Policy Decomposition for Collaborative Access Control

Dan Lin† Prathima Rao† Elisa Bertino† Ninghui Li† Jorge Lobo‡

†Department of Computer Science
Purdue University, USA
{*lindan,prao,bertino,ninghui*}*@cs.purdue.edu*

‡IBM T.J. Watson Research Center
USA
jlobo@us.ibm.com

ABSTRACT

With the advances in web service techniques, new collaborative applications have emerged like supply chain arrangements and coalition in government agencies. In such applications, the collaborating parties are responsible for managing and protecting resources entrusted to them. Access control decisions thus become a collaborative activity in which a global policy must be enforced by a set of collaborating parties without compromising the autonomy or confidentiality requirements of these parties. Unfortunately, none of the conventional access control systems meets these new requirements. To support collaborative access control, in this paper, we propose a novel policy-based access control model. Our main idea is based on the notion of *policy decomposition* and we propose an extension to the reference architecture for XACML. We present algorithms for decomposing a global policy and efficiently evaluating requests.

Categories and Subject Descriptors

K.4.1 [**Computers and Society**]: Public Policy Issues; K.6.4 [**Management of Computing and Information Systems**]: System Management

General Terms

security

Keywords

policy decomposition, collaborative access control

1. INTRODUCTION

Today a second generation of web-based communities and hosted services are emerging which facilitate collaboration among users and organizations. The term Web 2.0 has been coined to embrace all those new collaborative applications and also to indicate a new "social" approach to share resouces, characterized by resource sharing and decentralization of authority. For example, commercial coalitions may implement supply chain arrangements, subcontracting relationships, or joint marketing campaigns [6]. In the public sector, governments have taken various initiatives to increase collaboration among government agencies and non-government organizations in order to provide better public service to citizens. Main goals in such collaborations are resource sharing and joint tasks and activities. As such a critical issue is represented by access control systems able to support selective access to sensitive resources.

One key requirement in access control approaches for collaborative applications is the notion of *collaborative access control* by which we mean that several parties participate to make access control decisions. There are several motivations for collaborative access control. A first motivation is privacy and confidentiality. Modern attribute-based access control models, such as XACML [3], require as input information about subjects requiring access to protected resources. Such information may be sensitive and owned by different parties, that may not be willing or able to share it. In some cases, there may not be a unique party having all the necessary information to take an access control decision. A second motivation is represented by organizational and business process requirements. Decisions concerning authorizations to execute tasks often require approval from multiple-independent parties, especially in the case of complex tasks. A third motivation is represented by the fact that parties may have applications already in place supporting access control functions and it may not be feasible requiring them to adopt other mechanisms or protocols for access control. Such requirement is crucial for dynamically formed coalitions with temporal constraints; in such case, the collaborative access control decisions have to be taken as soon as possible and thus the parties have to leverage on whichever access control system they have in place.

As an example focusing mainly on the first motivation, consider a large enterprise with several departments, including a classified project management department, L_1, and financial department, L_2. Each department is responsible for managing access to its resources and each department stores confidential data concerning its specific operations and mission. Consider a global policy stating that no one can acquire crypto equipments if he works for a classified project for the US government and he has not enough funds in his/her budget. Suppose that information about the managers of the classified projects is managed by L_1 and the information about the funds is managed by L_2 and that each department is not able, for confidentiality reasons, to release its own information to others. Suppose moreover that

the global policy requires that the requests be approved by the head of each department and the approval be recorded, through the execution of obligation actions. In such scenario a centralized enforcement of the policy is not feasible. Rather a more suitable approach would consist of "distributing" the policy to the two departments asking each of them to return its own "local" decision. The local decisions are then combined to generate the global decision.

Several access control models [6, 7, 12, 14, 16] have been proposed to address some of the unique access control requirements in collaborative applications. But none of those approaches address issues related to decomposing global policies into local policies and strategies for protecting sensitive information of each party involved in collaborative access control. Works in the area of privacy preserving access control [9, 11] have focussed on techniques that prevent request owner from knowing the exact policy being evaluated for his request and also prevent policy owners from knowing the sensitive credentials of the request owner. These works however, are not applicable in the context of a collaborative access control application where multiple parties have to not only co-operate in making an access control decision but also have to ensure that no sensitive information stored at a collaborating party is divulged in the process of making this decision. Recognizing the need for a policy-based access control model that can enforce a global policy among collaborating parties without compromising the autonomy or confidentiality requirements of the involved parties, we propose a novel policy-based access control model for collaborative environments based on the notion of *policy decomposition*. The main idea is that in a collaborative environment, a global access control policy is decomposed to local policies so that the policies local to a party only need the information available at that party and that the decisions obtained from the local policies can be combined to derive the access control decision for the collaboration as a whole.

We cast our solution in the context of the eXtensible Access Control Mark-up Language (XACML) [3] framework. XACML is a general purpose access control policy language which defines a request/response language and framework to enforce authorization decisions. We have chosen XACML because of its widespread adoption as a language of choice for enforcing access control in traditional and distributed environments [13]. In a typical XACML framework, there is a policy enforcement point (PEP) and a policy decision point (PDP). The PEP is responsible for issuing requests and enforcing the access control decisions. The PDP receives requests from the PEP and evaluates policies applicable to the requests and sends a decision back to the PEP. To support collaborative access control, we extend the XACML reference architecture by introducing multiple PDPs that communicate with a centralized PEP through a request dispatcher/decision coordinator. A global policy is thus decomposed into local policies for each PDP according to availability/sensitivity requirements of each party and user specified constraints. Given a request, the central PEP modifies the request and dispatches it to corresponding PDPs, and then combines the decisions.

Our contributions are summarized as follows.

