above are satisfied and the message is not a replay attack according to the information stored in N_{IP} , dl updates the components of $State$, that is, it sets the variable $state$ as $norec$, $requests = 0$, $hpm = m$ and puts m into $history$. dl also sends the received message m to other delegates, in case sbj did not send m to them. So all the operative delegates will have the same state. That is, in case m of type *fw-signed-fpa-ar*, all operative delegates set components of their $State$, such as the current document version, $IMDD$, $LSRR$ etc. according to the one contained in $\widetilde{m}$. dl thus finishes a step, and $State$ is stable. It then sends a message containing Fpa and $IMDD$ to the next subject in the path.

The designed protocols assure that each operative delegate signs only once such information at each step, thus preventing Byzantine subjects from obtaining two different contents signed by at least $\mathcal{Q}$ distinct delegates and forcing different delegates, such as $dl_i, dl_j \in \mathcal{OP}$, to make stable $State^{x+1}_{dl_i}$ and $State^{x+1}_{dl_j}$ with $State^{x+1}_{dl_i} \neq State^{x+1}_{dl_j}$.

3.7 Recovery

The goal of recovery is to retrieve the last correct version of the regions accessible by s_c. It accomplishes this task by contacting subjects in $\mathcal{CG}$ that have received till that point at least a document package and that have executed at least a modification operation on at least one region accessible by s_c. Also, it is desirable to limit as much as possible the number of subjects to be contacted. Correctness of a region is determined according to the set of valid modification declarations inserted in Fpa.

A declaration of deletion of a document sub-tree is valid if 1) a certificate corresponding to this declaration exists or 2) a certificate corresponding to a deletion declaration of a document sub-tree that includes the deleted sub-tree exists.

A delete_attr modification declaration is valid if the subject that has inserted this declaration in Fpa has also inserted in the document control information of the currently analyzed document version a corresponding certificate. A delete_attr modification declaration is also valid if it is not evaluated as invalid till that point and another valid subsequent delete_attr declaration exists in Fpa. That is, A delete_attr declaration is considered valid if a subsequent subject s will correctly exercise the delete_attr privilege on the same region. The same strategy applies to update_attr and delete_elemt modification declarations. Even more, a valid declaration for the deletion of a sub-tree a, and its corresponding certificate, makes other not yet evaluated deletions of a sub-tree b such that $b \subseteq a$, valid. This implies that a recovery is done from backwards.

An update_attr declaration is valid in presence of a proper certificate and when the content of h_docae components associated with elements of the modified region are correct wrt the signature computed by the subject who generated the declaration. Moreover, content of atomic elements not declared as deleted must be correct wrt the corresponding h_docae component.

Next, we give a high level description of the recovery algorithm. This algorithm is used by a delegate to generate a document recovery version and the corresponding $IMDD$ structure.

Initially, the algorithm replaces all non-modifiable information in the document to be recovered (Doc), with those in the stable version of the document (Doc_{st}). Then function `Rec-MDD-Collection` collects all modification declarations, not yet inserted in the stable version of $IMDD$ and associated with at least a region in $MReg$, in structure MDD. Each delete_attr/update_attr declaration, among the previous selected ones, is collected if it is present in a position of Fpa greater than the position stored in $LSRR$ corresponding to the region to which the modification declaration is associated with. Each delete_attr modification declaration (dad) in MDD associated with a region r contains: 1) the r's identifier, 2) the cumulative set of elements declared as deleted, that is, the union of the sets of elements declared as deleted by all the delete_attr declarations, for r, in MDD that precede dad in Fpa and the set of elements specified in dad, and 3) the public key of the subject that has generated *dad*. In MDD are also inserted, for each region in $MReg$, the most recent valid update_attr/delete_attr declarations determined during the last recovery.

The algorithm considers the position in Fpa corresponding to the subject ($\texttt{sbj}(Doc)$) that has generated the corrupted document to be recovered as the initial recovery position. Component *set-del-docae* will contain all the elements declared as deleted in valid delete_elemt/delete_attr declarations. It analyzes declarations in MDD until the set is empty and when required it contacts a subject to obtain a document version against which to evaluate the declarations in MDD. The algorithm starts by the document version to be recovered and then, if required, other document versions are obtained in reverse order wrt the order associated with receiver specifications in Fpa.

When the algorithm analyzes certificates associated with delete_elemt privileges, it removes all certificates that are not correct, or that are not associated with a delete_elemt declaration, or that are contained in another certificate. Delete_elemt declarations are considered valid according to the strategy we described at the beginning of this Section. Elements declared as deleted in these valid declarations are saved in *set-del-docae* component.

Similarly, the algorithm analyzes each region for which there exists a delete_attr declaration in MDD. It determines the most recent delete_attr declaration in MDD for a region r (*dad*). If its position is less than or equal to that stored in $LSRR$ for r, then there are no valid delete_attr declarations after the last recovery. Therefore the certificate inserted in Doc is copied by Doc_{st}, also inserting in *set-del-docae* the set of elements declared as deleted associated with *dad* in MDD. If there is a corresponding correct certificate in the currently analyzed document version, then it inserts in Doc this certificate, saves all elements declared as deleted in *set-del-docae*, and removes from MDD all delete_attr declarations associated with r. At point 5, the algorithm analyzes update_attr declarations.

Next, at point 7 all declarations with position in the Fpa equal to that of the currently analyzed document version are removed from MDD, since no other subsequently required document version can make them valid. They are inserted in $IMDD$.

Then, the algorithm determines the set *decl_set* which contains the declarations of the subject who is at the highest position of the Fpa among these subjects who made declarations in MDD. If at least one declaration is at a position greater than that stored in $LSRR$ for the corresponding region, then a request is sent to the subject that has generated this set of declarations to obtain its last stored document version, if this subject has not been contacted up to that point. If the subject is unreachable, the algorithm removes *decl_set* from MDD and inserts *decl_set* in $IMDD$, then it determines the new value of *decl_set* according to the content of MDD. When *decl_set* does not contain anymore delete_elemt declarations and delete_attr/update_attr declarations with position greater than that stored in $LSRR$ for the corresponding region, the algorithm removes *decl_set* from MDD and for each declaration in *decl_set* it copies content and certificate (in case of update_attr declaration) or only certificate (in case of delete_attr declaration) in *Doc* from the stable document version Doc_{st}.

Finally, when MDD is empty, the algorithm sets to *null* all elements of *Doc* saved in *set-del-docae*. The algorithm ends returning *Doc*, the produced document recovery version, and the corresponding updated $IMDD$ structure.

Given $(b + 1)$ document recovery versions, there must be one that is consistent with these of all operative delegates. All operative delegates will sign such one and send it to the requester. A recovery responses merge algorithm can be found in [55]).

3.8 Performance evaluation

This section evaluates the performance of our approach. We compare it with the case that only one party (trusted) is involved for flow path integrity checking and document recovery.

3.8.1 Experimental setup

We measure the time to complete one *step.* Both delegates (or the trusted party) and subjects ran on identical Solaris workstations. These workstations have 450MHz CPUs and 2GB of memory. The network bandwidth is 100MB/s. Point-to-point communications is implemented using TCP. RSA of 1024-bit modulus is used for digital signatures. To ensure the confidentiality of the document, AES with a key size of 256 bit is used for encryption.

We performed the experiment with 1 Byzantine delegate, 0 down delegate. Therefore, there are 3 operative delegates; totally 4 delegates. Fpa is about 5KB and the document is 100KB.

3.8.2 Results

Figure 3.6 reports the time complexity when no recovery is requested. In this case, our approach does not have much more delay than the one-party approach. The overhead introduced in our approach is from the time spending on the change-delegate-state sub-phase. This sub-phase requires the subject to collect 3 signatures on the proposed Fpa and then broadcast them to all delegates, which takes less than 10ms. This low overhead is due to the fact that the subject does not need to perform any encryptions or generating any signatures; it only needs to verify at most 4 signatures. Compared to the overall time (362ms) of the one-party case when the operation time is 20ms, this overhead is only 3%. When operation time increases, this overhead decreases.

Next, we measured the time complexity when recovery is requested. Figure 3.7 reports the results when only one subject needs to be contacted by delegates (or one-party) for recovery of the document regions which is of size 0.5KB. As we can see, the overall time increases for both approaches. Even for a one-party case, the time is almost double that of no recovery. This due to the fact that the party needs to decrypt the document, and then find the subjects to be contacted in order to recover

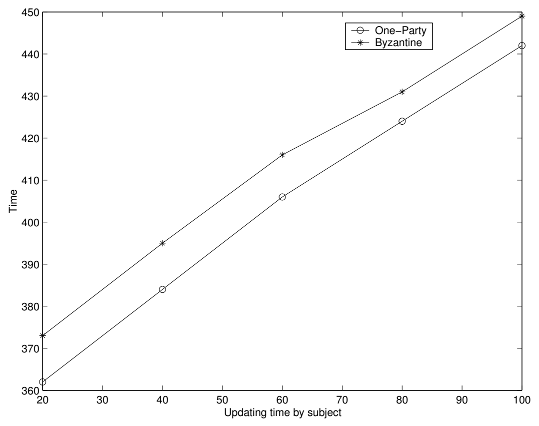


Figure 3.6. Time (no recovery)

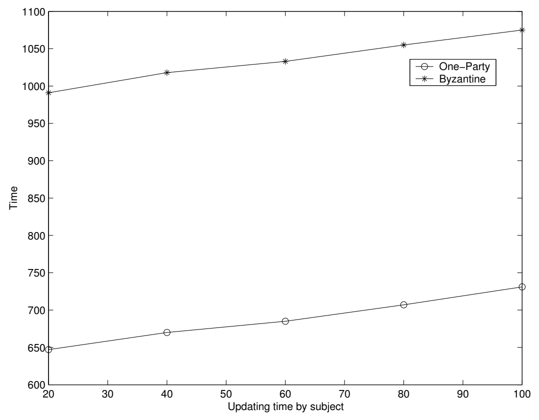


Figure 3.7. Time (with recovery)

the document. Encryption/Decryption are needed for the communication between the subjects and the delegates (one-party). This process also requires the signatures computation and verification, and transferring the recovered version to the requested subject. Thus, recovery almost doubles the time needed for both approaches. However, the Byzantine approach has more overhead relative to one-party case. For example, when the operation time is 20 ms, the Byzantine approach needs 991ms while one party needs 647ms. The overhead is almost 35%.

This overhead is mainly due to the encryption and decryption of recovered versions for recovery-merge operations and also the delegates sending back the merged version.

