# Distributed Access Control For XML Document Centric Collaborations

Mohammad Ashiqur Rahaman
SAP Research
805, avenue Dr. Maurice du Donat
06250, Mougins, France
mohammad.ashiqur.rahaman@sap.com

Yves Roudier
EURECOM
2229, route des Crêtes
06560 Valbonne, France
Yves.Roudier@eurecom.fr

Andreas Schaad
SAP Research
Vincenz-Priessnitz-Str. 1, 76131, Karlsruhe
Germany, +49/62 27/78-43082
andreas.schaad@sap.com

## Abstract

*This paper introduces a distributed and fine grained access control mechanism based on encryption for XML document centric collaborative applications. This mechanism also makes it possible to simultaneously protect the confidentiality of a document and to verify its authenticity and integrity, as well to trace its updates. The enforcement of access control is distributed to participants and does not rely on a central authority.*

*Novel aspects of the proposed framework include the adoption of a decentralized key management scheme to support the client-based enforcement of the access control policy. This scheme is driven by the expression of access patterns of interest of the participants over document parts to determine the keys required. A lazy rekeying protocol is also defined to accommodate the delegation of access control decisions that in particular reduces rekeying latency when faced with the addition and removal of participants.*

## 1. Introduction

Digital documents are an increasingly central concern in today's inter organizational exchanges and collaboration processes, as illustrated by the multiplication of XML standards for instance. In many cases, such documents are composite in that they are originated by multiple authorities in charge of their own portion of the document ruling who may read or edit fine grained parts of it. The document edition process is becoming increasingly collaborative with participants joining and leaving the collaboration in particular because of mobility or churn. This increasing complexity has been accompanied by the need to delegate access control to handle situations in which an authority is unavailable. Furthermore, the participants may exchange documents arbitrarily, thus raising various security issues like integrity, confidentiality, authenticity, or forward and backward secrecy [11].

Centralized XML access control, which has been vastly studied (see for instance [3, 7, 6, 10, 13, 20, 18]), relies on an authority that maintains document repositories and enforces access control on a per request basis. A distributed setting pleads for new models of access control in which access control specification is decoupled from its asynchronous enforcement.

We describe in this paper[1] basic mechanisms for enabling a collaboration framework that allows a selective, distributed, and flexible accesses to be made on document nodes and traced. Our approach is document driven and in particular aims at controlling the access to a document through the encryption of its parts with keys shared by a group of participants with similar access rights. It makes use of a tree-based group cryptographic technique which we adapt for fine grained and multi-authority access control. Our technique in particular limits the scope of rekeying when participants dynamically join or leave.

The paper is organized as follows. Section 2 introduces a collaboration edition scenario on XML documents and the scope and objectives of our solution. Section 3 provides an overview of our solution. Access control policies and their relationships with access patterns expressed by participants are discussed in Section 4. Section 5 introduces our key management scheme and the lazy rekeying approach. Section 6 describes the structure and usage of protected documents. Section 7 discusses the security of our controlled document edition protocol. Section 8 finally compares the scheme presented in this paper with related work, followed by a conclusion.

## 2. Motivating Example

European Union (EU) administrative bodies, Europol and Eurojust, and the associated law enforcement authorities of 27 member states [5] collaborate whenever there is an occurrence of cross border organized crime. Europol and Eurojust have representatives, respectively a Europol National Member and a Eurojust National Member, for each of the 27 member states. Each member state has its national contact points (National Authority) for Europol and Eurojust. This collaboration entails a request for Mutual Legal Assistance (MLA). In such cases, participants collaboratively define and work on a document called European Arrest Warrant (*EAW*), as follows:

1. A Europol National Unit of country A ($ENUA$) makes a written request of assistance (for a witness protection) to a Eurojust National Member of country A ($EJNMA$).
2. The $EJNMA$ opens a Temporary Work File (Twf) in a local Case Management System ($CMS$).
3. The $EJNMA$ contacts Eurojust National Member of country B ($EJNMB$) by forwarding the request of assistance.
4. The $EJNMB$ contacts the responsible national authority of country B ($NAB$). Steps are taken by the responsible $NAB$ to provide the requested assistance.

Figure 1 depicts a simplified *EAW* document instance composed of four document parts instances. It distinguishes the ownership of different parts of the *EAW* document by dotted rectangles around subtrees. *ENU* is the owner of the subtree rooted at `ENU`. Similarly *EJNM* and *NA* are the owners of the subtrees rooted at nodes `EJNM` and `NA` respectively. The *EAW* schema and labeling scheme used in Figure 1 is detailed in [21]. Collaboration activities rely on the secure edition of the *EAW* document as described in the following.

[1] This work is partly supported by the EU IST-2004-026650 project ”R4eGov”.

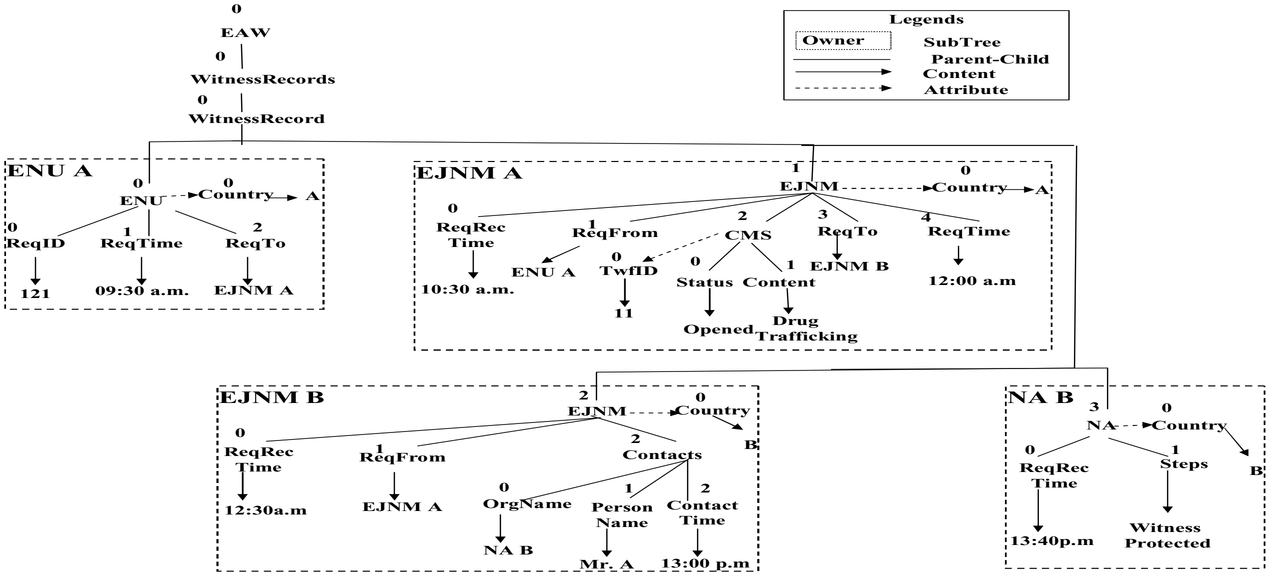


Figure 1. Four document part instances: *ENU A*, *EJNM A*, *EJNM B* and *NA B* of EAW document.