- We propose a novel access control model in multi-party collaborative environments. An important feature of our model is that we can protect sensitive information of one party from being known by the other parties. It is the first time that privacy issues are considered in policy decomposition.
- We develop an extension to the reference architecture for XACML to support collaborative access control.
- We propose a method to optimize the evaluation of requests.
- We present complexity analysis of our proposed approach.

The rest of the paper is organized as follows. Section 2 briefly introduces the structure of XACML policies. Section 3 presents details of our proposed policy decomposition approach. Section 4 analyzes the complexity of our approach. Section 5 reviews related work and Section 6 concludes the paper and outlines future research directions.

2. XACML POLICIES

XACML [3] is the OASIS standard language for the specification of access control policies. It is an XML language able to express a large variety of policies, taking into account properties of subjects and protected objects as well as context information. In general, a subject can request an action to be executed on a resource and the policy decides whether to deny or allow the execution of that action. Several profiles, such as a role profile and a privacy profile, have been defined for XACML. Commercial implementations of XACML are also available [1, 2].

XACML policies include three main components: a Target, a Rule set and a Rule combining algorithm. The *Target* identifies the set of subjects, resources, actions and environents to which the policy is applicable. Each *Rule* in turn consists of another optional *Target*, a *Condition* and an *Effect* element. The *Condition* specifies restrictions on the attribute values in a request that must hold in order for the request to be permitted or denied as specified by the *Effect*. The *Effect* specifies whether the requested actions should be allowed (Permit) or denied (Deny). The *Rule combining algorithm* is used to resolve conflicts among applicable rules with different effects. An XACML policy may also contain one or more *Obligations*, which represent functions to be executed in conjunction with the enforcement of an authorization decision. An XACML request consists of a list of attributes characterizing a subject and its environment along with the attributes of the action and resource.

3. POLICY DECOMPOSITION

As aforementioned, policy decomposition is considered in a multi-party collaborative environment. Such a multi-party collaborative environment can either be a group of individual organizations or a large enterprise with several departments. In our paper, we assume all collaborative parties to be peers.

Our collaborative access control system is based on the architecture shown in Figure 1 which also highlights the relevant information flows. The basic idea is to decompose a global policy in such a way that each participating party does not need to have any sensitive information belonging to other parties to make an access control decision, and to combine decisions made by each participating party to obtain the decision for the global policy. In our system, there are a central policy enforcement point (PEP) and multiple

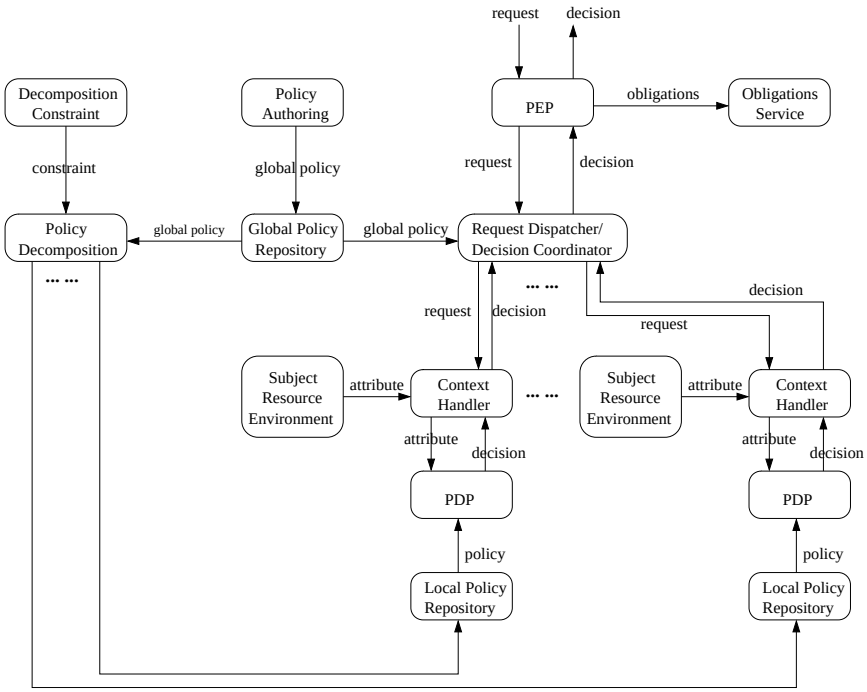

Figure 1: Data Flow Diagram

policy decision points (PDP). The central PEP and PDPs are connected by a *request dispatcher/ decision coordinator*(RDDC) and local *context handlers*.

The PEP, RDDC, policy decomposition module, global policy repository reside at one party called coordinator; each PDP and associated local policy repository reside at each collaborating party. The system implements two key functions: policy decomposition and request evaluation.

The policy decomposition function takes a global policy and policy decomposition constraints as input. The global policy is decomposed into local policies according to the constraints and then sent to the local policy repositories of corresponding PDPs. This function is performed by the trusted coordinator. After the decomposition, the global policy is encrypted and stored in a secure store. That means that the global policy will no longer be used for the subsequent request evaluation. Instead, only the non sensitive information of each global policy is kept as plain text in a *policy table* maintained by the coordinator. In particular, every row contains: policy targets, a list of participating local PDPs, and corresponding effect combining functions (details can be found in Section 3.2). Since the policy targets are kept public in the record we must ensure that no sensitive attributes are present in the policy targets. Any sensitive attributes that appear in a policy target are removed from the policy target and appended to the rule targets. The rules of the global policy which contain conditions that must be satisfied for a request to be permitted or denied is often more sensitive because it can involve the comparison of highly classified values. These rules are decomposed into local policies and sent to corresponding PDPs. In short, the coordinator is responsible for coordination and does not maintain any sensitive information; sensitive information is stored at each local PDP.