Furthermore, the subject needs to make delegates states stable by sending the merged version and signatures on it. We optimized performance by reducing the size of messages for recovery merge. If the first $b + 1$ recovered versions are the same, only one copy is sent to the delegates, along with all the digital signatures of the version. Similarly, a delegate only needs to send back the digital signature on the version that it approves. Figure 3.8 shows the results. The overhead is now reduced to only 18%.

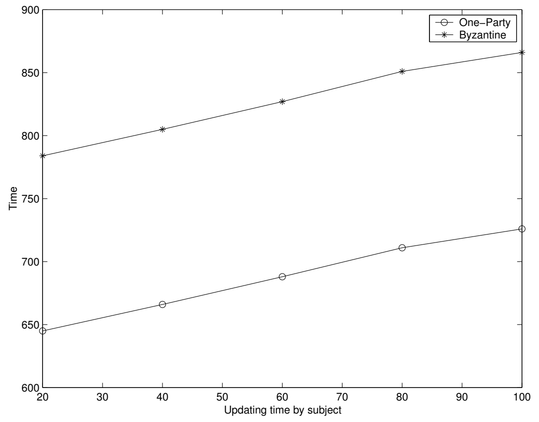


Figure 3.8. Time (with optimizations)

3.9 Conclusion and future work

In this chapter, we have proposed an approach supporting cooperative updates in Byzantine and failure prone distributed systems. The protocols we have developed are resistant to a number of colluding Byzantine subjects, and they are not affected by a maximum number of failures specified at the beginning. We have also developed a language and an infrastructure to specify dynamic paths, called flow policies, stating which subjects have to receive the document. In particular, we provide the possibility of modifying the specification of these flow policies during the update process, according to the stated modifications rules. Another important feature is the recovery process provided as part of our approach. Indeed, recovery is distributed, that is, the last correct version of the document content is built by a set of subjects, called delegates, using the document versions received by the contacted subjects in $\mathcal{C}G$.

We plan to extend the work presented in this paper in several directions. First, we want to develop protocols to manage rollback. Second, we plan to extend our work with mechanisms supporting receiver anonymity, and `add_element` and `add_attribute` privileges. We also plan to relax the constraint that subjects in $\mathcal{C}G$ cannot be changed once the update starts, which is currently a limitation of our approach. Finally, we will deal with how to allow different subjects to concurrently update disjoint XML document portions in a parallel distributed setting [56], giving the possibility of generating independent flow policies to be used on these portions.

4 A CRYPTOGRAPHIC APPROACH TO ACCESS CONTROL FOR PRIVACY PRESERVING COLLABORATIONS

In this chapter, we propose a cryptographic approach for distributed multi-party collaborations on document access and updating. Our solution ensures secure dynamic group communication, access privilege privacy, and participating anonymity. We provide security analysis for the protocol and time complexity analysis.

4.1 Introduction

Previous approaches we have developed are not scalable in that before a collaboration starts, each participants must receive some secret information securely. Therefore, each collaboration involves secret control information to be sent to each participant. Moreover, no previous approach addresses privacy in collaboration. By privacy, we means that in a distributed environment, each party's participation and its privileges should not be revealed to others unless necessary, during the collaboration. One simple approach to ensure privacy is by letting a trusted party to fully moderate the collaboration process. That is, a trusted party sends the data to the first participant, and after receiving the response, the trusted party sends the data to the next participant. However, the communication cost in this simple approach is high and more importantly, the trusted party is a bottleneck for collaborations.

In this chapter, we provide a scalable distributed protocol which uses cryptographic techniques to provide the following properties for multi-party collaborations:

1. *Dynamic group communications*: As a collaboration proceeds, the group members which are allowed to access the data can change dynamically.

2. *Data integrity*: A recipient should be able to verify that the data received is authentic, even though it receives the data from another party instead of from the trusted party;

3. *Participating anonymity and completeness*: A subject has no idea of who participates in the operation, except the one to whom it should send the data after it finishes the operation on it. However, when a subject receives the data, it should be able to verify that *all* its previous authorized subjects have accessed it already, even though it does not know whom they are. Any missed participation should be able to be detected by a receiver.

4. *Privacy on access privilege*: Each subject should know its access privilege, without leaking this information to other participants (except a recipient should know who authored the data, as this is for integrity purposes); Moreover, each subject may access data several times, and each time, the privileges may different from its previous privileges.

The rest of the chapter is organized as follows. We describe our model in Section 5.2 and preliminary notations in Section 4.3. The distributed privacy preserving document processing protocols are presented in Section 4.4. A security analysis is detailed in Section 4.5.

4.2 Model

We model the secure multi-party collaboration by a distributed document access and update. In such an application, several parties cooperatively update a document according to the access control policies and data flow policies. We assume that these policies are already defined for a document, for example, through multi-party negotiation processes. A participant may appear multiple times in a data flow policy. For each appearance, the corresponding access control policy specifies the access privileges. For simplicity, the privileges over the document are *read, write, and denied.* A *write* privilege consumes the *read* privilege, and the default privilege is denied.

Every party involved in the collaboration is assigned a secret ID, denoted as SID_i by the trusted party (TP). We assume that each party could be identified in public by its public key.

4.2.1 Threats

The threats to the collaboration include:

- an unauthorized party may access the document, and thus violates the confidentiality of the document.
- an unauthorized party may modify the document illegally. That is, a party which does not possess the write privilege from the access control policy may update the document.
- a party may be skipped during the collaboration and thus violates the participation completeness of the collaboration.

Other threats to secure multi-party collaboration are that a party, even an authorized party, may want to deduce the SID of the others who are participating in the collaboration and/or their access privileges.

4.3 Preliminary

In this section, we describe the concepts that are used in our protocol.

Definition 4.3.1 *A document id is identified for each collaboration. Therefore, we denote a collaboration as $G(id)$ where id is the document id used in the collaboration. A document version is an incremental number starting with 0 and incrementing every time a subject performs update privileges over it.*

Example 10 *A document with identifier* 1211 *will be accessed by A_1 to A_{12}. Suppose the data flow path is*

$$\mathbf{A_1}, A_2, A_3 \quad \mathbf{A_4} \quad A_5 \quad A_6 \quad A_7 \quad A_8 \quad \mathbf{A_9} \quad A_{10} \quad A_{11} \quad A_{12}$$

where the bold case participants (A_1, A_4 and A_9) have updated privileges, non-bold cased participants have read access, and others have denied access.

In the above example, document id is 1211, there are 3 version of its. Initial version is authored by A_1, then version 0 is updated by A_4 to be version 1, and the last version 2 is authored by A_9 during the collaboration.

Definition 4.3.2 *A session group $g_i(k, a, id)$ is formed by group members who have read access to the version i of a document id. The group member communicate with each other through a secret key k which is shared by the group members. The version of the document that they have read access is authored by the party a.*

In the example, $g_0(k_1, A_1, 1211) = \{A_1, A_2, A_3, A_4\}$. All members in g_0 have read access to the document modified by A_1. The data is encrypted with k_1 for secure group communication. A_4 is included since write privilege subsumes read access. Similarly, $g_1(k_2, A_4, 1211) = \{A_4, A_5, A_6, A_7, A_8, A_9\}$. After A_4 modifies the document, A_1, A_2 or A_3 is no longer able to access it, since neither of them is in g_1.

Therefore, we have the following

- A multi-party collaboration $G(id)$ consists of several $g_i(k_i, a_i, id)$;
- If $i \neq j$, then $k_i \neq k_j$; Each session group has a secret key for its group communication.
- For any $i > 1$, $g_i(k_i, a_i) \cap g_{i-1}(k_{i-1}, a_{i-1}) \supseteq a_i$. For any two sequence g, their interception contains the author of the last g.

For a collaboration $G(id)$, we have the following definitions.

- p: is a large prime from which a finite field F_p is formed.

- f: $\{0,1\}^* \rightarrow \{0,1\}^p$ is a one way hash function

- Dynamic collaboration polynomials (DCP)

  $$DCP = \{\gamma_1, \gamma_2, \ldots, \gamma_j\}$$

  where $\gamma_i = \langle P_{k_i}(x), P_{a_i}(x)\rangle$ and secret key polynomial $P_{k_i}(x)$ is a polynomial over $F_p[x]$ and is defined as

  $$P_{k_i}(x) = \prod_{j \in g_i(k_i, a_i, id)} (x - f(SID_j \| T_j \| r_1)) + k_i$$

  and authentication polynomial $P_{a_i}(x)$ is a polynomial over $F_p[x]$ and is defined as

  $$P_{a_i}(x) = \prod_{j \in g(k_i, a_i, id)} (x - f(SID_j \| T_j \| r_2)) + PubKey(a_i)$$

  T_j is the number of times the receiver j has received the document and r_1 and r_2 are random variables in F_p.

- A participant completeness polynomial (PCP) for verifying that all previous receivers have accessed the document

  $$PCP(x) = \prod_{i \in G(id)} (x - Q_i)$$

  where $Q_i = f(Q_{i-1} \| SID_{i-1} \| r_3)$ and r_3 is a random variable in F_p. Q_i is encrypted in the document;

- the next receiver set Φ for finding who is the next receiver.

  $$\Phi = \{\phi_1, \phi_2, \ldots, \phi_j\}$$

where $\phi_i = \langle \mu_i, \nu_i \rangle$, $\mu_i = f(SID_i \| r_5 \| T_i)$, $\nu_i = f(SID_i \| r_4 \| T_i) \oplus NextPubKey$ and $NextPubKey$ is the public key of the next receiver, r_4 and r_5 are random variables in F_p

4.4 Secure collaborative document processing

Before we detail the process, we first state our assumptions.

- A party which is involved in the collaboration will be available until the end of the collaboration.
- An authorized party will be collaborative. That is, it will send the document to the next receiver according to the protocol.

Before a collaboration starts, the trusted party publishes f, random variables $\{r_1, r_2, r_3, r_4, r_5\} \in F_p$, and DCP, PCP and Φ for each collaboration $G(id)$. A major difference with previous approaches is that no secret control information needs to be sent to each participant for each collaboration. Next we describe the collaboration protocols.