## 2.1. Scope

1. *Distributed Document Sources.* Parts of a document are issued asynchronously by participants that have to share a common document schema. Still, access to the different fields of the *EAW* document should abide by the local regulations and policies, which might vary depending on the European Union country or administrive body. For example, one country's law may prohibit the disclosure of the religious beliefs of a suspect, while it may be perfectly legal elsewhere.
2. *Fine Grained Document Access. NAB* may need to access a deeply nested element `PersonName` containing a suspect's name that is owned by the manager of *EJNM B*. Access patterns may range from the whole document to an individual element. These also may specify document content to capture the context which can be dynamic as participants perform their edition.
3. *Document Distribution.* No central repository should be assumed to be available. Document editors are supposed to send new or updated documents to other participants, an *ENU A* employee sending his update to an *EJNM A* employee for instance, or to store them into a peer-to-peer network.
4. *Autonomous Document Access.* At any time, a participant may need to access document parts that are originated by an authority that may become unavailable, provided the document can be obtained from another participant for instance. The edition of the document should not require a strong synchronization among all participants, and reconciliation algorithms and policies might be necessary if concurrent updates can happen due to the distribution scheme.

## 2.2. Security Objectives

**Document Authorization:**

1. *Access Decision Delegation.* For scalability reasons, access control decisions cannot be performed in centralized fashion, especially in scenarios involving virtual organizations or multiple authorities. For example, one may delegate the access decision regarding `PersonName` to a subordinate during vacation.
2. *Distributed Control of Data Disclosure.* Only authorized participants should get access to the protected document parts, yet access control enforcement should not rely on a central authority.

**Document Protection:**

1. *Document Confidentiality*. Selected document contents should only be disclosed to the authorized participants.
2. *Document Authenticity*. Participants should verify that the received document is from an authentic source.
3. *Content-wise and Structural Integrity*. Participants should be able to verify the content-wise and structural integrity of a document as intruders may alter original documents or inject invalid ones.
4. *Traceability of Document Access*. All accesses by participants should be tracked. Our proposal traces update access only, in particular to ensure that the update of a document part was performed in conformance with the authorization policy.

# 3. Solution Overview

This paper describes a distributed XML document access control scheme for XML documents featuring a two phase controlled edition protocol. This section outlines the edition protocol and introduces the encryption scheme used for the client based enforcement of access control.

## 3.1. Edition Protocol

Every run of the document edition protocol starts with an interest specification phase, in which all the interested participants request access to document parts to their direct authority (the owner in the beginning). Participants express their interest using access primitives described at Section 4.1. Upon receiving those descriptions, the participant's authority evaluates requests with respect to its access control policy and determines which participants share the same access interests (see Section 4.2). The authority finally distributes control information (Section 5.2) used by the participants to enforce access control and to manage group membership.

The protocol then moves on to a collaboration phase, in which actual accesses are performed on the document. The various parts of the exchanged documents are encrypted according to the access control policy. Access to a document part is made possible only through the computation of a common secret key based on the control data received from their authority (Section 5.3). Participants performing updates also have to encrypt the document parts they modified.

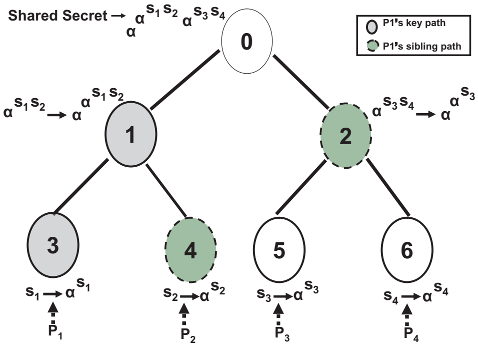

Shared Secret → $\alpha^{\alpha^{s_1 s_2}\alpha^{s_3 s_4}}$
P1's key path
P1's sibling path
0
$\alpha^{s_1 s_2} \rightarrow \alpha^{\alpha^{s_1 s_2}}$
1
2
$\alpha^{s_3 s_4} \rightarrow \alpha^{\alpha^{s_3 s_4}}$
3
4
5
6
$s_1 \rightarrow \alpha^{s_1}$
$s_2 \rightarrow \alpha^{s_2}$
$s_3 \rightarrow \alpha^{s_3}$
$s_4 \rightarrow \alpha^{s_4}$
$P_1$
$P_2$
$P_3$
$P_4$

Figure 2. Tree-based Group Diffie-Hellman (TGDH) key agreement for 4 participants $P_1, P_2, P_3, P_4$. The participants host their DH values in the leaves. The notation $k \rightarrow \alpha^k$ means that participants compute the DH private value and then compute the DH public value $BK = \alpha^k$ and broadcast it.

### 3.2. A Distributed Approach

Autonomous document distribution and access plead for the enforcement of authorizations and protection objectives through a client based approach using data encryption as opposed to server based access control enforcement. In addition, scalability makes it infeasible for a single authority to manage every participant. While access control models like RBAC make it possible to hide some complexity in the user and even resource dimensions, they do not provide enough control to address the fine-grained authorizations required at document level. The authorization scheme proposed in this paper deals with such aspects by having participants involved in the access control decision and by making them local and accountable authorities.

It ensures at least a partial availability of documents through the separation of document transmission, which can rely on caching, peer-to-peer exchanges, or even sneakernet transmission from access control enforcement. This can be made scalable through key management and an a posteriori compliance verification of updates with the policy. A centralized access control system constituting a single point failure is for instance more subject to denial of service with respect to document availability.

### 3.3. Dynamic Groups

Our approach relies on the encryption of document parts to protect their access. Decentralizing access control enforcement through a static encryption scheme is however not enough to address dynamic changes in group membership. We also assume that participants might not know each other, except through their hierarchy, subordinates, or past collaborators, and that some organizations specifically want to retain the management of their personnel and authorizations. These issues are addressed through the adoption of a rekeying mechanism by which adding a new member would only imply updating the keys of the groups sharing his access interests. Keying and rekeying are done with a tree based group Diffie-Hellman (TGDH) protocol [11] (see Figure 2) operated on a binary key tree [2].

This scheme was developed to allow a group of participants to compute a common secret based only on the partial knowledge of other members of the group and without relying on a central authority, which also provides backward and forward secrecy. The TGDH original paper assumes that, when a participant joins or leaves, all others rekey together either synchronously as in the original scheme or semi-synchronously using an interval based rekeying [14] or a non-blocking rekeying [15]. In contrast, our approach is asynchronous and rekeys (see Section 5.4) only when required upon receiving a document encrypted for the updated group.

### 3.4. Document-Based Updates

To address the abovementioned issues, the group updates required by participants joining and leaving are piggybacked with the document. These updates are then communicated when a document is exchanged among participants, thereby suppressing the need for broadcasting synchronizing all participants. The group update information does not contain a whole tree update and therefore requires a limited memory for secret key computation. Upon receipt of a document, participants can decide whether they need to rekey to access one particular part of the document, a process we call lazy rekeying.

## 4. Access Control Policy

This section describes how participants interested in accessing some document part can express access patterns that will be matched against the access control policy. Participants can describe targets in the document which we call Expressions of Access Interest (*EAI*s), as can be described with X-Path or X-Query[3]. A $Valid$ *EAI* refers to such a target whose presence in the document has been validated by the authority such as the owner of the document. This section describes how a policy rule can express the access to such a target using access primitives (*View, Append, Delete, Rename*) and how to determine participants with a similar access interest.

### 4.1. Access Expressions

We extend the access primitives described in [19] and adapt them to cryptographic enforcement. An access consists in an operation that a participant performs on one or several parts of a secured document, and for which he requests the distribution of appropriate keys. The document parts on which the operation will take place are defined by $Valid$ *EAI*s ('targetEAI' and/or 'sourceEAI') and a propagation value. The propagation is described by a non-negative integer value $n$ or the + symbol respectively indicating that the access interest is propagated towards the n-th descendant nodes (elements and its attributes) or the whole subtree, $n = 0$ meaning no propagation.

**Access Primitives.** An access primitive is a function taking the abovementioned parameters as inputs and returning a subset of a document part, as follows:

1. **View(targetEAI, [Propagation])**. The **View** primitive returns the nodes of the document part that matches the valid 'targetEAI'. For a propagation value of 0, only the matching node with the 'targetEAI' without the descendant nodes is returned.