When a request is issued, the coordinator will first check whether information in the request matches the policy targets. If there is a match, the coordinator will check the policy table and distribute the request to the corresponding local PDPs. A local policy may contain predicates on both sensitive and non sensitive attributes. For non sensitive attributes, the local PDP looks for the attribute values in the request. For sensitive attributes, the local PDP accesses its local database and resolves the attribute values regardless of whether the request includes such values or not. In some situations, additional information like the requester ID may be needed to help local PDPs to resolve values of the sensitive attributes. The responses of local PDPs are collected and returned to the coordinator where the final decision is made.

In the rest of this section, we first discuss a running example in order to present an overview of the entire mechanism. Then we present a security analysis of our approach. After that we introduce a basic algorithm for policy decomposition and discuss how to integrate user defined decomposition constraints into the basic algorithm. Finally, we present an optimized approach for request evaluation in our system.

3.1 An Illustrative Example

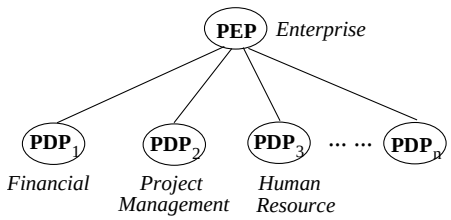

Figure 2: Hierarchy

PolicyId=P, RuleCombiningAlgorithm = deny-override
⟨RuleId=r1 Effect=Permit⟩
⟨Target⟩
⟨Subject ProjectName = "SecretCrypto" ⟩
⟨Action Action= "Buy"⟩
⟨/Target⟩
⟨Condition ProjectRole = "PI" and
ProjectLevel = "High"⟩
⟨/Rule⟩
⟨RuleId=r2 Effect=Deny⟩
⟨Target⟩
⟨Subject ProjectName = "SecretCrypto"⟩
⟨Action Action= "Buy"⟩
⟨/Target⟩
⟨Condition ProjectLevel = "High" and
Funding < 100000 ⟩
⟨/Rule⟩

Figure 3: Global Policy P

To illustrate the discussion we consider an enterprise with several departments, like financial department, project management department and human resource department as shown in Figure 2. Suppose there is a global policy P written in a simplified format as shown in Figure 3. The policy P contains two rules: $P.r_1$ and $P.r_2$. The rule $P.r_1$ states that a principal investigator (PI) of project called "SecretCrypto" with level "High" can buy advanced equipments. While the rule $P.r_2$ denies the right to buy advanced equipments to any subject with funding less than 100,000 dollars in the project called "SecretCrypto" with level "High".

We further assume that "ProjectName" and "Action" are public information and known by any department, while information about "ProjectRole" and "ProjectLevel" is only stored in the project management department and information about "Funding" is only stored in the financial department.

According to the available information at each department, we can decompose the policy P into P_1, P_2 and P_3 as follows, where policy P_1 and P_2 only contains project information and policy P_3 only contains financial information.

P_1 (Project Management Department): A principal investigator (PI) of project called "SecretCrypto" with level "High" can buy advanced equipments.

P_2 (Project Management Department): No one can buy advanced equipments for the project called "SecretCrypto" with level "High".

P_3 (Financial Department): No one can buy advanced equipments for the project called "SecretCrypto" with funding less than 100,000 dollars.

After the decomposition, the coordinator stores the policy targets in P together with a record stating that the policy P needs to be evaluated by policy P_1 and P_2 at project management department(PDP_1) and policy P_3 at financial department(PDP_2). Note that the coordinator only records the IDs of local policies but not the content of the local policies. The coordinator also stores functions stating that $P.r_1$ has the same decision as P_1; $P.r_2$ can be evaluated to true if both P_2 and P_3 are evaluated true. A record stored by the coordinator for this example is shown in Table 1.

Next, we show an example of a request evaluation based on decomposed local policies. Suppose Bob working on the project "SecretCrypto" wants to buy some equipment. A corresponding request ⟨ Bob, ProjectName="SecretCrypto", Action="Buy" ⟩ is received by the coordinator. The coordinator checks the policy table and finds that this request matches the policy targets of policy P. Then it sends the request to the project management department and the financial department. The project management department knows Bob is a PI and the corresponding ProjectLevel is "High", and hence a permit decision regarding P_1 and a deny decision regarding P_2 are returned to the coordinator. Similarly, P_3 is evaluated in the financial department and we assume a permit decision is returned to the coordinator. Finally, the coordinator combines the decision and permit Bob's request.

Global Policy ID	{PDP, Local Policy IDs}	Rule Effect Combination function
P	$\{PDP_1, P_1, P_2\}$ $\{PDP_2, P_3\}$	$P.r_1 : f_1(P_1)$ $P.r_2 : f_2(P_2, P_3)$

Table 1: Decomposition record for global policy P

In the above example, we observe that both P_1 and P_2 need to check whether the ProjectLevel is "High". Alternatively, we can modify policy P_1 as follows.

P_1' (Project Management Department): A principal investigator (PI) of project called "SecretCrypto" can buy advanced equipments.

Correspondingly, we need to revise the effect combination functions stored in the coordinator. Now $P.r_1$ is evaluated to true only when both P_1' and P_2 are true. This alternative solution is more efficient since it reduces redundant evaluation at local PDPs.

3.2 Security Analysis

In our approach, we assume that each participating party is not willing to share sensitive information; each participating party is trusted and local PDPs will correctly enforce the portions of the global policy that have been sent to them. Based on these assumptions, we proceed to analyze possible attacks to our system.

First, consider the situation when several parties other than the coordinator have been hacked. The information stored at these parties would be leaked while the information stored at other parties is still safe. Similarly, only the request evaluation involving the hacked parties may not be carried out properly.