4.4.1 Updating process protocol

A party i could calculate possible secret keys used in the collaboration by calculating the polynomial $P_{k_i}(f(SID_i||T_i||r_1))$. When it receives a document, it performs the following steps:

1. It decrypts the document with key $\gamma_j.k_j$ such that $\gamma_j \in DCP$.
2. If the sender of the document is the trusted party, no integrity needs to be checked. Otherwise,
   - it verifies the integrity of the document according to the $\gamma_j.a_j$. That is, the receiver calculates the $PubKey_j = P_{a_i}(f(SID_i||T_i||r_2))$ and checks if

the digital signature signed by the $PubKey_j$ matches the hash value of the document content;

- it verifies if all previous participants have received the document by checking if $PCP(Q_i) = 0$ where Q_i is the value stored in the current version of the document;
- If there is any error, the participant sends the received data to the data server for recovery and its operation on the received document is done. We will detail the recovery process later.

3. The subject checks whether it has *write* privileges over the document. That is, it checks if there exist a $\gamma_i \in DCP$ such that polynomial $P_{a_i}(f(SID_i \| T_i \| r_2))$ equals to its public key. If so, it has *write* access to the data, and it will increases T_i by 1. Otherwise, it has *read* access.

   - If the subject has *read* access and the document it received is from the trusted party, however,the author of the document is another subject (this is the case when the trusted party recovered the document), then the subject finishes its operation on the received document. It does not need to send anything out;
   - Otherwise, the subject
     (a) updates value Q_{i+1} to be $f(Q_i \| SID_i \| r_3)$ in the document;
     (b) If the subject has *write* access, it modifies the content and should sign the message digit with its private key.
     (c) finds the next subject to send by computing the public key of the next; That is, it finds a ϕ_i such that $\mu_i = f(SID_i \| r_4 \| T_i)$. The next receiver is $\nu_i \oplus f(SID_i \| r_5 \| T_i)$.
     (d) If the subject only has *read* privileges this time over the document, it encrypts the document using the same key with which it decrypted

the document. Otherwise, it encrypts the document with the key k_i by calculating $P_{k_i}(f(SID_i||T_i||r_1))$.

4. The subject increases T_i by one for the document.

4.4.2 Recovery protocol

The trusted party keeps a record of session group from which the most recent recovery request has been requested. We denote this session group as g_r (note: at the beginning, g_r is null).

When the trusted party receives a recovery request from an authorized party p_c, it decides within which session group g_c the error happens this time by executing algorithm 4.1.

```
Algorithm: Find Group Session
Input: $p_c$, $doc$, $g_r$
Output: $g_r$

1. set $g_c = g_{(r+1)}$
2. if(valid(doc))
      Get from $doc$ the session group and assign it to $g_c$
3. for all $g_i \in G(id)$
      find $g_m$ s.t. $m \geq c \wedge p_c \in g_m \wedge \forall n$ s.t. $p_c \in g_n$, $n \geq m$
4. $g_r = g_m$
5. return $g_r$
```

Figure 4.1. Algorithm for finding session group

After finding out the session group of the recovery document, TP gets the document from the author of the g_r. The trusted party double checks if the next recipient of the author satisfied $PCP(Q_a) = 0$ and $PCP(f(Q_{a+1}||SID_{a+1}||r_3)) = 0$.

Then TP will update the Q_i values for the next $g_{(r+1)}$'s author, and broadcasts this to every members in the g_r.

4.5 Security analysis

Before we prove the security properties of the protocol, we first prove that SID is secure.

Theorem 4.5.1 *No party can derive any other one's SID during the process.*

Proof We prove this for the case that a party cannot derive its group member's SIDs. All other cases do not give the adversary such advantages.

When adversary i receives a document, it could derive the secret key used for the group g_i. Therefore, the adversary knows k_i, the secret key polynomial P_{k_i}. It could only derive the $\prod_{j \in g_i(k_i,a_i,id)} f(SID_j \| T_j \| r_1)$. Suppose there are only two parties in this session group. Then the adversary can at most get the value of $f(SID_j \| T_j \| r_1)$. As the f is an one-way function, it is hard to find SID_j. Therefore, no party can derive any other one's SID during the process. □

Next, we prove that the confidentiality and integrity of the document is ensured.

4.5.1 Confidentiality

In order to prove that the protocol ensures data confidentiality, it is enough to prove the following theorem.

Theorem 4.5.2 *If a party is not in $g_i(k_i, a_i, id)$, then it cannot get the secret key k_i used in the group communication.*

Proof We prove it according to the following cases

1. The adversary j is in $g_j(k_j, a_j, id)$ where $i \neq j$. That is, the adversary is authorized to participate in the collaboration, but in a different group. In this case, the secret key used to for session group i is mixed with the constant $\prod_{m \in g_i(k_i,a_i,id)} f(SID_m \| T_m \| r_1)$.

   The worst case is that the adversary and the participants in session group i were in the same session before, such as session g_b. In this case, the adversary could derive the value of $\prod_{m \in g_b(k_b, a_b, id)} f(SID_m \| T_m \| r_1)$ and therefore, to get the value of $\prod_{m \in g_b(k_b, a_b, id)^m \neq j} f(SID_m \| T_m \| r_1)$. However, in session i, the adversary needs to get the value of $\prod_{m \in g_b(k_b, a_b, id)^m \neq j} f(SID_m \| (T_m + 1) \| r_1)$. Since the f is a one way function and SID is secure according to our theorem, the adversary cannot derive the secret key of k_i. Therefore, the confidentiality is ensured.

2. The adversary is not authorized. In this case, an adversary has no advantage over the previous case, therefore, it cannot derive the secret key.

$\square$

Next we prove that the integrity of the document is ensured.

4.5.2 Integrity

We need to prove that no subject can modify the content issued by another authorized subject.

Theorem 4.5.3 *The protocol ensures data integrity.*

Proof This is ensured by authentication polynomials P_{a_i}. Each authorized participant could derive from the authentication polynomial the public key of the party which authored this version of the document. Based on the public key signature scheme, the party will sign the message digest of the content with its private key. No other party could derive this private key, and more over, the data is different for different versions of the document (because at least the Q value will be different for different versions of the document), therefore, the signature on the message digest cannot be forged or replayed. The data integrity is thus ensured. $\square$

4.5.3 Privacy

We need to prove that the protocol ensures access privilege privacy and participant privacy. We first prove the participant privacy.

Theorem 4.5.4 *The protocol ensures participant privacy.*

Proof According to our definition, participant privacy means that a participant cannot know who participates in the collaboration except its next receivers, previous senders, and authorizers of its session group document versions. □

Theorem 4.5.5 *The protocol ensures access privilege privacy.*

Proof We need to prove that a party cannot learn the privileges of participants in its group, except who authors the document. Moreover, a party cannot learn the privileges of participants that are not within its session groups;

Since our protocol ensures participant privacy, a receiver cannot learn who participated in its session group or other sessions, not to mention their privileges. When a party sends the document to the next receiver, the chance that it will know what kinds of privileges of the receive has does not increase than it does not know who is the next receiver. That is, the probability that the sender guesses whether the receiver has read or write access to the document is still the same. □

4.5.4 Participant completeness

Even though a participant does not know who the participants are, except its senders, next receivers, and authors of its session groups, it can verify that the *all* the previous participants have accessed the document.

Theorem 4.5.6 *The protocol ensures participant completeness.*

Proof If there is any participant left out, then the receiver can detect it by checking the participant completeness polynomial. Since in this polynomial, Q_i is based on the participants' SIDs and the current value of Q_i, it is impossible for any one to forge a Q_i, since according to our theorem, SID is secure and Q_i is aggregated. □

5 TIMELY DISSEMINATION OF CONFIDENTIAL EVENTS IN CONTENT-BASED PUBLISH/SUBSCRIBE SYSTEMS

In this chapter, we investigate confidentiality and availability problems in the application of content-based publish/subscribe (pub/sub) systems. We propose an efficient approach which increases event availability and ensures event confidentiality in large scale pub/sub systems. Instead of keeping the subscription information of the whole network, a broker only needs to keep the subscription information of its group, which is a small number of brokers. Therefore, not only are storage requirements reduced for each broker, but also time and network traffic are reduced for subscription information propagation. We propose a hierarchical event forwarding scheme. This scheme increases system availability by tolerating some broker failures. Additionally, our approach can efficiently determine the subscription groups to which an event has to be delivered by exploiting locality. Moreover, we propose an efficient encryption scheme, under which a broker encrypts an event only once. The encryption key can be efficiently derived by subscribers, even though they may belong to different subscription groups.

5.1 Introduction

Publish/Subscribe (pub/sub) systems provide a new distributed paradigm for event (message) distribution. In these systems, a publisher publishes an event through a broker, also called an event dispatcher. Subscribers specify their interests by registering with a broker. Brokers form a network in which they forward events to each other and, when needed, deliver events to subscribers which have registered with them.

Basically, there are two types of pub/sub systems. The first, referred to as *subject-based* or *type-based* pub/sub, is a system in which events are labeled with predefined subjects to which subscribers may subscribe. The second type, referred to as *content-based pub/sub* system, is more flexible and powerful than the subject-based one. In such a system, both subscriptions and content are specified with respect to a set of *attributes*. An attribute is an ordered pair of *name* and *type*. A subscriber subscribes to events by specifying *predicates* against attributes. For example, if a schema for a stock trade is (*company: string, price: integer, shares: integer*), a subscription could be: $(price < 20) \wedge company = \text{“IBM”}$. Because there are no explicit destination addresses associated with an event, brokers are responsible for delivering each event to subscribers whose subscriptions are satisfied by the event, which is called event *matching*. Decoupling publishers from subscribers makes the system scalable and powerful.

In this chapter, we focus on the issue in which confidentiality of events needs to be guaranteed and at the same time, events should be delivered on time, because their value decreases with time. Stock trading is one application where such issues are of paramount importance.

Meeting these two requirements can be contradictory, especially in large scale content-based pub/sub systems where the volume of published events is huge. To ensure confidentiality, an event should be encrypted during transmittal, so that only authorized subscribers are able to decrypt it. Usually, a group key shared by both the group members and the brokers is used to encrypt the event. However, since there could be many attributes and thus a large number of complex predicates, for n subscribers, there are possibly 2^n subscription groups that may be interested in an event. Therefore, encrypting the event with group keys could result in a significant performance cost and make the timely dissemination of events difficult.

A simple approach such as multicasting an event by the broker from which the event is published requires replicating subscription information at each broker. However, broker space requirements are a challenge for such approach.

Another consideration for these timely confidential event distribution applications is fault tolerance. A broker failure should not prevent subscribers from receiving events on time. System architecture proposed in [57,58] where an event is distributed along a spanning tree structure, may involve very expensive reconfigurations [59] if there is a broker failure.