2. **Append(targetEAI, newNode, [Propagation])**. The **Append** primitive creates a new node (i.e. element,attribute) with the name 'newNode' as a child node of each matching node of the valid 'targetEAI'. If the propagation value is 0 only the first matching node is considered.

[2] The key tree is a binary search tree. The list of nodes in the path from the leaf to the root are termed a key path. A sibling path is the list of sibling nodes of the nodes in a key path.

[3] We also defined a label based language [21] for expressing access interests based on the document semantics rather than its syntactic structure as X-Path or X-Query do; still our access control policy expression is quite independent from the target description language. Such labels are used in Figure 1.

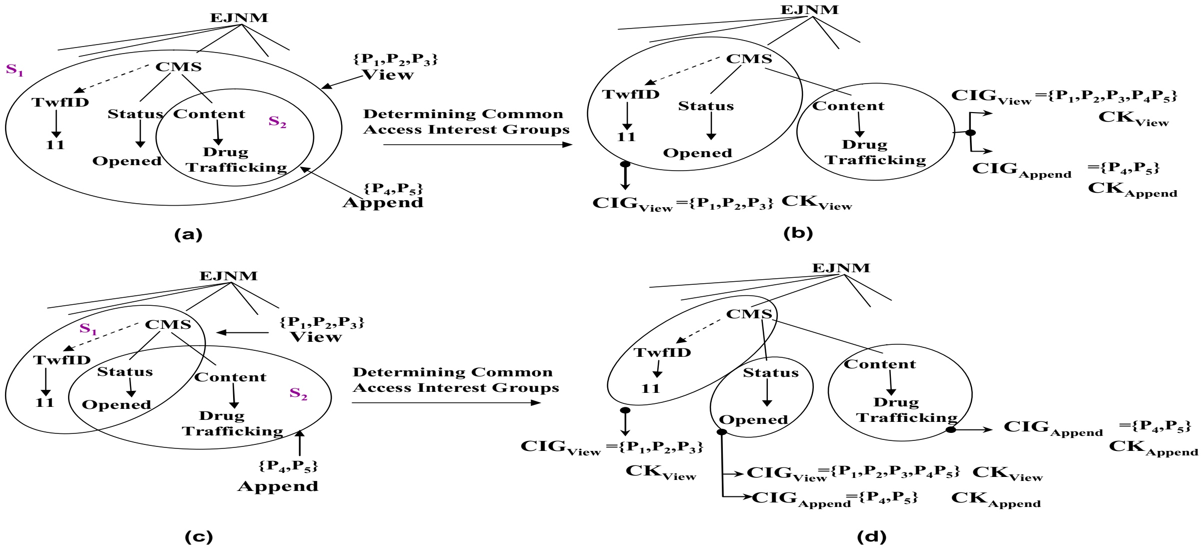


Figure 3. Determining common access interest groups (CIG). (a) $S_2$ is subsumed by $S_1$. (b) 3 groups are determined with two disjoint sets of nodes. (c) $S_2$ is partly subsumed by $S_1$. (d) 4 groups are determined with three disjoint sets of nodes.

3. **Delete(targetEAI, [Propagation])**. The **Delete** primitive deletes the nodes rooted at the matching valid 'targetEAI'. The deletion is performed either up to the n-th descendants of the matching node or the whole subtree from that node.

4. **Rename(targetEAI, newName[ ], [Propagation])**. The **Rename** primitive renames the nodes of the document parts matching the valid 'targetEAI'. It is propagated either down to the n-th descendant nodes of the matching node or down the whole subtree rooted at the matching 'targetEAI'. Each propagation of the access primitive renames the corresponding descendant node with a new name from the list 'newName[ ]'.

We term Append, Delete, and Rename as update primitives hereafter. Update primitives implicitly grant a permission for the View primitive to the nodes they apply to. Other operations like Copy(sourceEAI, targetEAI, [Propagation]) and Move(sourceEAI, targetEAI, [Propagation]) can be built using these primitives. Copy creates an exact subtree up to n-th descendants of the document parts rooted at the node matching the valid 'sourceEAI'. The created subtree is then appended as a child of the nodes matching the valid 'targetEAI'. Intuitively, it uses the Append primitive. Move does exactly the same operation as Copy except that it additionally deletes the subtree matched by the 'sourceEAI'.

**Credentials.** We assume that the access control policy is described based on credentials associated with the participants. Such credentials, which are outside the scope of this paper, should make it easy to determine whether a participant can read and subsequently control the dissemination of some part of a document. In applications involving the collaboration of several organizations, credentials would likely describe organization or group membership, roles, clearance level, or possibly trust. Our only assumption is that credentials combine such information together with the participant's public key, either through the signature of a certificate by an appropriate authority or through secured exchanges between trusted modules. Access control rules are formed through the association of credentials with access primitives. Access interest requests sent by participants to obtain the common secret key corresponding to a given primitive associate the participant's access key related to this primitive together with the access primitive description. Such requests are signed with the participant's key for authentication purposes.

## 4.2. Common Access Interest

The authority determines a disjoint set of all common access interest groups ($CIGs$) with respect to the 'targetEAI's of all the access interest requests received at the interest specification phase. We assume thereafter that the authority is a member of every group it is managing. Let us assume two access primitives $ap1$ and $ap2$ from $P_1$ and $P_2$ containing $targetEAIs$ $e_1$, $e_2$ respectively refer to the two subtrees $S_1$ and $S_2$ of the document part $d_i$ and $O_i$ is $d_i$'s authority.

**Disjoint Sets.** If $S_1$ and $S_2$ are disjoint, meaning $S_1 \cup S_2 = null$, then $e_1$ and $e_2$ do not overlap. $P_1$ and $P_2$ are assigned to two disjoint sets of common access interest groups $CIG_{ap1} = \{P_1\}$ and $CIG_{ap2} = \{P_2\}$ respectively.

**Non-Disjoint Sets.** If any subtree $S_2$ is either (1) entirely subsumed by the other $S_1$, or (2) partly subsumed by the other $S_1$, then some overlapping occurs between $e_1$ and $e_2$. Determining the disjoint set of common access interest groups in this case proceeds as follows. Regarding case (1), two disjoint subtrees of nodes are determined: one with the subsumed subtree $S_2$ and the other with $S_1 \backslash S_2$. Regarding case (2), three disjoint subtrees of nodes are determined: one with $S_1 \backslash S_2$, the second with $S_1 \cap S_2$ and the last with $S_2 \backslash S_1$. Each disjoint subtree is associated with an access primitive accordingly.

**Update vs. View.** Update primitives ($U = ap \in \{A, D, R\}$) generate two different groups $CIG_{U_i}$ and $CIG_{V_i}$ while View primitives ($V = View$) require only one group $CIG_{V_i}$ to be formed. Note that any participant having an Update access interest may need to be a member of multiple groups because of the implicit granting of a view access and of the overlapping of different access interests:

- In case (1), assuming that $e_1$ and $e_2$ respectively refer to view and update primitives, the disjoint groups formed are: $CIG_{V_{1\backslash 2}} = \{P_1\}$, $CIG_{V_2} = \{P_2\}$ and $CIG_{U_2} = \{P_2\}$. If on the contrary $e_1$ and $e_2$ respectively refer to update and view primitives, the disjoint groups formed are now: $CIG_{V_{1\setminus 2}} = \{P_1\}$, $CIG_{U_{1\setminus 2}} = \{P_1\}$, $CIG_{V_2} = \{P_1, P_2\}$ and $CIG_{U_2} = \{P_1\}$.