Next, suppose the coordinator has been attacked. In this case, it is hard for the attacker to know the original global policies as they are encrypted. In order to guess policies that are stored at each local PDP, the attacker may send out fake requests as in the polling attack. However, since the attacker does not know based on what criteria (sensitive information) a request is evaluated, it needs to try a lot of combinations of attributes. Even it generates a set of requests like "Bob wants to buy equipments and he has 10,000 dollar funding", "Bob wants to buy equipments and he has 20,000 dollar funding", ..., "Bob wants to buy equipments and he has 90,000 dollar funding", in order to infer the funding requirements stated in the local policy, it still will not succeed. The reason is that if funding information is considered sensitive, the values regarding this funding information will be resolved at the local PDP, which means

Bob's funding amount in the request will be replaced by the real value at the local PDP and hence the above set of requests is essentially the same. At the end, the attacker will only know whether Bob can buy the equipments or not.

To sum up, our approach can prevent the attackers from knowing predicates on sensitive attributes in local policies as well as sensitive information stored at local PDPs.

3.3 Basic Decomposition

The decomposition consists of four main steps (outlined in Figure 4). First, we convert the global policy into compound

Algorithm 1. Policy_Decomposition(P)
Input : P is a global policy

```
//— step 1 —//
1. for each rule r_i in P
2.     B_{r_i} ← a compound Boolean expression of r_i
//— step 2 —//
3. identify unique atomic Boolean expressions in all B_{r_i}
4. label each unique atomic Boolean expression
//— step 3 —//
5. C_1,...,C_k ← Decomposition_Plan_Search
6. for j ← 1 to k
7.     convert every C_j into a local policy P_j
8.     distribute P_j to the destination PDP
//— step 4 —//
9. construct effect combination table for each rule at PEP
```

Figure 4: Policy Decomposition Algorithm

Boolean expressions over policy attributes [4]. A compound Boolean expression is composed of atomic Boolean expressions combined using the logical operations $\vee$ and $\wedge$. Atomic Boolean expressions that appear in most policies belong to one of the following two categories: (i) one-variable equality Boolean expressions, $a \triangleright c$, where a is an attribute name, c is a constant, and $\triangleright \in \{=, \neq\}$; (ii) one-variable inequality Boolean expressions, $c_1 \triangleleft a \triangleright c_2$, where a is an attribute name, c_1 and c_2 are constants, and $\triangleleft, \triangleright \in \{<, \leq, >, \geq\}$. Given a policy, we represent each rule together with the policy target as a compound Boolean expression. An example is given below.

EXAMPLE 1. *Consider policy P in Figure 3. The Boolean expression of rule $P.r_1$ is*
$\{(ProjectName = \text{“}SecretCrypto\text{”}) \wedge (Action = \text{“}Buy\text{”})\}_T$
$\wedge (ProjectRole = \text{“}PI\text{”}) \wedge (ProjectLevel = \text{“}High\text{”})$
The Boolean expression for rule $P.r_2$ is
$\{(ProjectName = \text{“}SecretCrypto\text{”}) \wedge (Action = \text{“}Buy\text{”})\}_T$
$\wedge (ProjectLevel = \text{“}High\text{”}) \wedge (Funding < 100000)$
The notation $\{..\}_T$ indicates that all atomic Boolean expressions appearing within it correspond to targets.

The second step is to decide what portion of the policy needs to be assigned to which PDP. We first identify unique atomic Boolean expressions in the policy. We cluster all syntactically equal atomic Boolean expressions into a group. Each such group corresponds to a unique atomic Boolean expression.

The values of certain attributes or a decision involving certain attributes may have to be provided by specific PDPs(and associated context handlers) in the collaboration. For example, the value of and the decision involving the *Funding* attribute must be provided by the financial department. Therefore we categorize and label the unique atomic Boolean expressions with the corresponding PDP's ID. Typically each atomic Boolean expression contains an attribute associated with a single PDP. However, for attributes in policy targets which are known by every party, we refer to them as *common attributes* and give them a special label for distinction, denoted as L_s.

EXAMPLE 2. *Consider policy P in Figure 3 again. Assume that the PDP's IDs with respect to the project management and financial departments are L_1 and L_2 respectively. Table 2 shows the unique atomic Boolean expressions in rule $P.r_1$ and the labels of each Boolean expression.*

ID	Unique atomic Boolean expression	Label
B_1	ProjectRole = “PI”	L_1
B_2	ProjectLevel = “High”	L_1
B_3	Funding < 100,000	L_2
B_4	ProjectName = “SecretCrypto”	L_s
B_5	Action = “Buy”	L_s

Table 2: Atomic Boolean Expressions and Labelling

The third step is to generate the actual local policies according to the labels attached to the atomic Boolean expressions. The basic idea is to replace a group of *correlated* atomic Boolean expressions with a local policy. We convert the compound Boolean expression of a policy into a disjunctive normal form (DNF) expression. The definition below introduces the notion of correlated atomic Boolean expressions.

DEFINITION 1. *Let F be a Boolean expression in DNF. Let B_i and B_j be two atomic Boolean expressions in F. We say that B_i and B_j are correlated iff they satisfy one of the following two conditions.*
(i) B_i and B_j have the same label and appear in the same disjunct of F.
(ii) B_i and B_j have the same label L and appear in two different disjuncts in which every atomic Boolean expression has the same label L.

For example, consider the DNF Boolean expression $F = (B_1^{L_1} \wedge B_2^{L_1}) \vee (B_2^{L_1} \wedge B_3^{L_2} \wedge B_4^{L_3}) \vee (B_5^{L_1})$. B_1 and B_2 are correlated according to condition (i) and B_5 is correlated with both B_1 and B_2 according to condition (ii). Based on the above definition, we define the notion of cluster of correlated Boolean expressions.