Moreover, the system should minimize *registration information propagating time (RIP time)*, which is the time delay for new subscription information to be propagated into the network. For example, broadcasting an event to each broker, who then distributes the event to authorized subscribers registered from it, has minimal RIP time. Any newly accepted subscribers will get matched events. However, if the broker from which an event is published multicasts directly to authorized subscribers, then it takes time for new subscription to be propagated to each broker, especially when the network is large. Therefore, some newly accepted subscribers may miss some events.

Our contribution We propose an event forwarding scheme called hierarchical event routing. This scheme increases system availability by tolerating some broker failures. Additionally, our approach can efficiently determine the subscription groups to which an event has to be delivered by exploiting locality. We also propose an efficient encryption scheme, under which a broker encrypts an event only once. The encryption key can be efficiently derived by subscribers, even though they may belong to different subscription groups. In our solution, a broker needs only to keep the subscription information of its group, which is a small number of brokers. Therefore, not only are storage requirements reduced for each broker, but also time and network traffic are reduced for subscription information propagation. We provide theoretical proofs that our approach ensures event confidentiality. Experiment results validate the high event throughput of our approach.

The rest of the chapter is organized as follows. We describe our application model in Section 5.2, and then present our hierarchy event routing scheme in Section 5.3 and our event distribution scheme in Section 5.4. Experimental results and related work are presented in Sections 5.5 and 5.6, respectively. Section 5.7 concludes the chapter and outlines future work.

5.2 Model

In this chapter, we focus on how to ensure that confidential events are delivered on time to authorized subscribers. It is important to note that pub/sub systems involve several security and service quality issues. Here we only address part of them. Other issues, such as event access control policies, integrity and authentication, are not the focus of this chapter. Therefore, we assume that brokers are trusted. They will enforce event access control policies when subscribers subscribe from them; and they only accept events published by authorized publishers, and guarantee integrity of events they route.

In our system, a broker may fail, or come under DoS attacks, therefore it may not be available to deliver events. An unauthorized subscriber of an event (whose subscription is denied by brokers, or if accepted, the event does not match this subscriber's subscription) may want to access the event.

Note that even though our approach eliminates the matching performed by brokers while an event is forwarded among them, a matching algorithm is needed when a broker has to decide to which groups of subscribers an event should be delivered. However, such a matching algorithm is likewise not the focus of this chapter. We assume that such an algorithm exists and that locality is used in the algorithm for efficiency.

In the next two sections, we describe our schemes for event distribution. It includes two steps: the first one is that an event is routed from the broker from which the event is published to some brokers, and the second one is the event is forwarded from these brokers to authorized subscribers.

5.3 Hierarchial event routing scheme

In this section, we describe our event routing scheme among brokers, followed by a discussion of the main features of this scheme.

5.3.1 Hierarchy event routing

In our pub/sub system, all brokers are labeled with an ID, such labeling can be performed by the party responsible for accounting the services of the system.

Definition 5.3.1 *A leaf broker group (LBG) with label i is denoted as LBG_i where $LBG_i = \{b_{i1}, b_{i2}, \ldots, b_{im}\}$ such that*

- *brokers $b_{i1}, \ldots, b_{im} \in LBG_i$ are located closely in network topology;*
- *the ID of each broker in LBG has the same prefix as its group ID;*
- *the size of LBG_i is $|LBG_i|$ and $|LBG_i| \geq t$ where t is equal to $\lceil \frac{1}{1-r} \rceil$ and r is the estimated broker failure rate; and*
- *if a subscription request is submitted to and then approved by broker $b_{ij} \in LBG_i$, this subscription will be securely multicasted by b_{ij} to all other brokers in LBG_i.*

Therefore, any broker in a LBG maintains the subscription information of subscribers who are registered with any broker in the LBG.

Based on the labels of LBGs, a tree is formed where LBGs are the leaves and all inner nodes are formed by virtual broker group (VBG). Specifically, a VBG_i has label i and is virtually formed by either all LBG_i* or all VBG_i*.

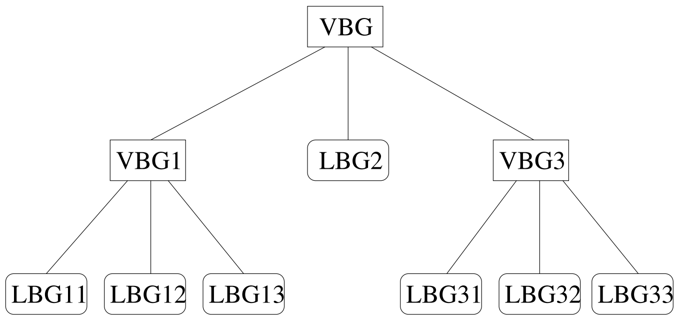


Figure 5.1. A 3-ary tree formed by broker groups

Example 11 *Figure 5.1 shows an example of 3-ary tree with a height of 2. The leaves are LBGs.* LBG_{11} *includes brokers* $b_{111}, b_{112}, b_{113}$ *and* b_{1113}. *All these brokers share the same group label prefix (11). Figure 5.2 illustrates the locality of brokers in Figure 5.1.*

Brokers in an *LBG* periodically authenticate each other and exchange their subscription information; however, a broker does not propagate its subscription information to another *LBG*.

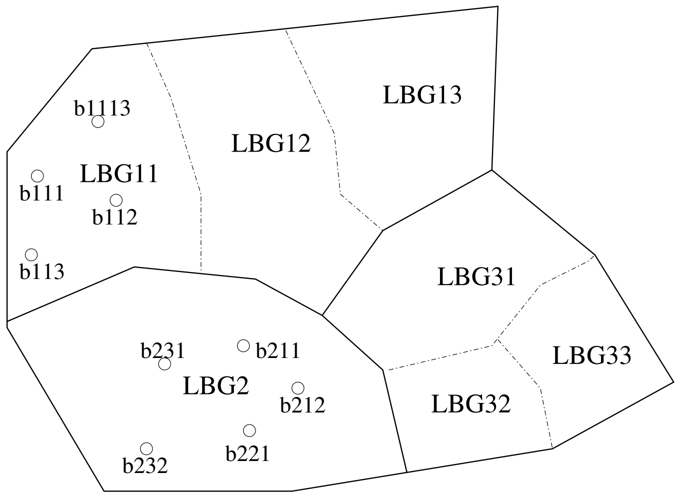


Figure 5.2. The locality of brokers

An additional information kept by each broker is a forwarding table. If a broker belongs to an LBG at depth h' of the tree, then the table is of dimension $h' \times (n-1)$. Each entry of such a table stores IP addresses of t brokers of other LBGs. Thus, if $r = 10\%$, each entry keeps information about 2 brokers. In such a table, each column has $n-1$ entries. For each of these entries, the label of the stored brokers shares the same prefix as the entry label. These brokers are randomly chosen from their group. Table 5.1 shows broker 111's table and Table 5.2 shows broker 213's table.

A broker periodically authenticates those brokers kept in its forwarding table and updates the information in case some brokers are under DoS attack or system failure.

Table 5.1
Event forwarding table of broker 111

2*(211, 221)	12* (122, 121)
3*(311, 322)	13* (133, 135)

Table 5.2
Event forwarding table of broker 213

1**(114, 132)
3**(312, 321)

Algorithm: Hierarchial Event Routing Scheme
Input: S, H, T
$//S$: ID of event sender, H: ID of host, T host's routing table of dimension $h' \times (n-1)$

```
1. if($S = H$)
     for($i = 1$ to $h'$)
         for ($j = 1$ to $n - 1$)
              choose a broker in entry $(i, j)$ and forwards the event;
         endfor
     endfor
2.else let p = max number of matched prefix of $S$ and $H$
        for($i = p + 1$ to $h'$)
            for($j = 1$ to $n - 1$)
                choose a broker in entry $(i, j)$ and forwards the event;
            endfor
        endfor
end
```

Figure 5.3. Hierarchy event forwarding algorithm

Algorithm 5.3 shows how a broker routes an event. Line 1-5 is the case where the broker is the one where the event is published. In this case, it routes the event to one broker in each of its entry in the routing table. Line 8-13 is the case where the broker receives the event from another broker. In this case, the broker uses the max common ID prefix with the sender to determine to which level it should start forwarding an event (Line 8), and then starts to forward the event to all entries from that level.

5.3.2 Discussion

We now discuss our routing scheme with respect to several metrics.

Broker Space Requirements: In most previous approaches, a broker must keep subscription information about the whole network. In large scale pub/sub systems, such a requirement implies that all subscription information is replicated at each broker. Under our approach, a broker only needs to maintain $1/n^h$ of the whole network's subscription information, where n and h are the degree and height of the tree, respectively. In a 4-ary tree with a height of 3, this is only 1/64 of the total network's subscribe information.

Subscription Information Update: In most previous approaches, new subscription may need to be propagated to the whole network. Our approach needs only to multicast such information to $1/n^h$ of the network, which greatly reduces network traffic.

Subscription Information Propagation Delay Time: Our approach decreases the delay time by propagating this information only within $1/n^h$ of the network.

Execution of Sequential Matching: Our approach requires only a single broker in a group to perform event matching, thus avoiding the execution of sequential matching which must be performed by brokers in most tree-based approaches.

Failure Tolerance: Our approach achieves the same level of fault tolerance as approaches based on event flooding; however, event delivery is faster in our approach since it takes $O(h)$ for an event to reach a leaf broker. By contrast, under the flooding approach it takes $O(n^h)$ for an event to reach a leaf broker.

Load Balancing: Most tree-based event delivery systems suffer from unbalanced loads. Leaf brokers in the tree seldom perform event matching and forwarding to other brokers, while brokers which are *centroids*[1] of the tree suffer from a heavy load. In our approach, the load is almost uniformly distributed among brokers, if a publisher publishes an event randomly at any broker. Our broker routing table ensures this property.

Matching Cost: Instead of letting one broker perform matching and event delivery to all subscribers in the system, in our approach, event matching and distribution are executed in parallel by n^h brokers, each supporting $1/n^h$ of the load. Also, our scheme possesses locality characteristics and therefore could use caching or popular group matching algorithms for efficient matching.

Dynamic: When a LBG i has been added too many brokers, such a group could be handled as a VBG which is formed by several LBGs and all brokers in previous i are divided into these LBGs. Or when a LBG contains too few brokers as some brokers have been deleted, such a group could be merged with another LBG.

After a broker finishes routing an event, it needs to distribute the event to authorized subscribers, if any exist. Next, we describe how a broker encrypts an event and distributes it to all the groups of authorized subscribers within its broker group.