- In case (2), assuming that $e_1$ and $e_2$ respectively refer to view and update primitives, the disjoint groups formed are: $CIG_{V_{1\setminus 2}} = \{P_1\}$, $CIG_{V_{1\cap 2}} = \{P_1, P_2\}$, $CIG_{U_{1\cap 2}} = \{P_2\}$, $CIG_{V_{2\setminus 1}} = \{P_2\}$ and $CIG_{U_{2\setminus 1}} = \{P_2\}$ . In contrast, if $e_1$ and $e_2$ respectively refer to update and view primitives, the disjoint groups formed are: $CIG_{V_{1\setminus 2}} = \{P_1\}$, $CIG_{U_{1\setminus 2}} = \{P_1\}$, $CIG_{V_{1\cap 2}} = \{P_1, P_2\}$, $CIG_{U_{1\cap 2}} = \{P_1\}$ and $CIG_{V_{2\setminus 1}} = \{P_2\}$ .

Figure 3 depicts case (1) and (2) considering view and append primitives for $S_1$ and $S_2$ respectively from $P_1, P_2, P_3$ and $P_4, P_5$. A similar figure for the append and view primitives for $S_1$ and $S_2$ can be found in [21].

## 5. Key Management

A participant may have different access interest (View, Append, Delete, Rename) depending on its collaboration needs which they convey to the authorities by sending access primitives. An authority needs to identify the participants based on their signature in the access primitives. In effect the participants may not know other participants who have the same access interest yet they want to compute a common secret key keeping their privacy. In the case of unavailability of the authority, participants should be able to act as delegate. As a result participants need means using which they will be able to not only compute common secret keys of the groups they are in but also to be a delegate dynamically.

This section introduces three different keys and their management to achieve those. First, **participant key** pair $(PK_i, SK_i)$ associated with each participant $P_i$ to relate them with credentials as described in Section 4.1 and is not discussed further. Second, **access keys** associated with each access primitive identifies particular access interest of a participant. Finally, a **common secret key** defines each common access interest group.

**Notation.** In the sequel of the paper, $d$ refers to a document made of $n$ document parts $d_{i\in[1,n]}$. $\hat{d_i}$ refers to the root node of any document part $d_i$. $\mathcal{M}$ and $\mathcal{K}$ denote the message and key space respectively. $h_1$ and $h_M$ denote a one way hash function and a Merkle hash function [1] respectively. The encryption and signature of a message $m \in \mathcal{M}$ with key $K \in \mathcal{K}$ is written as $[m]_K$ and $Sign_a(m) = [h_1(m)]_{K_a}$ (where $K_a$ is the private key of $a$) respectively.

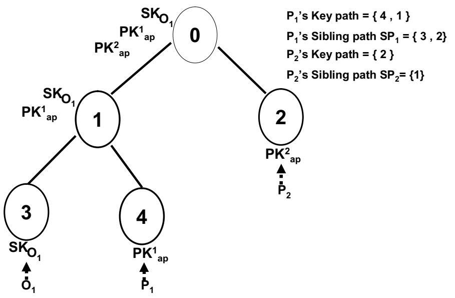
### 5.1. Access Key Generation

Each participant $P_i$ possesses a set of **access key** pairs ($SK^i_{ap}$, $PK^i_{ap}$) associated with a particular access operation $ap$. These keys also serve to compute the common secret key used for document encryption and decryption. Based on a new unique private access key $SK^i_{ap}$, the participant generates his corresponding public access key using the Diffie-Hellman scheme [9].

$$PK^i_{ap} = \alpha^{SK^i_{ap}} \quad mod \quad p$$

Participants $P_i$ send a message consisting of the access primitive and the corresponding public access key $PK^i_{ap}$ signed using the participant's private key $SK_i$ to Authority $A_i$.

$$P_{i\in[1,n]} \xrightarrow{Sign_{P_i}(AccessPrimitive(), PK^i_{ap})} A_i$$



P$_1$'s Key path = { 4 , 1 }
P$_1$'s Sibling path SP$_1$ = { 3 , 2}
P$_2$'s Key path = { 2 }
P$_2$'s Sibling path SP$_2$= {1}

Figure 4. Owner $O_i$'s key tree with two participants $P_1$ and $P_2$. $O_i$ computes the sibling paths for $P_1$ and $P_2$.

### 5.2. Control Data Block Distribution

After determining the common access interest groups the authority takes the charge of building a control data block $CD_{A_i}$ containing information for common secret key computation for each member of a group it manages. This block consists of a set of individual blocks $CD^{z\in[1,m]}_{A_i}$ for $m$ members of group $CIG^{d_i}_{ap}$ interested in document part $d_i$ for access $ap$ and defined as follows:

$$CD^{z\in[1,m]}_{A_i} = [SP_z]_{PK_z}$$

$SP_z$ is the sibling path containing a list of public DH values that participant $P_z$ uses to compute its key-path. The number of such values being variable with the participant but always smaller or equal to number $m$ of participants in that group and larger than or equal to $log(m)$ in case of a balanced tree. Note that, these values are computed DH public values of the group members rather than being the public participant keys of the members. This prohibits one participant to identify other members in the group and thus they remain anonymous to him.

The authority finally encrypts and sends each individual block $CD^z_{A_i}$ with the public key $PK_z$ of every participant $P_z$ as determined from the submitted credential and signed requested access pattern.

$$A_i \xrightarrow{CD^z_{A_i}} P_{z\in CIG^{d_i}_{ap}}$$

Knowledge of the control data block enables each participant to compute the common secret key of its respective groups and act as a delegate afterwards. Such a message cannot be intercepted since each individual block is encrypted with authorized member's participant public key.

### 5.3. Common Secret Key Management

In TGDH, every node in the key tree (Figure 2) is assigned a unique number $v$, starting with the root node that is assigned 0: the two child nodes of a non-leaf node $v$ are set to $2v+1$, and $2v+2$ respectively. Each node $v$ is associated with a key pair consisting of a DH private value $K_v$ and of a DH public value $BK_v$, relying on the hardness of solving the discrete logarithm. For every node $v$, $K_v$ is computed recursively as follows:

$$K_v = \begin{cases} (BK_{2v+1})^{K_{2v+2}}\ mod\ p; \text{ if } v \text{ is a non-leaf node;} \\ = (BK_{2v+2})^{K_{2v+1}} \quad mod \quad p \\ = \alpha^{K_{2v+1}K_{2v+2}} \quad mod \quad p \\ SK^i_{ap}; \text{ if } v \text{ is a leaf node.} \end{cases}$$

In short, computing the DH private value $K_v$ of a non-leaf node requires the knowledge of the DH private value of one of the two child nodes and the DH public value of the other child node. In effect, one participant only needs to compute the DH private values along its key-path. In other words, one participant only needs to know the DH public values of the