DEFINITION 2. *Let F be a Boolean expression in DNF. Let $B_1, \ldots, B_n$ be atomic Boolean expressions in F. We say that $B_1, \ldots, B_n$ form a cluster of correlated Boolean expressions iff B_i is correlated with B_{i+1} for every $1 \leq i \leq n-1$. The cluster is maximum if B_1 and B_n are not correlated with any other atomic Boolean expressions that are in F but not in this cluster.*

For each rule, it is easy to find all maximal clusters of correlated atomic Boolean expressions. Each cluster corresponds to a candidate local policy. Note that we do not create a separate local policy for a cluster with the special label. Instead, such a special cluster is appended to every cluster with label other than L_s to create local policies. The reason we call them candidate local policies is that maximal clusters are not always the best choice. We also need to consider the problem from the view of an entire global policy. This

is because some rules may share same Boolean expressions which are not the maximal cluster in all rules, but the creation of a local policy regarding this set of shared atomic Boolean expressions can improve the system performance. From now on, we will refer to a decomposition of a policy as a policy decomposition plan. An example is given below.

EXAMPLE 3. *Consider the rules $P.r_1$ and $P.r_2$ of the global policy P in Figure 3. Let B_1, B_2, B_3 and B_4 be atomic Boolean expressions, and let L_1 and L_2 be the labels referring to the project management and financial department PDPs. Using Example 1 and table 2 we derive :*
$P.r_1 : B_1^{L_1} \wedge B_2^{L_1} \wedge \{B_4^{L_s} \wedge B_5^{L_s}\}_T$
$P.r_2 : B_2^{L_1} \wedge B_3^{L_2} \wedge \{B_4^{L_s} \wedge B_5^{L_s}\}_T$
In $P.r_1$, $\{B_1^{L_1}, B_2^{L_1}\}$ is a maximal cluster of atomic Boolean expressions; in $P.r_2$, $\{B_2^{L_1}\}$,$\{B_3^{L_2}\}$ are maximal clusters respectively. The cluster $\{B_4^{L_s} \wedge B_5^{L_s}\}_T$ denotes the target. Two decomposition plans for policy P are shown in Table 3.

Plan A	Plan B
$P_1 : B_1^{L_1} \wedge B_2^{L_1} \wedge (B_4^{L_s} \wedge B_5^{L_s})$	$P_1' : B_1^{L_1} \wedge (B_4^{L_s} \wedge B_5^{L_s})$
$P_2 : B_2^{L_1} \wedge (B_4^{L_s} \wedge B_5^{L_s})$	$P_2' : B_2^{L_1} \wedge (B_4^{L_s} \wedge B_5^{L_s})$
$P_3 : B_3^{L_2} \wedge (B_4^{L_s} \wedge B_5^{L_s})$	$P_3' : B_3^{L_2} \wedge (B_4^{L_s} \wedge B_5^{L_s})$

Table 3: Decomposition Plans

Plan A creates local policies by always using maximal clusters. Compared to Plan A, Plan B has the same number of policies. However policies in Plan B have smaller sizes (we define policy size as the number of Boolean expressions in the policy). From the performance aspect, Plan B would be more efficient than Plan A since Plan B needs to evaluate fewer attributes.

Therefore, to guide the search of decomposition plans, we propose the following cost function which estimates the total workload for evaluating all local policies.

$$cost(P_1, ..., P_{N_P}) = \sum_{i=1}^{N_P} S_{P_i} + \alpha \cdot N_P \quad (1)$$

In equation 1, N_P is the total number of local policies, S_{P_i} is the size of policy P_i which is estimated by using the number of atomic Boolean expression in P_i. α is the overhead introduced at evaluation time due to the increased number of policies. Specifically, instead of evaluating a single large policy, evaluating a number of smaller policies may require a little bit more time. The smaller the value of *cost*, more efficient the policy decomposition plan will be. It is worth noting that we also need to check if the decomposed policy is consistent with the original rule. We present more details about determing a consistent decomposition plan in Section 3.3.1.

So far we have obtained a decomposition plan where each local policy is represented as a Boolean expression. We proceed to introduce the procedure for converting these Boolean expressions into actual XACML policies. As we know, a Boolean expression in a policy corresponds to one of policy components, like target and condition. Boolean expressions in a certain type of component of a global policy are placed in the same type of component in local policies. For example, if a Boolean expression belongs to a resource element in the target of the global policy, this Boolean expression will also be in the resource element in the target of the local policy. Second, in XACML, there are structures which can represent the meaning of $\vee$ and $\wedge$. Next, we set the effect of all local policies to *permit*. Then the constructed local policies are distributed to the PDPs indicated by the labels.

The last step is to compute the effects of original rules from the corresponding local policies. To achieve this, we propose an *effect combination table* maintained by the request dispatcher/decision coordinator. Each row in such table has the form of $\langle RID, F\rangle$, where RID is a rule ID and F is a Boolean expression on the decisions (denoted as $e(P)$) returned by local policies associated with rule RID. F is constructed based on the Boolean expression representing the rule. Specifically, given a request, if the decision of P is *permit*, the value of $e(P)$ will be *true*. Otherwise, the value of $e(P)$ will be $false$. If F is evaluated *true*, the decision of the corresponding rule will be determined by the rule effect. If F is evaluated *false*, the decision of the rule will be *NotApplicable*. Below is an example.

EXAMPLE 4. *Table 3 is an effect combination table for the policy decomposed using Plan B in Example 3.*

RID	F
$P.r_1$	$e(P_1') \wedge e(P_2')$
$P.r_2$	$e(P_2') \wedge e(P_3')$

Table 4: An Effect Combination Table

The effects of $P.r_1$ and $P.r_2$ are **permit** *and* **deny** *respectively. Consider a request q applicable to local policies P_2' and P_3'. Suppose that the decisions returned by P_2' and P_3' are both* **permit**. *Correspondingly, we have $e(P_1') = false$, $e(P_2') = true$, $e(P_3') = true$. The Boolean expression F of $P.r_2$ is then evaluated true, and hence the decision of the request q is* **deny**.