5.4 Confidentiality-preserving event delivery

We assume that brokers accept subscription requests by following the policy of the system, such as requiring payment evidence. After a subscriber submits its subscription, if the request is permitted, the subscriber receives one or more keys corresponding to the groups in which its subscription falls. Since our focus is not on event space partitioning algorithms, we denote the event space as follows.

Let $\mathcal{G} = \{G_1, \ldots, G_n\}$ represent all event space of the subscription information in a broker group $\mathcal{B}$.

Group G_i is defined as (S_i, K_i, V_i) where S_i is part of the subscription space, K_i is the secret key shared by all brokers in $\mathcal{B}$ and these subscribers whose subscription

[1]A centroid of an n-node tree T is a node such that its removal from T leaves no connected components of size greater that $n/2$.

predicates p are satisfied by S_i, and V_i is a linked list of subscribers which belong to group G_i. Furthermore, K_i is in $\{0,1\}^l$ where l is a security parameter. For any $G_i, G_j \in \mathcal{G}$, $S_i \neq S_j$. However, we do not require that $S_i \cap S_j = \emptyset$

Note that because brokers in a group share subscription information, they will assign subscribers the same group and the corresponding key. In another word, all brokers in a group keep the same G_i.

Given an event e, a broker must first run a matching algorithm *match (e, $\mathcal{G}$)* which returns $\mathbf{G} \subseteq \mathcal{G}$, that is, a set of groups to which the event should be delivered. Next, the broker encrypts the event so that it can be decrypted only by the subscribers in these groups belong to $\mathbf{G}$.

Because an event may match several groups, encrypting the event several times with different group keys makes event delivery very inefficient. Here we propose an efficient and practical encryption scheme, which has the property that the encryption key is independent from the group keys. However, all authorized subscribers can derive the encryption key. Our strategy is the following:

Assume $\mathbf{G}= \{G_1, \ldots, G_m\}$ are the groups to which the event should be delivered. To encrypt an event, the broker

1. generates a random symmetric encryption key T such that T is in $\{0,1\}^l$ and a nonce r where $r \in \{0,1\}^l$;

2. encrypts the event with T;

3. calculates $A = ([G_1, D_1], \ldots, [G_m, D_m])$ such that $D_i = h(K_i \oplus r) \oplus T$ (for $i = 1, \cdots, m$), where $\oplus$ is XOR operation and h is a secure one-way function $h : \{0,1\}^l \rightarrow \{0,1\}^l$.

Then the broker multicasts the encrypted event to all subscribers in $\mathbf{G}$. In the encrypted event, the broker appends A and the nonce r. A member of group G_i can obtain T by $D_i \oplus h(K_i \oplus r)$ and thus decrypt the event.

5.4.1 Discussion

In this subsection, we discuss several important correctness properties of our approach. We first provide our correctness criteria and then prove that our encryption scheme satisfies them.

Definition 5.4.1 Completeness *Each authorized subscriber should be able to derive the key to decrypt an event.*

Definition 5.4.2 Soundness *If an individual is not authorized to access the event, then it is not able to decrypt the event. Further, a member* subj *of some group $G_i \in \mathbf{G}$, knowing T, r and A, is not able to derive any secret key K_j of group $G_j \in \mathbf{G}$ $(j \neq i)$, unless* subj *belongs to group G_j.*

Definition 5.4.3 Collusion-resilience *It is impossible for any set of adversaries to derive a secret key that none of them possesses.*

Theorem 5.4.1 *The proposed scheme is complete.*

Proof For an event, an authorized subscriber belongs to at least one of $G_i \in \mathbf{G}$ to which the event falls; thus, it has at least one key K_i. As a result, the subscriber can decrypt T as $D_i \oplus h(K_i \oplus r)$ from the public D_i and r, and the event too. $\square$

Theorem 5.4.2 *The proposed scheme is sound.*

Proof To decrypt an event, an individual must obtain the encryption key T. Since T is mixed in D_is, in order to obtain T, the individual must get at least one $h(K_i \oplus r)$. If the individual is an outsider or a system subscriber but does not belong to any of $G_i \in \mathbf{G}$, the individual cannot compute any of $h(K_i \oplus r)$s. Thus, the individual cannot obtain T.

Consider a subscriber belonging to $G_i \in \mathbf{G}$. Given the nonce r, A and T, the subscriber can obtain T by computing $H_i = h(K_i \oplus r)$ and then $D_i \oplus H_i$. Once knowing T, the subscriber can compute $H_j = D_j \oplus T$ (for any $j = 1, i-1, \ldots, i+1, m$).

However, due to the one-way property of h function, the subscriber cannot get K_j from $H_j(i.e., h(K_j \oplus r))$. Thus, the proposed scheme is sound. □

Theorem 5.4.3 *The proposed scheme is collusion-resilient.*

Proof We consider the collusion of subscribers not outsiders since involvement of outsiders cannot contribute any bit of information useful to reconstruct the keys. There are three possible collusion: by the subscribers not belonging to any $G_i \in \mathbf{G}$; by the subscribers belonging to some $G_i \in \mathbf{G}$; and by subscribers across several G_i's in $\mathbf{G}$. For the first case, these subscribers are equivalent to outsiders, and cannot get T or any K_i. For the second case, these subscribers can get their T and any of other $H_j = h(K_j \oplus r)$s. Because the security of the one-way function is independent of the number of users trying to break it, the collusion of these subscribers is no stronger than an individual subscriber trying to get K_j from H_j. The same principle applies to collusion of multiple subscribers from different groups.

As a result, the proposed scheme is resilient to any kinds of collusion attacks. □

5.4.2 Dynamics and rekeying

In a dynamic pub/sub system, the user can subscribe to or unsubscribe/be revoked from the system, even move from one group to the other. In order to guarantee that all (and only) current subscribers in a group G_i can obtain the events destined to the group, the group key K_i needs to be updated. The updated key needs to be distributed to the users securely. We propose using the same scheme a second time to address such a requirement. During user registration, the system will assign each user a unique personal secret ID (denoted as UID_i for user U_i). As a result, if a new K_i needs to be securely distributed to the current users $\{U_1, U_2, \cdots, U_n\}$ of a group G_i, the following message will be multicast (note: r is a new random nonce): $\{r; h(UID_1 \oplus r) \oplus K_i, h(UID_2 \oplus r) \oplus K_i, \cdots, h(UID_n \oplus r) \oplus K_i\}$.

As for the performance, the proposed scheme is very efficient. The computation of h depends totally on the algorithm selected and is independent of m, so its computation complexity can be considered as constant, such as $O(c)$ where c is a constant. The XOR operation is fast and can be considered to run in $O(1)$ time. Thus, time complexity of computing A is $O(mc)$ (i.e., $O(m)$).

As it can be seen from the discussion, the proposed scheme is secure, efficient, and dynamic and is able to enforce content confidentiality for pub/sub systems.

5.5 Simulation results

The simulations were conducted using a 5-ary tree of 3 levels, i.e., there are total 125 broker groups, with each group having 5 brokers. The total number of brokers in the system is 625.

Each broker has 50 subscribers. For simplicity, the event space is partitioned into units. Each subscriber subscribes to 1 unit from the event space.

We experimented with different distributions for event popularity and compared ours with a basic tree-based approach and a multicast-based approach:

- Uniform: Subscribers from any broker subscribe to event space units randomly.
- Normal distribution: Subscribers from a broker i subscribe to event units following a normal distribution. For an event space of n units, the subscription from broker i has a mean at the unit $n * (i + 0.5)/625$. We evaluate the effect of different standard deviations: 1/6 of the event space, 5 and 50. This follows from the fact that a subscription has locality characteristics: subscribers within a broker have similar distributions of interests, whereas the greater the distance between brokers, the more different the subscription distributions.

We present our results on space requirements for a broker, and on the delay in receiving an event by subscribers.

5.5.1 Space Requirements

We measure the space requirement for storing subscription information by a broker. IP addresses are 4 bytes long and event space is 2 bytes. For each event unit, if there is at least one subscription, then it forms a subscription group. For each group, a linked list is used to store subscribers information. Figure 5.4 shows the result of subscriptions which are uniformly distributed. It reveals that a multicast approach requires a dramatically large space. When comparing our approach with the tree-based approach in detail, our approach uses a relatively larger space than a tree-based approach (see Fig. 5.5). We think this is an acceptable space requirement (less than 1.4 KB). The results for normal distributions are reported in Fig. 5.6, 5.7, 5.8, 5.9, 5.10, 5.11. They follow similar patterns.

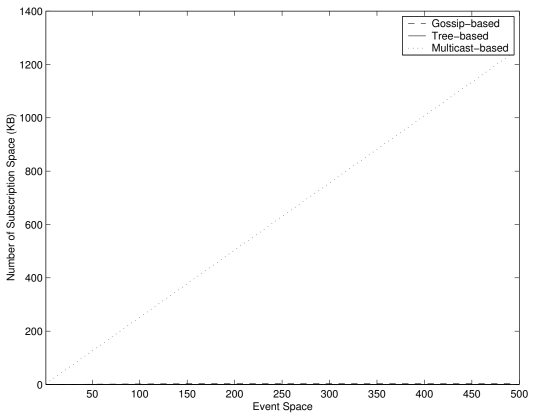


Figure 5.4. Space requirement [1] for a broker (uniform)

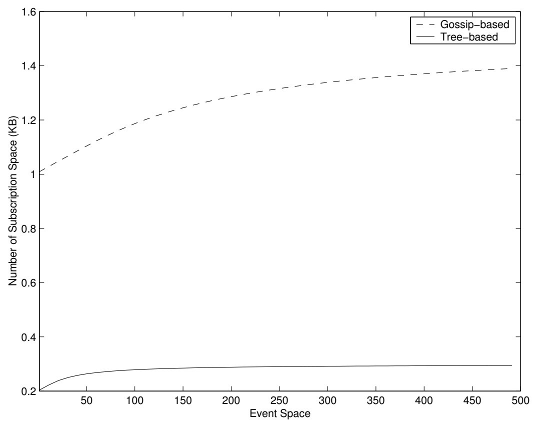


Figure 5.5. Space requirement [2] for a broker (uniform)

Next, we check the number of subscription groups formed under different subscription distributions. These results are reported in Figure 5.12, 5.13, 5.14, 5.15, 5.16, 5.17. In multicast approach, the number of groups is the same as the number of event space units. When the standard deviation decreases, our approach has a similar number of groups as the tree-based approach. For a tree-based approach, the routing