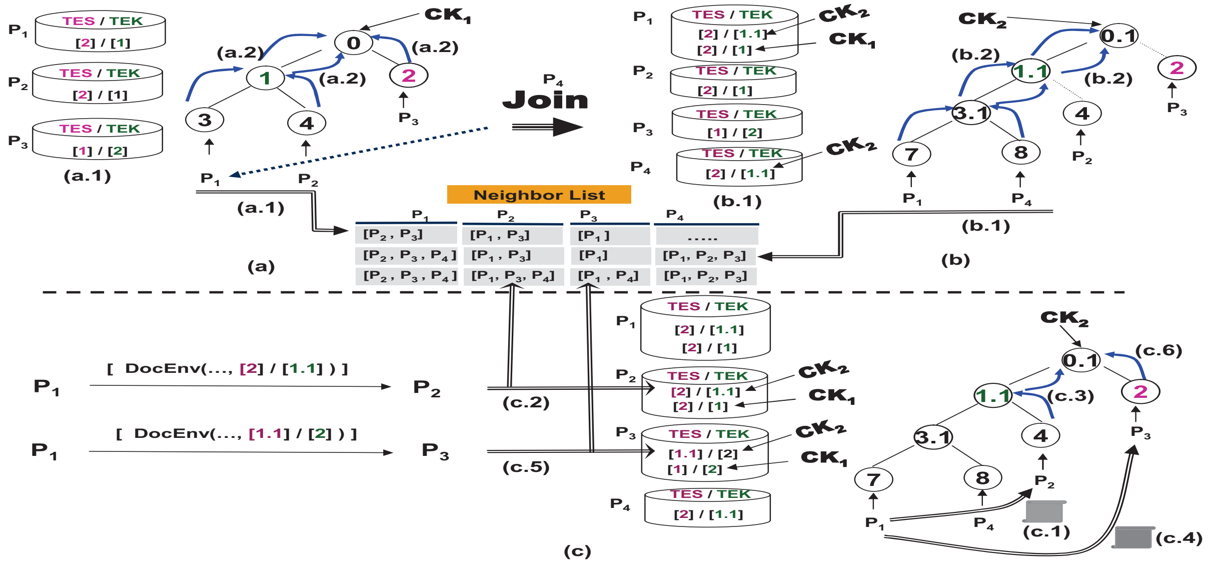


Figure 5. (a) $P_1$, $P_2$ and $P_3$'s key-paths with two data structure (i.e. TES/TEK, Neighbor List). (b) after $P_4$ joins to $P_1$. (c) Lazy rekeying of $P_2$ and $P_3$ after receiving document envelope from $P_1$. DocEnv(..) is the secure document envelope (Section 6.1).

siblings of the nodes of its key-path (sibling path). Therefore, the value $K_0$ computed for the root is the secret for all the participants (including owner). At this point, the common secret key is derived from the shared secret as follows: $CK_{ap}^{d_i} = h_1(K_0)$.

In a distributed environment like the EAW scenario (Section 2), one cannot assume the presence of any centralized entity for computing and distributing the common secret key. Even if such an entity were available, it would constitute a single point of failure thereby rendering the system vulnerable. In contrast, the owner (original authority) takes the charge of initializing the group collaboration by exploiting the key tree structure of TGDH. In particular, the owner generates a key tree by providing its DH private value in one leaf node and taking other participants' DH public values (i.e Public access keys ($PK_{ap}^{i}$)) one by one as other leaf nodes.

**Example.** Figure 4 illustrates how the owner $O_1$ builds the key tree for two participants with keys $PK_{ap}^{i \in [1,2]}$. Once the key-tree is generated, the owner determines the sibling path values $SP_{z \in [1,2]}$ and sends them as part of control data block required for $P_1$ and $P_2$. □

While the owner initiates group collaboration, the scheme is not centralized as multiple owners can intervene on their respective documents, and as the participants compute the common secret key along their key-paths independently. Moreover, the computation of the shared secret is contributory [11] by nature as the owner takes public access keys of all participants as the leaf nodes of the logical key tree to compute the common secret key and therefore $SP_z$. Furthermore, this scheme has a twofold advantage. First, participants can compute the common secret key without generating the complete key tree nor identifying other participants in the group, which is essential with respect to document centric exchanges. Second, group membership scales as described in [14, 11].

**Definition 1** *Protected Document Part $d_i^e$: Given a common interest group $CIG_{ap}^{d_i}$ with a 'targetEAI' $e$ over a document part $d_i$, any participant in the group can build a protected document part $d_i^e$ by encrypting all nodes $N \in e$ with the common secret key $CK_{ap}^{d_i}$ while other nodes of $d_i$ (i.e. $d_i \backslash N$) are left unchanged.* ■

According to the definition any participant having the common secret key $CK_{ap}^{d_i}$ is able to get $ap \in \{V, A, D, R\}$ access to $N \in e$ of $d_i^e$. The document owner is assumed to originally distribute a protected document in which all subtrees are protected with appropriate common secret keys as determined by the set of access interests it received.

## 5.4. Lazy Rekeying

Managing dynamically joining and leaving participants requires updating the common secret key. Lazy rekeying refers to re computation of a new common secret key by a participant only when it requires to do so (see Figure 5). This will take place when interacting with a participant that knows about a different version of the group, which may happen upon receiving a document envelope (further described in Section 6.1).

**Definition 2** *Neighbors: A neighbor of participant $P_i$ is member of a TGDH group who provides DH public values contributing to the computation of the DH private values along the key path of $P_i$.* ■

It can be observed from a participant's point of view that any dynamic change in its neighbors incurs an update in its key-path and similarly any dynamic change in its non-neighbors incurs an update in its sibling path. In particular, incurred dynamic changes cause new DH values to be computed in corresponding key paths and sibling paths.

**Definition 3** *Top End Key-path Value (TEK) and Top End Sibling-path Value (TES): A participant $P_i$'s TEK is the computed DH private value associated with the top most node along its key path and TES is the received DH public value associated with the top most node along its sibling path.* ■

**Example.** In Figure 2 $P_1$ and $P_2$ are neighbors to each other and so are $P_3$ and $P_4$. The DH values of nodes 1 and 2 are the TEK and TES for $P_1, P_2$ respectively and the DH values of nodes 2 and 1 are the TEK and TES of $P_3, P_4$ respectively. In other words, neighbors have exactly the same TEK and TES for a common access interest group. □

Lazy rekeying relies on the usage of Neighbor List and the pair TES/TEK maintained by each participant in a group where TES/TEK values are piggybacked with the secure document envelope. The usage of Neighbor List and TES/TEK is as follows: $P_i$ updates its neighbor list and TEK only when acting as a delegate for a joining/leaving event or receiving a secure document envelope containing a new TEK value indicating there has been a change in its neighbor list. $P_i$ updates its TES only when it receives a document envelope containing a new TES value meaning there is a dynamic change in the key-paths associated with it's TES. The TES/TEK being piggybacked merely adds a small amount of information to the envelope, which makes it scalable.

As group membership changes (further described in Section 5.5), initial re computation is performed only for the key-paths associated with the current authority and the participant that is subject to join or leave. The pair TES/TEK also contains the subject participant's key (not shown in the Figure 5) so that recipient can update its neighbor list accordingly. At this point, both can either exchange previous document updates to the existing group members or perform new updates in documents and then send document updates to the current group members including or excluding the subject participant. In the former case, the secure document envelope contains the previous TES/TEK whereas in the latter case, a new TES/TEK is introduced. Dynamic changes in the group are not broadcasted and the members that did not interact with newcomers still can collaborate using previous common secret keys and they will not even notice the join/leave event.

**Example.** In Figure 5, $P_1$, $P_2$ and $P_3$, originally under $O_i$'s authority (not in the figure), have an initial Neighbor List of respectively $[P_2, P_3]$, $[P_1, P_3]$ and $[P_1]$ (a.1). $P_1$ and $P_2$'s TES/TEK as [2]/[1][4] and $P_3$'s TES/TEK as [1]/[2] (a.1). All of them have computed the common secret key $CK_1$ (a.2). When $P_4$ joins with the delegate authority $P_1$: $P_1 and P_4$ can compute their new key-paths and update their Neighbor List to $[P_2, P_3, P_4]$ and $[P_1, P_2, P_3]$ (b.1). $P_1$ and $P_4$ update their TES/TEK with the new value of [2]/[1.1] (b.1). At this point $P_1$ and $P_4$ can compute the new common secret key $CK_2$ (b.2). However, $P_2$ and $P_3$ are unaware about this joining event and thus do not know $P_4$'s identity.