After the effects of the original rules are obtained, the rule combining algorithm is applied to generate the final decision.

Finally, we briefly discuss the situation when there is an update on the global policy. An update can be seen as a deletion followed by an insertion. If almost every rule has been modified, we remove all related information about the old global policy and decompose the new policy. If there is only minor modification to several rules in the global policy, we only need to re-decompose the rules sharing the same local policies with the updated rules.

3.3.1 Decomposition Plan

In the previous section, we have mentioned that the decomposition is based on finding the clusters of atomic Boolean expressions. An important requirement is to ensure that the decomposed policies are consistent with the original one. The following definition introduces our notion of consistency.

DEFINITION 3. *Let r be a rule in a policy, let P_1, P_2, ..., P_k be local policies with respect to r, and let F be the effect combination function of r. If for any request q, the decision yielded by F is the same as the decision yielded by r, we say the decomposition from r to P_1, P_2, ..., P_k is consistent.*

In what follows, we first introduce an algorithm for the search of a decomposition plan and then prove that the obtained decomposition plan is consistent.

The algorithm consists of two main steps. First, we reorder the atomic Boolean expressions in the DNF Boolean

expression of each rule. In particular, we rearrange the atomic Boolean expressions in the same clause by moving together the atomic Boolean expressions with the same label, i.e. atomic Boolean expressions belonging to the same PDP. It is now easy to find the maximal clusters of correlated Boolean expressions in each clause. Next, we move together the clauses with the same single label.

The maximal clusters of correlated Boolean expressions found in previous step are treated as candidate local policies. We can compute the evaluation cost of this set of candidate local policies according to Equation 1. The next step is to find out if other alternative local policies exist. As discussed in Example 3 when one local policy is included by another, we need to split the bigger candidate local policy to reduce the total evaluation cost. This only happens to policies that are local to the same PDP. Based on the observation, we propose the following heuristic to guide the decomposition plan search.

HEURISTIC 1. *Let P_1, P_2 be two policies local to the same PDP, and let f_1 and f_2 be the corresponding Boolean expressions of P_1 and P_2 respectively. If there exists a Boolean expression f_1' such that $f_1 = f_2 \wedge f_1'$ or $f_1 = f_2 \vee f_1'$, then we split P_1 into two local policies P_2 and P_1' where P_1' corresponds to f_1'.*

$P_1(f_2 \wedge f_1'), P_2(f_2) \Longleftrightarrow P_1'(f_1'), P_2(f_2)$
$P_1(f_2 \vee f_1'), P_2(f_2) \Longleftrightarrow P_1'(f_1'), P_2(f_2)$

In the above heuristics, it is obvious that evaluating P_1' and P_2 costs less than evaluating P_1 and P_2.

The situation becomes complicated when two local policies have an intersection rather than one being included by the other. In such case, we need to use the cost function to decide if we need to separate the intersection part from the candidate local policies. In particular, we have the following heuristic.

HEURISTIC 2. *Let P_1, P_2 be two policies at the same PDP, and let f_1 and f_2 be the corresponding Boolean expressions of P_1 and P_2 respectively. If the following two conditions are satisfied, we replace P_1 and P_2 with P_1', P_{12}, P_2' which correspond to Boolean expressions f_1', f_{12} and f_2' respectively.*

(i) $\exists f_1', f_{12}, f_2'$, $f_1 = f_1' \bigodot_1 f_{12}$ and $f_2 = f_{12} \bigodot_2 f_2'$, where $\bigodot_1, \bigodot_2 \in \{\wedge, \vee\}$.

(ii) $cost(P_1, P_2) > cost(P_1', P_{12}, P_2')$.

We now proceed to describe how to use these two heuristics. We insert policies local at the same PDP into a list. For each list $\{P_1, P_2, ..., P_k\}$, we start with the first two local policies. First, we apply Heuristic 1 to P_1 and P_2. Without loss of generality, suppose P_2 can be split into two local policies P_1 and P_2'. We replace P_2 with P_2' and the list becomes $\{P_1, P_2', ..., P_k\}$. We then consider policies P_1 and P_3. Suppose Heuristic 2 can be applied to them so that P_1 is split into P_1' and P_{13} while P_3 is split into P_3' and P_{13}. We remove P_1 and P_3 from the list and insert P_1', P_{13} and P_3' after P_2'. The list is then changed to $\{P_2', P_1', P_{13}, P_3', P_4, ..., P_k\}$. Since the first policy in the list, i.e. P_1, has been removed, we start a new round by selecting policies P_2' and P_1' and keep applying two heuristics. The same procedure continues until we reach the end of the list. The last step is to eliminate any duplicate Boolean expression created during the decomposition. The final result is a list of local policies represented as Boolean expressions. Figure 5 summarizes the algorithm for decomposition plan search.