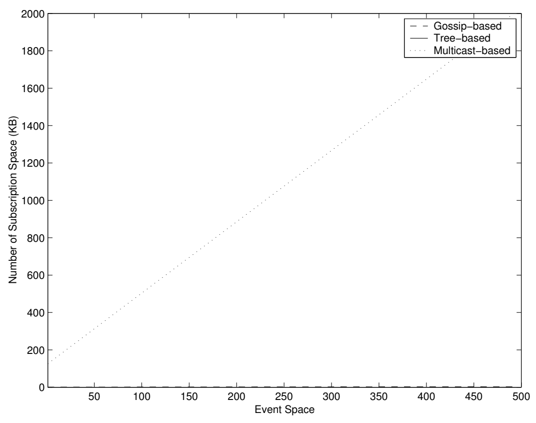


Figure 5.6. Space requirement [1]($\sigma = 1/6$ of event space)

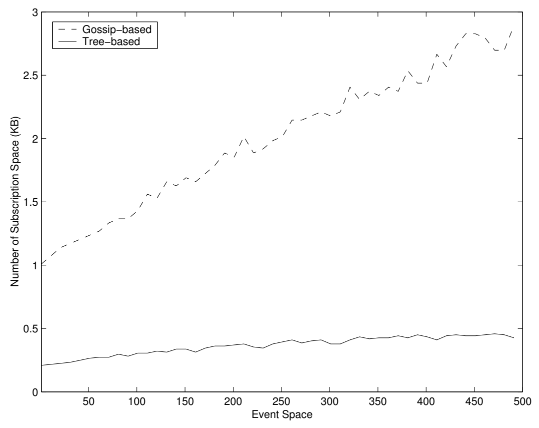


Figure 5.7. Space requirement [2] ($\sigma = 1/6$ of event space)

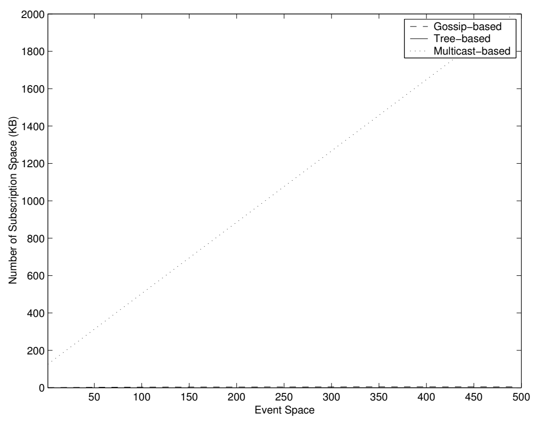


Figure 5.8. Space requirement [1] for a broker (σ =50)

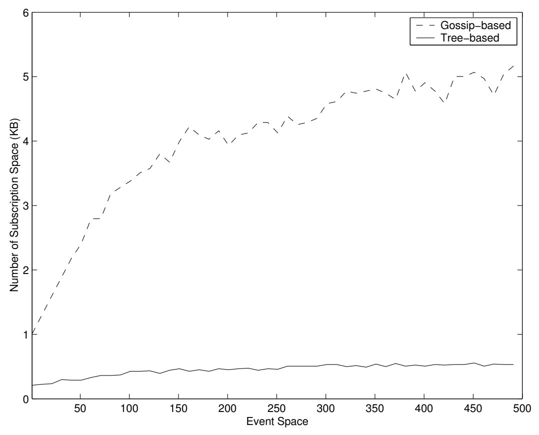


Figure 5.9. Space requirement [2] for a broker (σ =50)

information is aggregated, therefore, a broker only needs to maintain the subscription information from its subscribers. Even for normal distribution with large deviation, we found that a broker can aggregate these brokers' subscription information into a very small space.

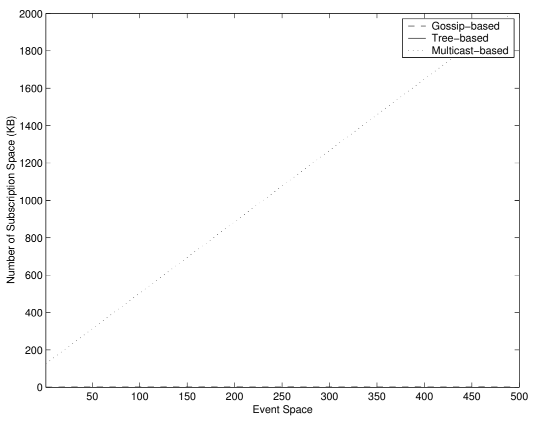


Figure 5.10. Space requirement [1] ($\sigma = 5$)

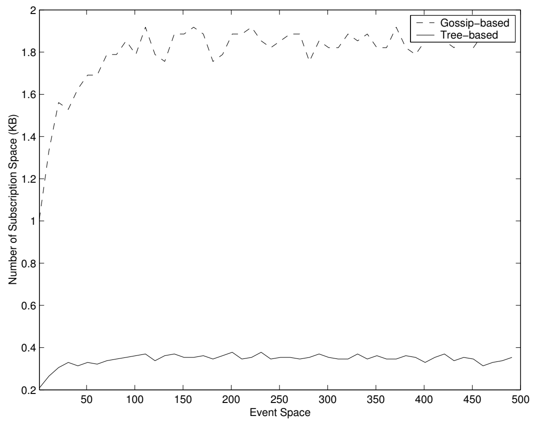


Figure 5.11. Space requirement [2] (σ =5)

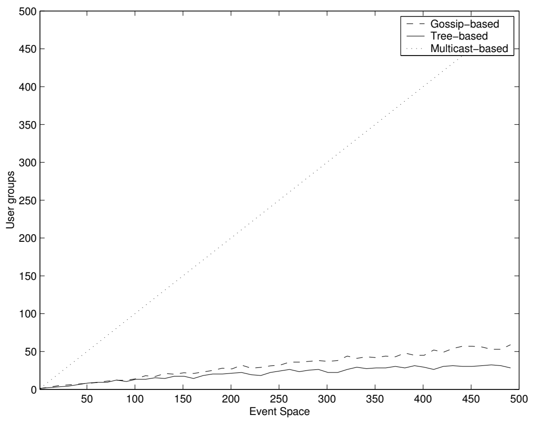


Figure 5.12. Number of subscription groups [1] (σ = 1/6 of event space)

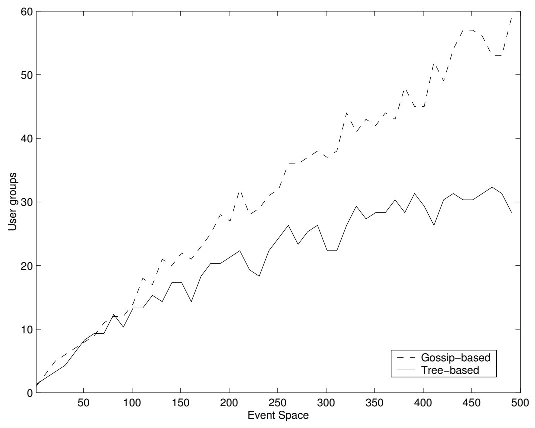


Figure 5.13. Number of subscription groups [2] (σ = 1/6 of event space)

5.5.2 Time Delay

We evaluate the time delay for a subscriber to receive an event. In the evaluation, it takes 1 unit of time to forward an event to a group. Figure 5.18 reports the time to deliver an event in a uniform distribution. The tree based approach takes more time

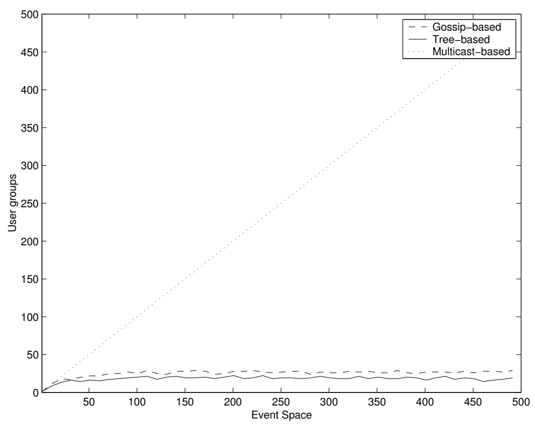


Figure 5.14. Number of subscription groups [1] ($\sigma = 5$)

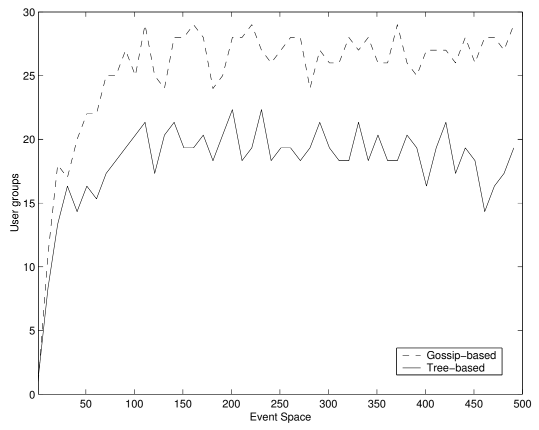


Figure 5.15. Number of subscription groups [2] ($\sigma = 5$)

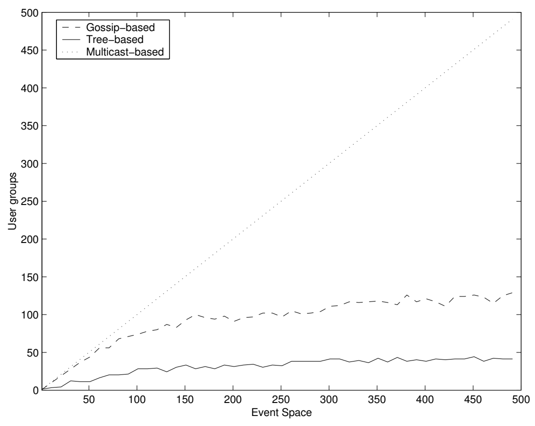


Figure 5.16. Number of subscription groups [1] (σ =50)

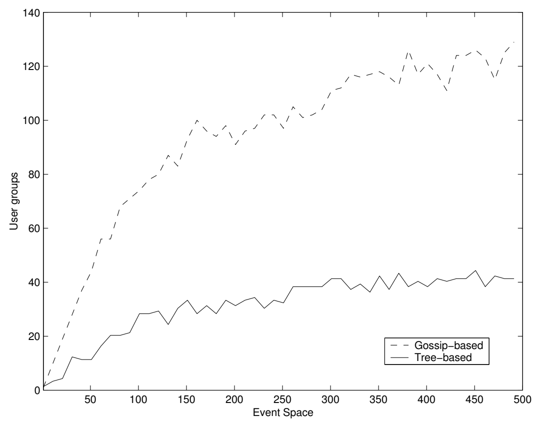


Figure 5.17. Number of subscription groups [2] (σ =50)

due to the series of event forwarding by brokers in such a structure. By contrast, multicasting needs to forward to all subscription groups of the whole network. Our hierarchy approach, however, uses parallel event forwarding, which is very efficient. Figure 5.18 also shows that when the number of subscription groups increases, the performance of a multicast approach degrades; it is slower than the tree-based approach when the event space size reaches 350. Figures 5.19, 5.20, 5.21 report the results for normal distributions. The tree-based approach is relatively stable regardless of the standard deviation. When the standard deviation increases, our approach takes a longer time.