If $P_1$ sends a secure document envelope with an updated TES/TEK [2]/[1.1] to $P_2$, the latter will notice TEK's update and thus the change in its neighborhood by comparing its TES/TEK with the received one (c.1). $P_2$ then updates its TES/TEK (c.2) and computes the new common secret key $CK_2$ in order to decrypt the document envelope (c.3). Similarly $P_3$ can update its neighbor and TES/TEK and compute the key $CK_2$ if $P_1$ sends a secure document envelope containing updated TES/TEK [1.1]/[2] to $P_3$ (c.4,c.5,c.6). Note that the sent TES/TEK is inversed for $P_3$ with compare to $P_2$ as $P_3$'s TEK is $P_2$'s TES in the key tree. □

### 5.5. Joining and Leaving

The authority delegates its access decision among the participants it is managing so that a group of participants can be updated dynamically even when the owner (initial authority) is unavailable. We now assume that the control data block $CD_{A_i}$ introduced in Section 5.2 contains additional information, in particular the description of access decision delegations and a description of the access control policy rules that apply to the document part whose access is granted to group members, as follows:

$$CD_{A_i}^{z\in[1,m]} = [SP_z, Cert_{A_1}..Cert_{A_i}, SecObj]_{PK_z}$$

$(Cert_{A_1}..Cert_{A_i})$ denotes a chain of certificates originated from the owner $O_i$ (i.e. $A_1$) to the participant $P_z$ for an access primitive $ap \in \{V, A, D, R\}$. Each certificate $Cert_{A_i} = Sign_{A_i}(PK_z, PK_{ap}^z)$ asserts that the authority $A_i$ authorizes $P_z$ to perform the access $ap$ over the document nodes of $d_i$ by binding $PK_z$ with $PK_{ap}^z$ in a signature. The first certificate in the chain being from the owner enables a participant to be a delegate which then also may add its certificate in the chain delegating further. This certificate then can be used as a proof to other participants of $CIG_{ap}^{d_i}$ that $P_z$ was entitled to access $d_i$. This can be also used to trace that $P_z$ has performed the updates on $d_i$.

'SecObj' defines security objectives, i.e., access control policy rules relevant for the $d_i$, like for instance the fact that the data referred to in the $d_i$ should be reserved to the German police. Based on the chain of certificates and the objectives, the participant, acting as a delegate for the owner, takes over the access decision related tasks of the owner in the interest specification phase, and can evaluate later access requests.

A new participant sends its access primitives to an authority $P_r$ it knows just as described in Section 5.1. $P_r$ being a delegate evaluates the new participant's request and determines its eligibility to becoming a new member of an existing group (or to create a new group). $P_r$ re-computes its key path taking the new member's access key into account and sends control data to the new members as described in Section 5.2.

The access control policy might additionally specify whether backward secrecy applies to the new participant, which should be described in 'SecObj'. If it does, the new member can start document exchanges using the new common secret key from that point on. Otherwise, the authority sends the previous *n* common secret keys to the new member $P_j$ so that it can observe the previous updates and collaborate on these if possible.

$$P_r \xrightarrow{[CK_1...CK_i..CK_n]_{PK_j}} P_j$$

In case of a voluntary leave, the participant sends its associated certificate $Cert_{A_i} = Sign_{A_i}(PK_z, PK_{ap}^z)$ in similar fashion to the direct authority $P_r$ it joined before.

$$P_{i\in[1,m]} \xrightarrow{[Cert_{A_i}]_{PK_j}} P_{j\neq i\in[1,m]}$$

$P_r$ deletes the leaving member node from its key path and recomputes its new key-path. More often, the participant's authority will decide on his group members' leave. If forward secrecy applies $P_r$ immediately sends a secure document envelope with new TES/TEK values to the available members of the groups wherein the leaving participant was a member of so that they can re compute the new common secret key. The other available members may not collaborate on document updates performed with the new key right after new members join and thus do not recompute the new common secret key. This means that available members recompute the corresponding common secret key in a lazy fashion only when they receive a secure document envelope from other participants in the group as described in Section 5.4 and depicted in Figure 5. In case the direct authority is unavailable, a participant may accordingly notify the next indirect authority that it knows from the certificate chain of the received control data block.

## 6. Document-Related Security Metadata

As mentioned earlier, the scheme described above makes no special assumption regarding how participants interact. In particular, we target scenarios in which only documents would be exchanged, possibly only on top of an asynchronous messaging scheme like email for instance. In that

[4] The labeled integer value of a key tree node represents the corresponding DH values.

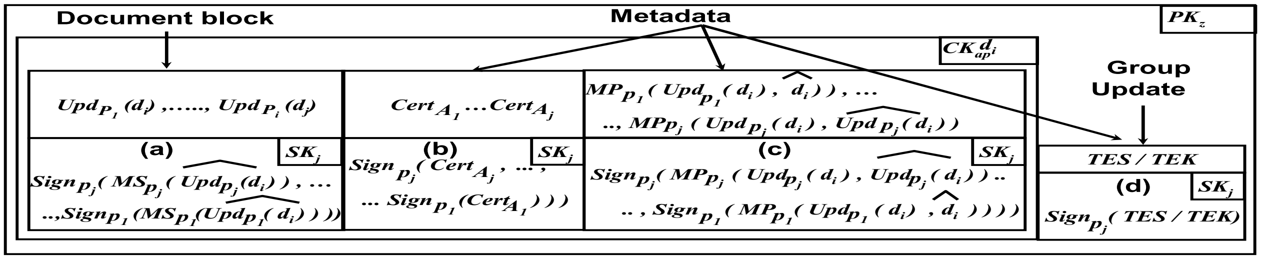

Document block
Metadata
PK_z
CK_ap^di
Group Update
Upd_P1 (d_i) ,....., Upd_Pi (d_j)
Cert_A1 ...Cert_Aj
MP_p1 ( Upd_p1 ( d_i ) , d_i ) ) , ...
.., MP_pj ( Upd_pj ( d_i ) , Upd_pj ( d_i ) )
(a)
SK_j
Sign_pj ( MS_pj ( Upd_pj (d_i) ) , ...
..,Sign_p1 (MS_p1 (Upd_p1 ( d_i) ) ))
(b)
SK_j
Sign_pj ( Cert_Aj , ... ,
... Sign_p1 (Cert_A1 ) ) )
(c)
SK_j
Sign_pj ( MP_pj ( Upd_pj ( d_i ) , Upd_pj ( d_i ) ) ..
.. , Sign_p1 ( MP_p1 ( Upd_p1 ( d_i ) , d_i ) ) ) )
TES / TEK
(d)
SK_j
Sign_pj ( TES / TEK)

**Figure 6.** Secure Document Envelope.

context, a document should contain all security metadata related to its content and data structure as well as to the correctness of its updates so far. It should also carry the necessary security metadata making it possible for the receiving participant to decide whether to rekey as explained above in Section 5.4 and which key to use to decrypt the various document parts. This section describes the secure document envelope data structure that carries such security metadata.

## 6.1. Secure Document Envelope

Document parts may be arbitrarily exchanged between and modified by an authority/editor of any authorized participant. This requires ensuring the authenticity and integrity of the data exchanged, even though the documents may be passed through third-parties like unauthorized participants or a node on the communication network. The Merkle Tree authentication mechanism [17] used for instance in [1, 8] to produce a Merkle signature out of a static XML document addresses such issues. A unique digital signature can be applied at the root node of the document to ensure both its authenticity and integrity as a whole. The collaborative edition process iteratively modifies document fields, therefore this technique alone is not enough. Definition 4 therefore introduces a document containment property similar to the one discussed in [1] that addresses such concerns:

**Definition 4** *Document Containment: Given a set of updated nodes $N \subseteq d_i$ of a document part $d_i$, a Merkle signature [1] of $MS_i(\hat{d_i})$ and the Merkle hash path*[5] *$MP(N, \hat{d_i})$: $N$ is said to be contained in $d_i$ if the locally computed Merkle hash value of $\hat{d_i}$ from the received $N$ and $MP(N, \hat{d_i})$ is equal to the verified signature value of $MS(\hat{d_i})$.* ■

A secure document envelope $SDE_j$ (Figure 6) consists of the document and of associated metadata. The metadata comprise a certificate chain, a Merkle hash paths blocks, and group updates that serve to notify about joining and leaving members.