Algorithm 2. Decomposition_Plan_Search(f_1, ..., f_n)
Input : f_1, ..., f_n are Boolean expressions of rule r_1,..., r_n

```
    //— step 1 —//
1.  for each rule r_i
2.      f_i' ← cluster atomic Boolean expressions
                with the same label in f_i
3.      append maximal clusters of atomic Boolean
        expressions to a list ℂ
    //— step 2 —//
4.  take every maximal cluster in ℂ as a local policy
5.  for each PDP
6.      𝕃 ← a list of local policies P_1, ..., P_m at this PDP
7.      P_i ← the first policy in 𝕃
8.      while P_i is not the last second policy in 𝕃
9.          P_j ← the policy following P_i
10.         while P_j is not the last policy in 𝕃
11.             apply Heuristic 1 to P_i and P_j
12.             if P_i can be split into P_i' and P_j
13.                 P_i ← P_i'
14.             else if P_j can be split into P_i and P_j'
15.                 P_j ← P_j'
16.             else
17.                 apply Heuristic 2 to P_i and P_j
18.                 if P_i and P_j can be split into P_i', P_ij and P_j'
19.                     insert P_i', P_ij and P_j' after P_j
20.                     delete P_i and P_j from 𝕃
21.                     P_i ← the first policy in 𝕃
22.                     P_j ← the policy following P_i
23.                 else P_j ← the policy following P_j
24.         P_i ← the policy following P_i
25. eliminate duplicate policies from 𝕃
26. return 𝕃
```

Figure 5: Decomposition Plan Search

3.3.2 Consistency

THEOREM 1. *Let r be a rule in a global policy P, and let P_1, P_2, ..., P_k be local policies generated by Algorithm 2 (Figure 5). P_1,P_2, ..., P_k are consistent with r.*

PROOF. To prove this theorem, we only need to prove that the combination of Boolean expressions regarding local policies P_1, ..., P_k is the same as the original Boolean expression F regarding r. We examine the decomposition plan search algorithm step by step.

First, we look at the reordering step. Suppose F' is the Boolean expression obtained after changing the order of the atomic Boolean expressions in the same clause in F. According to the commutative property, i.e. $a \bigodot b = b \bigodot a$ where $\bigodot \in \{\wedge, \vee\}$, and a, b are Boolean expressions, F' is equivalent to F. Suppose F'' is the Boolean expression obtained after changing the order of clauses in F'. Again, according to the commutative property, F" is equivalent to F', and hence equivalent to F.

Let f_1, ..., f_l be the maximal clusters of atomic Boolean expressions found in F''. According to the associative property, i.e., $(a \bigodot b) \bigodot c = a \bigodot (b \bigodot c)$ where $\bigodot \in \{\wedge, \vee\}$, and a, b, c are Boolean expressions, the combination of f_1, ..., f_l is equivalent to F''. By using the associative property, it is also easy to prove that the Boolean expressions obtained by applying any of the two heuristics is equivalent to the original Boolean expression.

To sum up, the local policies P_1, ..., P_k generated by the decomposition plan search algorithm are consistent with rule r. □

3.4 Constraint-based Decomposition

In some cases, policy owners have specific requirements when their policies are decomposed, which we refer to as *decomposition constraints*. For example, a party only trusts some specific PDPs for the evaluation of its policies and hence it requires that its policies can only be stored in these sites that it trusts.

To support such a function, we keep a profile for each PDP. The profile contains the PDP ID and a set of attribute and value pairs in the form of $\langle$ (PDP_ID, val_1), ($Attr_2$, val_2), $\cdots$, ($Attr_k$, val_k)$\rangle$. These attributes can be trust levels (TRUST), average response time taken to evaluate a request (AVG_RESP), etc. Decomposition constraints are then specified as Boolean expressions on some attributes listed in profiles, denoted as $f_c(Attr_1, ..., Attr_l)$.

Given a policy and a decomposition constraint, we carry out a filtering phase before the normal policy decomposition as described in previous section. In this filtering phase, we evaluate every PDP and leave out PDPs whose profile cannot satisfy the decomposition constraint. The decomposition is executed only among the qualified PDPs. Below is an example.

EXAMPLE 5. *Suppose there are three PDPs, whose ID are PDP_1, PDP_2 and PDP_3. Their profiles are as follows:*

- $\langle$*(PDP_ID, PDP_1), (TRUST, 3), (AVG_RESP, 0.1s)*$\rangle$.
- $\langle$*(PDP_ID, PDP_2), (TRUST, 1), (AVG_RESP, 0.03s)*$\rangle$.
- $\langle$*(PDP_ID, PDP_3), (TRUST, 5), (AVG_RESP, 0.03s)*$\rangle$.

A policy owner specifies his decomposition constraint f_c as: ($PDP\_ID = PDP_1$) $\vee$ ($TRUST > 2 \wedge AVG\_RESP < 0.05s$), which requires that the policy can only be handled by PDP_1 or PDPs which has a TRUST higher than 2 and AVG_RESP less than 0.05s.

After evaluating profiles of the three PDPs against f_c, we obtain two qualified PDPs, i.e. PDP_1 and PDP_3. The decomposition procedure will then only consider these two qualified PDPs.

It is worth noting that some decomposition constraints may render the policy decomposition unsatisfiable. For example, if a policy contains information only maintained at a PDP that does not satisfy the decomposition constraint, this policy cannot be decomposed. In that case, the user will be informed that his policy cannot be decomposed and be advised to revise his policy constraints.

3.5 Request Evaluation Optimization

A straightforward way to evaluate a request consists of three basic steps: (i) for each rule applicable to the request, evaluate its local policies; (ii) combine the effects of local policies based on the effect combination table; and (iii) apply the rule combining algorithm to obtain the final decision of the request. However, this naive method may not be always efficient. First, different rules may share the same local policies, and hence some policies may be repeatedly evaluated. For example, in Table 4, rules r_1 and r_2 share the local policy P_2. Second, the naive method evaluates all applicable rules which may not be necessary for some rule combining algorithms. Consider the permit-override combining algorithm as an example. If a rule with the permit effect is evaluated true, we do not need to check other rules, i.e., we do not need to check corresponding local policies. Based on these observations, we propose a novel method to generate an evaluation plan for a global policy by carefully considering the relationships among local policies, the properties of local PDPs, and the characteristics of rule combining algorithms.

Two main data structures are used in our method. IRE is an intermediate result table which stores the effects of local policies on a given request. RS is a response time table which keeps a record of evaluation time of each rule and each local policy.