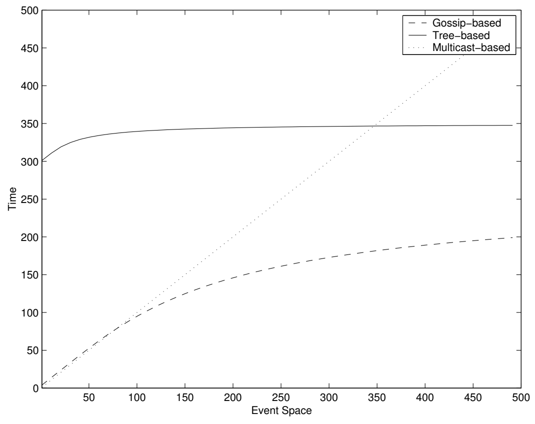


Figure 5.18. Time delay (uniform)

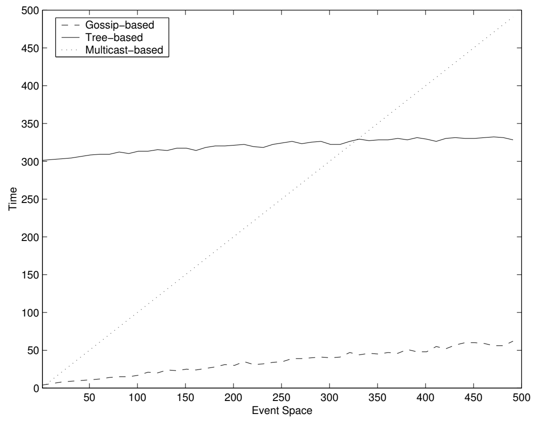


Figure 5.19. Time delay (σ = 1/6 of event space)

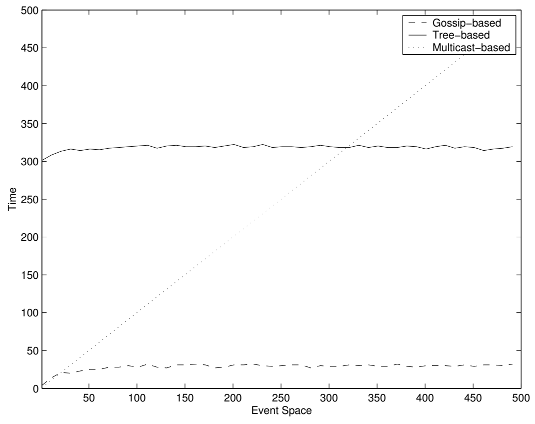


Figure 5.20. Time delay (σ =5)

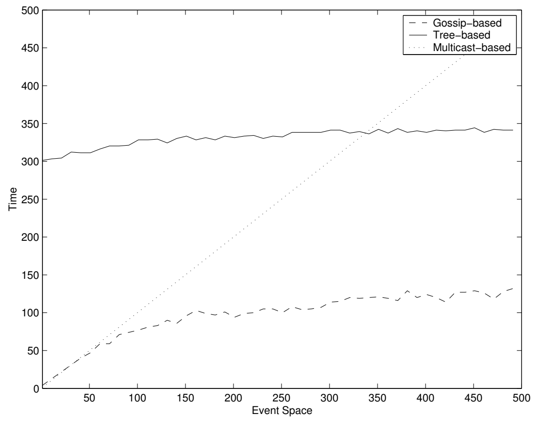


Figure 5.21. Time delay (σ =50)

5.5.3 Broker Involvement

The number of brokers involved in forwarding an event reflects the load that a broker takes. Figure 5.22 shows the number of brokers that are involved in delivery of an event. The subscription rate of events is uniformly distributed. The horizontal axis shows the subscription rate of each *broker*. The vertical axis shows the number of brokers involved in delivering an event. For the tree-based approach, a spanning tree is built by applying Dijkstra's algorithm on a graph of 625 nodes, each node with degree d where d is uniformly distributed from 1 to 4. The result is the average of ten runs, where at each run, an event is published at a randomly chosen broker. We also present the ideal approach, where only the brokers subscribing to an event are those which participate in the delivery of the event. From Fig. 5.22, the tree-based approach needs more brokers to participate in the event delivery, especially when subscription rate is 20%, half of the brokers involved are not interested in an event. As the subscription rate increases, the tree-based approach reaches the same number as the ideal approach at 100%. In that instance, all brokers in a tree-based approach are interested in the event.

Hierarchical event forwarding uses an almost constant number of brokers for forwarding an event. If there is no broker failure, this number is equal to the number of broker groups and it is not related to the subscription rate. Therefore, when the subscription rate is below 20%, our approach needs more brokers than the ideal approach. Our approach uses less brokers than the tree-based approach when the subscription rate is above 6%. For a broker with 50 subscriptions, this means an event has popularity below 0.12%. Thus, our approach seldom uses more brokers than the tree-based approach.

5.6 Related work

Several studies have been devoted to investigating efficiency issues concerning pub/sub systems [1, 3, 58–61, 61–70] and several prototype systems have been devel-

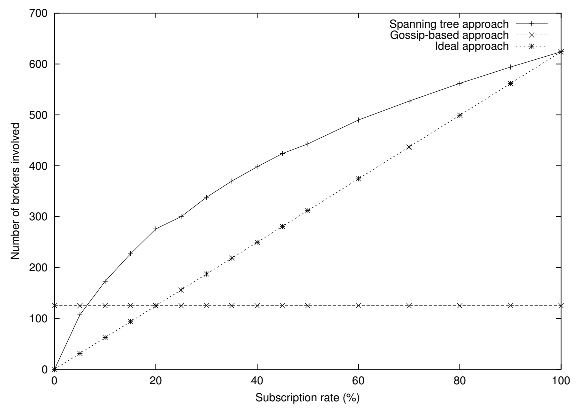


Figure 5.22. The number of brokers involved

oped. Most approaches like [58,59,66] use a spanning tree structure for event routing. In order to reduce the matching that has to be performed by brokers from the root to the leaves, several optimization techniques have been proposed. Virtual groups are used to reduce the matching performed by brokers [57].

However, security issues [65] in content-based pub/sub systems have not been so widely investigated. Srivatsa and Liu [1] propose a resilient network which, instead of providing only a single path from each publisher to its subscribers which is inherited from the spanning tree structure, several independent paths from a publisher to each of its subscribers are provided. Such paths are built deterministically. In their approach, building several independent paths from a publisher to every subscriber involves complex topology computations. In dynamic environments, such computation is expensive. Such expensive reconfigurations of tree structures have been completely eliminated in our hierarchy event forwarding scheme. Each broker maintains a forwarding table which ensures that at least one broker in the next forwarding level is operative.

To avoid unnecessary event broadcasting, Carzaniga et al. [58] proposes an approach that broadcasts events only along the spanning tree. As previously mentioned, in dynamic environments, a tree structure is hard to maintain and may become disconnected. Broadcasting along a disconnected tree involves more redundancy and does not solve the availability problem.

Opyrchal and Prakash [71] discuss how a broker can encrypt an event and deliver it to a possibly very large number of groups. As each group has a secret key shared by members and brokers, encrypting the event using a group key may involve performing many encryption operations, and there may be several groups to which this event should be delivered. Caching and clustering are therefore used to make fewer encryptions. In our confidentiality-preserving encryption scheme, an event is encrypted only once. All authorized subscribers can derive the key for decryption the event efficiently. Our scheme is provably secure.

5.7 Conclusions and future work

In this chapter, we address some security issues of content-based pub/sub systems. We focus on increasing the availability of events and ensuring confidentiality when events are delivered to authorized subscribers. Our schemes (hierarchical event forwarding and confidentiality-preserving encryption) are efficient and scalable. Our approach is especially suitable for large-scale content-based pub/sub systems. Simulation results validate the efficiency of our approach.

We plan to investigate other security issues in content-based pub/sub systems, especially how brokers can efficiently authenticate each other, and how to ensure confidentiality of events forwarded among brokers. We would like to investigate an efficient mechanism for matching an encrypted event against subscriptions, which is suitable for large-scale content-based pub/sub systems.

LIST OF REFERENCES

LIST OF REFERENCES

[1] M. Srivatsa and L. Liu. Securing publish-subscribe overlay services with eventguard. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS '05)*, 2005.

[2] Y. Diao, S. Rizvi, and M. J. Franklin. Towards an internet-scale XML dissemination service. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*, 2004.

[3] F. Cao and J. P. Singh. Efficient event routing in content-based publish-subscribe service networks. In *Proceedings of IEEE INFOCOM '04*, 2004.

[4] M. J. Freedman, E. Freudenthal, and D. Mazires. Democratizing content publication with coral. In *Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04)*, 2004.

[5] T. Li, Y. Wu, D. Ma, H. Zhu, and R. H. Deng. Flexible verification of MPEG-4 stream in peer-to-peer CDN. In *Proceedings of the 6th International Conference on Information and Communications Security (ICICS)*, 2004.

[6] G. Berhe, L. Brunie, and J. Pierson. Modeling service-based multimedia content adaptation in pervasive computing. In *Proceedings of ACM International Conference on Computing Frontiers*, 2004.

[7] A. Fox, S. D. Gribble, Y. Chawathe, and E. A. Brewer. Adapting to network and client variation using active proxies: Lessons and perspectives. *IEEE Personal Communications*, 1998.

[8] V. Cardellini, P. S. Yu, and Y. W. Huang. Collaborative proxy system for distributed web content transcoding. In *Proceedings of the 9th ACM International Conference on Information and Knowledge Management*, 2000.

[9] C. Chi and Y. Wu. An XML-based data integrity service model for web intermediaries. In *Proceedings of the 7th International Workshop on Web Content Caching and Distribution*, 2003.