**Document Block.** The document block, $(Upd_{P_1}(d_i), ..., Upd_{P_j}(d_i))$ is the series of updated document parts of $d_i$ performed by the participants $P_1, ..., P_j$ respectively. Each participant $P_j$ computes a Merkle signature over the root node of $Upd_{P_j}(d_i)$ which it signs together with the received Merkle signatures from the previous editors using its private key $SK_j$ (a).

**Metadata.** The metadata are generated as follows:

- *Certificate Chain.* Each certificate in the chain, $(Cert_{A_1}..Cert_{A_i})$ is formed as described in Section 5.5. Each participant $P_j$ signs its certificate $Cert_{A_j}$ received from its authority together with the received certificate chain with its private key $SK_j$ (b).
- *Merkle Hash Paths Blocks.* It consists of the list $(MP_{P_1}(Upd_{P_1}(d_i), \hat{d_i})$, ..., $MP_{P_j}(Upd_{P_j}(d_i), \widehat{Upd_{P_j}(d_i)}))$ of Merkle hash paths of the nodes of $d_i$ that are required for the recipient to compute locally the corresponding Merkle signatures with respect to the series of updates. Each participant $P_j$ signs its Merkle hash path $MP_{P_j}(Upd_{P_j}(d_i), \widehat{Upd_{P_j}(d_i)})$ together with the received Merkle hash paths starting from the owner (i.e $P_1$) with its private key $SK_j$ (c).
- *Group Update.* If a new participant edited the document or if the editor knows about a joining or leaving participant, he should compute the signature of the TES/TEK using his secret key $SK_i$, which should be added to the secure document envelope.

**Secure Envelope** The metadata are not all handled in the same fashion: the first two types of metadata are bundled with the document and hence encrypted together with the (potentially new) group common secret key $CK^{d_i}_{ap}$; group updates instead are only piggybacked with the document to perform lazy rekeying, and hence signed together with the previous encrypted block using the private key of their originator. Finally, all these blocks are encrypted together with the public key $PK_z$ of other interested participant $P_z$ which only $P_{z \neq j} \in CIG^{d_i}_{ap}$ can decrypt.

$$P_j \xrightarrow{SDE_j} P_{z \in CIG^{d_i}_{ap}}$$

## 6.2. Document Navigation

As described in Section 4.2, an accessible target document part can be divided into several disjoint fine grained target nodes, thus can be encrypted by a unique common secret key for a group. Similarly, a participant can be assigned to several groups and thereby needs to maintain several common secret keys. However, participants possess the knowledge of the document schema using which they annotate the document part schema nodes with the associated common secret keys that they compute and use those as encryption/decryption keys for corresponding document envelopes. As such participants can determine which key to use for which document part nodes before sending and receiving of secured document envelopes.

- Before sending an updated document envelope participants parse the schema to find the annotated common secret key associated with the updated document part.
- After receiving a document envelope participants can determine the required decryption key by observing the piggybacked TES/TEK value. If a re computation of a new common secret key is performed as a result of new TES/TEK, participants update their corresponding annotation in the schema with the associated new common secret key.

# 7. Security Evaluation

This section discusses the security of our dynamic access control model, with respect to both key management and protection achieved by the secure document envelope.

[5] a list of nodes' hash values required to compute the root's hash value.

### 7.1. Vulnerabilities in Key Management

**Join and Leave.** The possibility for an authority for evaluating huge number of join or leave requests may potentially expose the system to denial of service. This risk is first reduced by the distribution of authorities. If the threat model is such that traffic can be intercepted or the location of the authority of a critical group of participants is known to the attacker, the protocol should firsthand authenticate participants known by the authority. Asking the requestor to solve a cryptographic puzzle based on a secret shared with the authority should complement this measure to reduce the potential attack rate. Resorting to indirect authorities might be an attempt to bypass the direct authority. The policy they enforce should therefore always be more stringent than that of direct authorities and favor its immediate subordinate to mitigate further denials of service.

**Rekeying** Group updates correspond to retrieving a new TES/TEK during the collaboration phase. These are protected by the secure document envelope which a participant $P_z$ has to decrypt with $SK_z$. Similarly to join and leave, participants should authenticate the sender of a document update as a member of the group to prevent being tricked into decrypting a bogus document or computing a bogus key repeatedly. Annotating the document schema (Section 6.2) with the keys of groups and thus identifying the potential keys alleviate this problem partly. However, fake group updates may render into unnecessary rekeying. Additional constraints at the application level might help to detect fake group updates.

### 7.2. Document Protection

**Confidentiality.** Certificate chain and Merkle hash paths are encrypted by the common secret key and thus only be disclosed to a participant having the common secret key or capable of computing the key. The use of TGDH as the basic scheme for encrypting document parts ensures that a new participant will not get access to past exchanges (backward secrecy), except if handed the common secret key used to encrypt it. However, even though TGDH ensures forward secrecy, the lazy rekeying scheme we suggest makes it possible for a participant leaving or dismissed from one group to read parts of a document exchanged by existing participants that are unaware of the departure and that have not rekeyed. This is the result of the flexibility brought by the asynchronous mode of operation of our scheme. This might be alleviated by requesting a hard synchronization in the access control policy if possible; otherwise, participants that have been isolated from other members of a group should systematically rekey when sending updates to a document after some timeout.

**Resilience to Pruning and Grafting.** The nested signature over every block prevents any malicious participant in the group to include any fake data block (i.e. fake document parts, certificates and Merkle hash paths) or to suppress an existing block. In the extreme, an attacker can destroy the document if he controls the communication medium. This should be prevented using appropriate replication techniques.

**Containment.** Upon decrypting the encrypted data block any participant in the group can verify the received document part's containment as a whole thanks to the Merkle hash. For instance, when $P_j$ receives the initial secure document envelope containing $Upd_{A_1}(d_i)$ it can verify $Upd_{A_1}(d_i)$'s containment in the original document $d_i$ by computing a Merkle hash out of the received Merkle hash path $MP_{A_1}(Upd_{A_1}(d_i), \widehat{d_i})$ and locally computed hash values of $Upd_{A_1}(d_i)$. The computed Merkle hash should match with the verified signature value of $Sign_{A_1}(MS_{A_i}(\hat{d}_i))$.

**Integrity and authenticity.** Any participant $P_j$ upon receipt of a secure document envelope from $P_{j-1}$ can verify the document integrity and authenticity by computing the Merkle hashes out of the received Merkle hash paths $MP_{P_1}(..), .., MP_{P_{j-1}}(..)$ and locally computed hash values of corresponding document part updates $Upd_{P_1}(d_i), .., Upd_{P_{j-1}}(d_i)$. Each locally computed Merkle hash should match with the corresponding verified signature values of $Sign_{P_{j-1}}(MS_{P_{j-1}}(\widehat{Upd_{P_{j-1}}(d_i)})), ..., Sign_{P_1}(\widehat{Upd_{P_1}(d_i)}))$. Metadata integrity is also verified by checking the integrity of certificate chains and Merkle hash paths using $PK_i$.

**Traceability.** As each participant signs its document updates along with the previous series of updates performed by previous editors the recipient can trace everyone's updates by simply verifying the signatures iteratively. It can also verify the eligibility of previous editor $P_i$ to perform a given access through the iterative verification of the certificate chain, which should contain a certificate $Cert_{A_i}(PK_i, PK^i_{ap})$.