[10] C. Chi, Y. Lin, J. Deng, X. Li, and T. Chua. Automatic proxy-based watermarking for WWW. *Computer Communications*, 24(2):144–154, 2001.

[11] B. Thuraisingham, A. Gupta, E. Bertino, and E. Ferrari. Collaborative commerce and knowledge management. *Knowledge and Process Management*, 9(1):43–53, 2002.

[12] P. Maglio and R. Barrett. Intermediaries personalize information streams. *Communications of the ACM*, 43(8):99–101, 2000.

[13] J-L. Huang, M-S. Chen, and H-P. Hung. A QoS-aware transcoding proxy using on-demand data broadcasting. In *Proceedings of the IEEE INFOCOM*, 2004.

[14] Y. Koglin and E. Bertino. Secure content services from cooperative internet intermediaries. Manuscript, 2005.

[15] S. Chandra and C. S. Ellis. JPEG compression metric as a quality aware image transcoding. In *Proceedings of USENIX 2nd Symposium on Internet Technology and Systems*, 1999.

[16] R. Han, P. Bhagwat, R. LaMaire, T.Mummert, V. Perret, and J. Rubas. Dynamic adaptation in an image transcoding proxy for mobile web browsing. *IEEE Personal Communications*, 5(6):8–17, 1998.

[17] Y. Koglin, G. Mella, E. Bertino, and E. Ferrari. An update protocol for XML documents in distributed and cooperative systems. In *Proceedings of International Conference On Distributed Computing Systems*, 2005.

[18] J. Apostolopoulos. Secure media streaming & secure adaptation for non-scalable video. Technical Report HPL-2004-186, Hewlett-Packard Laboratories, 2004.

[19] S. Wee and J. Apostolopoulos. Secure transcoding with JPSEC confidentiality and authentication. Technical Report HPL-2004-185, Hewlett-Packard Laboratories, 2004.

[20] S. Wee and J. Apostolopoulos. Secure scalable streaming enabling transcoding without decryption. In *IEEE International Conference on Image Processing*, 2001.

[21] S. Wee and J. Apostolopoulos. Secure scalable video streaming for wireless networks. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2001.

[22] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. Caching on the world wide web. *Knowledge and Data Engineering*, 11(1):95–107, 1999.

[23] B. Li, X. Deng, M. J. Golin, and K. Sohraby. On the optimal placement of web proxies in the internet. In *Proceedings of the IEEE INFOCOM*, 1999.

[24] S. Sivasubramanian, M. Szymaniak, G. Pierre, and M. V. Steen. Replication for web hosting systems. *ACM Computing Surveys*, 36(3):291–334, 2004.

[25] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *Proceedings of the IEEE INFOCOM*, 1999.

[26] S. Buchholz and A. Schill. Adaptation-aware web caching: Caching in the future pervasive web. In *Proceedings of the 13th GI/ITG Conference Kommunikation in Verteilten Systemen (KiVS)*, 2003.

[27] C. Lesniewski-Laas and M. F. Kaashoek. SSL splitting: Securely serving data from untrusted caches. In *Proceedings of the 12th USENIX Security Symposium*, 2003.

[28] B. Awerbuch, B. Patt-Shamir, D. Peleg, and M. Tuttle. Effective collaboration without trust in peer-to-peer systems. Manuscript, 2004.

[29] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in a constant number of rounds of interaction. In *Proceedings of the 8th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, 1989.

[30] D. Boneh, G. Durfee, and M. Franklin. Lower bounds for multicast message authentication. In *Advances in Cryptology — EUROCRYPT'01*, 2001.

[31] J. Camenisch, J. Piveteau, and M. Stadler. An efficient fair payment system. In *Proceedings of the 3rd ACM Conference on Computer and Communicatons Security*, 1996.

[32] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Advances in Cryptology — CRYPTO'03*, 2003.

[33] A. Chan, Y. Frankel, P. MacKenzie, and Y. Tsiounis. Mis-representation of identities in E-cash schemes and how to prevent it. In *Advances in Cryptology — ASIACRYPT'96*, 1996.

[34] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In *Advances in Cryptology — EUROCRYPT'96*, 1996.

[35] E. Bertino, E. Ferrari, and G. Mella. An XML-based approach to document flow verification. In *Proceedings of the 7th International Conference on Information Security(ISC04)*, 2004.

[36] E. Bertino, E. Ferrari, and G. Mella. An approach to cooperative updates of XML documents in distributed systems. *Journal of Computer Security*, 13(2):191–242, 2005.

[37] E. Bertino, G. Correndo, E. Ferrari, and G. Mella. An infrastructure for managing secure update operations on XML data. In *SACMAT'03*, 2003.

[38] Extensible markup language (XML). Available at: http://www.w3.org/XML/.

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

[40] C. G. Pollmann. The XML security page, 1999. Availabe at: http://www.dcs.uni-siegen.de/geuer-pollmann/xml_security.html.

[41] I. Tatarinov, Z. G. Ives, A. Y. Halevy, and D. S. Weld. Updating XML. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, 2001.

[42] C.H. Lim, S. Park, and S.H. Son. Access control of XML documents considering update operations. In *Proceedings of the ACM Workshop on XML Security*, 2003.

[43] B. Kane, H. Su, and E. Rundensteiner. Consistently updating XML documents using incremental constraint check queries. In *Proceedings of the 4th ACM CIKM International Workshop on Web Information and Data Management (WIDM'02)*, 2002.

[44] R. Vitenberg, I. Keidar, G. Chockler, and D. Dolev. Group communication specifications: A comprehensive study. *ACM Computing Surveys*, 33(4):1–43, 2001.

[45] M. K. Reiter. A secure group membership protocol. *IEEE Transactions on Software Engineering*, 22(1):31–42, 1996.

[46] M. K. Reiter. Secure agreement protocols: Reliable and atomic group multicast in rampart. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, 1994.

[47] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[48] D. Malkhi and M. K. Reiter. Byzantine quorum systems. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, 1997.

[49] D. Malkhi, Y. Mansour, and M. K. Reiter. On diffusing updates in a Byzantine environment. In *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems*, 1999.

[50] D. Malkhi, M. K. Reiter, O. Rodeh, and Y. Sella. Efficient update diffusion in Byzantine environments. In *Proceedings of the 20th IEEE Symposium on Reliable Distributed Systems*, 2001a.

[51] D. Malkhi, M. K. Reiter, A. Wool, and R. N. Wright. Probabilistic quorum systems. *The Information and Computation Journal*, 170(2):184–206, 2001b.

[52] E. Bertino, S. Castano, and E. Ferrari. On specifying security policies for web documents with an XML-based language. In *Proceedings of the 1st ACM Symposium on Access Control Models and Technologies*, 2001.

[53] XML Path Language (Xpath). Availabe at: http://www.w3.org/TR/xpath.

[54] E. Bertino and E. Ferrari. Secure and selective dissemination of XML documents. *ACM Transactions on Information and System Security (TISSEC)*, 5(3):290–331, 2002.

[55] G. Mella. *Distributed and Cooperative Updates of XML Documents*. PhD thesis, University of Milano, DICO Department, Milano, Italy, December 2004. Availabe at: http://homes.dico.unimi.it/dbandsec/mellagiovanni.

[56] Y. Koglin, G. Mella, E. Bertino, and E. Ferrari. An update protocol for XML documents in distributed and cooperative systems. In *Proceedings of the 25th International Conference on Distributed Computing Systems*, 2005.

[57] R. Zhang and Y. C. Hu. HYPER: A hybrid approach to efficient content-based publish/subscribe. In *In Proceedings of International Conference on Distributed Computing Systems*, 2005.

[58] A. Carzaniga, M. J. Rutherford, and A. L. Wolf. A routing scheme for content-based networking. In *Proceedings of the IEEE INFOCOM*, 2004.

[59] G. Cugola, D. Frey, A. L. Murphy, and G. P. Picco. Minimizing the reconfiguration overhead in content-based publish-subscribe. In *Proceedings of the 19th ACM Symposium on Applied Computing (SAC04)*, 2004.

[60] A. K. Datta, M. Gradinariu, M. Raynal, and G. Simon. Anonymous publish/subscribe in P2P networks. In *Proceedings of the International Parallel and Distributed Processing Symposium*, 2003.

[61] G. Banavar, T. Chandra, B. Mukherjee, and J. Nagarajarao. An efficient multicast protocol for content-based publish subscribe systems. In *Proceedings of the 19th International Conference on Distributed Computing Systems (ICDCS)*, 1999.

[62] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *Symposium on Principles of Distributed Computing*, 1999.

[63] A. Riabov, Z. Liu, J. L. Wolf, P. S. Yu, and L. Zhang. Clustering algorithms for content-based publication-subscription systems. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems,*, 2002.

[64] A. Carzaniga and A. L. Wolf. Forwarding in a content-based network. In *Proceedings of ACM SIGCOMM*, 2003.

[65] C. Wang, A. Carzaniga, D. Evans, and A. L. Wolf. Security issues and requirements for internet-scale publish-subscribe systems. In *Proceedings of Hawaii International Conference on System Sciences*, 2002.

[66] P. Costa and G. P. Picco. Semi-probabilistic content-based publish-subscribe. In *Proceedings of the 19th International Conference on Distributed Co mputing Systems (ICDCS)*, 2005.

[67] G. P. Picco, G. Cugola, and A. L. Murphy. Efficient content-based event dispatching in presence of topological reconfigurations. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS03)*, 2003.

[68] H. Zhou and S. Singh. Content based multicast (CBM) in ad hoc networks. In *Proceedings of MobiHoc*, 2000.

[69] P. Costa, M. Migliavacca, G. P. Picco, and G. Cugola. Epidemic algorithms for reliable content-based publish-subscribe: an evaluation. In *Proceedings of the 19th International Conference on Distributed Co mputing Systems (ICDCS)*, 2004.

[70] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transaction on Computer Systems*, 19(3):332–383, 2001.

[71] L. Opyrchal and A. Prakash. Secure distribution of events in content-based publish subscribe systems. In *Proceedings of the 10th USENIX Security Symposium*, 2001.

VITA

VITA

Yunhua Koglin (*née* Lu) was born in Laiyang City, Shandong Province, China. She obtained her bachelor's degree at the Chengdu University of Technology. In May 2003, she obtained her master degrees in Computer Science and Engineering Geology from Purdue University. She received the degree of Doctor of Philosophy in December 2006 under the direction of Professor Elisa Bertino. Her research interests are computer security, applied cryptography, distributed systems and privacy enhancement techniques.