## 8. Related Work

There has been a quite remarkable progress in the area of fine grained access control on XML documents in [7]. It depicts a client-server centralized framework. Clients request the server for accessing a document. The server, which is responsible for designing the document schema, decides about the authorizations and at the same time enforces access control on the document. In [6] the authors describe a fine grained access control technique for SOAP based communication among web services.

From the enforcement perspective, these approaches are known as view based XML access control. However, the view based approach inherently contains two significant limitations [3]: scalability and storage. As an increasingly large number of requesters is involved, the management of views does not scale up and the increasing number of documents and clients demands more storage and cost on the server side. The view based approach also does not consider the issue of document updates where documents are dynamically exchanged among several participants and in particular document protection aspects. Moreover, the assumption here is that access control is specified and enforced by a centralized entity (e.g. DBA) of the XML data sources.

[16] presents mechanisms and algorithms for cooperative updates of XML documents in a distributed environment. While similar to our scheme in its use of cryptography to support controlled document edition, this work does not consider distributed sources of documents and their ownership. The approach is more tailored to a posteriori verification of the correct execution of a document edition process.

Encryption as an enforcement mechanism for access control decisions made at a server has been discussed in the literature for a while [2, 18, 12]: the server encrypts the data; the client can access these if it possesses the right decryption keys. This technique supports dynamic change only through the use of the server as a centralized point of enforcement that computes and distributes keys and therefore constitutes a single point of failure. In our case clients are able to perform there updates and send these autonomously. Moreover, key distribution is completely eliminated utilizing the TGDH. Scalability and performance are central issues growing with the number of clients accesses, notably regarding the need for partial reencryption of data because of changes in the access control rules. [12] improves scalability by introducing hierarchical publishers as so called routers in between producers and consumers of the documents. However, it relies on DOM to model the entire XML document in memory which is impractical for large enterprise documents for memory and performance reasons. Moreover, it is not able to disseminate dispersed document nodes in the same request from several XML subtrees. It should also be mentioned that these papers altogether do not address traceability issues with respect to document updates.

The use of tamper-resistant modules [4] however makes it possible to alleviate the limitation of the latest approach regarding policy dynamicity, both in terms of access control decision and enforcement. Even though this approach makes it possible to enforce flexible and context-aware policies suitable for ubiquitous computing, we believe delegation might be enough to adapt the access control policy to collaborative document edition. This approach in addition requires the difficult and expensive deployment of a trusted infrastructure.

Our distributed access control solution is fundamentally different compared to these approaches:

- It provides an access interest specification based on primitives which allows each participant to specify its fine grained access interests on the document parts owned by other participants.
- Instead of a purely server based approach, it makes it possible to follow indiscriminately a push or pull based approach since access control enforcement does not rely on a central authority.
- While controlled by the owner, the edition of documents can be initiated and run by all participants interacting autonomously.
- Lazy rekeying avoids the latency time for group collaboration during group membership changes.
- No assumption is made regarding participant execution environments.

## 9. Conclusion and Future Work

We proposed a solution for a distributed and fine grained access control framework for XML document centric collaboration. This solution perfectly fits document edition scenarios in which multiple organizations are involved and manage a part of a composite document at their own discretion, as illustrated by the EAW case of mutual legal assistance. This addresses both document authorization and document protection in situations in which authorization granting authorities may be off line or unable to cope with numerous users and may not be directly in touch with collaboration participants. Document authorization relies on a cryptographic enforcement: only participants that have computed proper encryption keys can view or update a document part. Document protection also encompasses verifying the conformance of the update of a document part with the authorization policy of its authority. To the best of our knowledge, this framework is the first to cope with distributed access control decision and its decoupled enforcement through the use of lazy rekeying. We are currently working on an implementation of this framework to validate its deployment, and on further issues like document versioning and consolidation.

## References

[1] E. Bertino, B. Carminati, and E. Ferrari. Merkle Tree Authentication in UDDI Registries. Idea Group Inc, International Journal of Web Services Research, 1(2):37-57, 2004.

[2] E. Bertino and E. Ferrari. Secure and Selective Dissemination of XML Documents. *ACM Trans. Inf. Syst. Secur.*, 5(3):290–331, 2002.

[3] W.-C. L. Bo Luo, Dongwon Lee and P. Liu. *A Flexible Framework for Architecting XML Access Control Enforcement Mechanisms*, volume Volume 3178/2004 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, December 2004.

[4] L. Bouganim, F. D. Ngoc, and P. Pucheral. Dynamic access-control policies on xml encrypted data. *ACM Trans. Inf. Syst. Secur.*, 10(4):1–37, 2008.

[5] A. D. A. Boujraf and M. Noble. Towards e-Administration in the Large (R4eGov). Deliverable WP3-D7, 2007. EU IP R4eGov.

[6] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. Fine Grained Access Control for Soap E-services. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 504–513, New York, NY, USA, 2001. ACM.

[7] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. A Fine-grained Access Control System for XML Documents. *ACM Trans. Inf. Syst. Secur.*, 5(2):169–202, 2002.

[8] P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, and S. G. Stubblebine. Flexible authentication of xml documents. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 136–145, New York, NY, USA, 2001. ACM.

[9] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[10] W. Fan, C.-Y. Chan, and M. Garofalakis. Secure XML Querying With Security Views. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 587–598, New York, NY, USA, 2004. ACM Press.

[11] Y. Kim, A. Perrig, and G. Tsudik. Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Groups. In *CCS '00: Proceedings of the 7th ACM conference on Computer and communications security*, pages 235–244, New York, NY, USA, 2000. ACM Press.

[12] A. Kundu and B. Elisa. Secure dissemination of xml content using structure-based routing. In *EDOC '06: Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference*, pages 153–164, Washington, DC, USA, 2006. IEEE Computer Society.

[13] G. Kuper, F. Massacci, and N. Rassadko. Generalized XML Security Views. In *SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies*, pages 77–84, New York, NY, USA, 2005. ACM Press.

[14] P. Lee, J. Lui, and D. Yau. Distributed Collaborative Key Agreement Protocols for Dynamic Peer Groups. In *Network Protocols, 2002. Proceedings. 10th IEEE International Conference.*, pages 322– 331, New York, NY, USA, 12-15 Nov. 2002. ACM Press.

[15] X. S. Li, Y. R. Yang, M. G. Gouda, and S. S. Lam. Batch Rekeying for Secure Group Communications. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 525–534, New York, NY, USA, 2001. ACM Press.

[16] G. Mella, E. Ferrari, E. Bertino, and Y. Koglin. Controlled and Cooperative Updates of XML Documents in Byzantine and Failure-Prone Distributed Systems. *ACM Trans. Inf. Syst. Secur.*, 9(4):421–460, 2006.

[17] R. Merkle. A certified digital signature. *In Advances in Cryptology - CRYPTO '89 Proceedings, Lecture Notes in Computer Science*, 435:218–238, 1989.

[18] G. Miklau and D. Suciu. Controlling Access to Published Data Using Cryptography. In *VLDB*, pages 898–909, 2003.

[19] S. Mohan, J. Klinginsmith, A. Sengupta, and Y. Wu. Acxess - access control for xml with enhanced security specifications. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, page 171, Washington, DC, USA, 2006. IEEE Computer Society.

[20] M. Murata, A. Tozawa, M. Kudo, and S. Hada. XML Access Control Using Static Analysis. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 73–84, New York, NY, USA, 2003. ACM Press.

[21] M. A. Rahaman, Y. Roudier, and A. Schaad. A Distributed Access Control Framework For XML Document Centric Collaborations. Technical Report RR-08-219, Eurécom, 04 2008.