We now proceed to introduce how to generate an evaluation plan for a global policy. The first step is to determine a proper order for evaluating rules. The rule evaluation plan consists of two levels: first-level plan and second-level plan. The first-level plan decides the evaluating order of the rules. For each rule, there is a second-level plan providing the evaluating order of local policies corresponding to this rule.

To obtain the first-level plan, we sort all the rules in an ascending order according to estimated evaluation cost. We use equation 1 to estimate the cost of evaluating a rule r by replacing N_P, S_{P_i} with N_P^r and $S_{P_i^r}$ respectively, where N_P^r is the total number of local policies regarding rule r, and $S_{P_i^r}$ is the corresponding policy size. If we do not have any knowledge about a rule-combining algorithm, like when a user defined algorithm is used, we use the current ascending order as the first-level plan. Otherwise, we can further customize the plan according to the types of rule-combining algorithms used by the global policy. In the following, we consider four common rule-combining algorithms.

Permit-overrides. The effect of the policy is "Permit" if a rule is encountered that evaluates to "Permit", regardless of the evaluation result of the other rules.

This rule-combining algorithm gives precedence to the rules with "Permit" effect. Thus, we reorder the rules which are already in an ascending order according to the evaluation cost, by first selecting rules with "Permit" effect and then selecting rules with "Deny" effect. Thus, once a rule returns a decision, the evaluation can stop and return the decision as the final decision for the request. Also, we can benefit from the ascending order as we always evaluate rules with lower cost first.

Deny-overrides. The effect of the policy is "Deny" if any rule is encountered that evaluates to "Deny". The effect of the policy is "Permit" if any policy evaluates to "Permit" and all other rules evaluate to "NotApplicable".

Deny-overrides is the opposite of permit-overrides. Thus, this time we move the rules with "Deny" effects before the rules with "Permit" effect. Similarly, once a rule returns a decision, the evaluation stops and this decision is treated as the final decision for the request.

First-one-applicable. The effect of the policy is the same as the result of the first applicable rule.

In this case, the first-level plan is skipped and the original order of rules is maintained.

Only-one-applicable. The effect of the policy corresponds to the result of the unique rule in the policy which applies to the request. Specifically, if no rule or more than one rules are applicable to the request, the result of rule combination should be "NotApplicable"; if only one rule is considered applicable, the result should be the result of evaluating the rule.

For this rule-combining algorithm, we do not need to make any change to the previously obtained first-level plan. We evaluate the rules in an ascending order of their estimated

evaluation cost. The evaluation will stop and return the decision "NotApplicable" if two applicable rules are encountered.

The second-level plan is developed based on the assumption that a rule is represented by a DNF Boolean expression of local policies. As we know DNF expressions have an important property. That is when one clause of the DNF expression is evaluated to true, the DNF expression is true. This means that the evaluation can stop when a satisfied clause is encountered. It would be beneficial to first evaluate the clauses with lower evaluation cost. Further, we notice that if a request does not match the rule target, the evaluation will also stop. Therefore, the second-level plan will first evaluate clauses with respect to the rule target in an ascending order of the evaluation cost and then evaluate the remaining clauses also in an ascending order of the evaluation cost. We employ the equation 1 again to estimate the cost of evaluating each clause. We replace N_P and S_{P_i} in equation 1 with N_{r_j,c_k} and $S_{P_i^{r_j,c_k}}$ respectively, where N_{r_j,c_k} is the number of local policies in the clause c_k of rule r_j, and $S_{P_i^{r_j,c_k}}$ is the corresponding policy size.

The last step is to actually evaluate local policies and rules according to the orders given by the rule evaluation plans. Each evaluation result returned by rules and local policies is stored in the intermediate result table IRE. For another rule containing the same local policies, we can directly use the results in IRE. Figure 6 summarizes the algorithm. If the global policy remains unchanged, we can adopt some optimizations to reduce the time of composing rule evaluation plans. First, the estimated evaluation cost of each rule needs to be computed only once and all rules are sorted once based on this cost. Second, the estimated evaluation cost of local policies is also computed only once and the second-level plans are the same for any request.

Finally a request issued to the system must be properly routed to the appropriate PDPs.

It is worth noting that there is an alternative method to estimate the evaluation cost of rules and local policies if we log the response times for the evaluation of each rule and each local policy. After the system has run some time and we have collected a history of the response time for each rule and local policy, we can then use this historical information as the measure to decide the order of the rule evaluation and local policy evaluation instead of using the estimated cost.

Algorithm 3. Request_Evaluation(q, P)
Input : q is a request regarding policy P

```
    //— step 1 —//
1.  Order rules in RuleSet based on estimated evaluation cost
2.  Reorder rules in RuleSet based on
    types of rule combining algorithms
3.  for each rule r_j in RuleSet
4.      Order all local policies regarding r_j
    //— step 2 —//
5.      for each local policy p_k regarding r_j
6.          Tailor the request q for p_k
7.          if p_k has been evaluated then
8.              obtain the effect of p_k on r_j from IRE table
9.          else
10.             evaluate p_k
11.             store the effect of p_k in the result table
12.             store the evaluation time of p_k in IRE table
13.     store the evaluation time of r_j in the RS table
14.     if the final decision of the r_j can be made then
15.         return the final decision
```

Figure 6: Request Evaluation Algorithm

4. COMPLEXITY ANALYSIS

The two main functions involved in our approach are Policy_Decomposition and Request_Evaluation.

Consider the Policy_Decomposition algorithm shown in Figure 4. Let n_R denote the number of rules in a policy to be decomposed, n_B denote the maximum number of atomic Boolean expressions for each rule in the policy and m denote the maximum number of local policies for each rule in the policy. Steps 2-4 can be executed in time $O(n_B)$. For step 5 we must look at the algorithm in Figure 5. In Figure 5, the complexity of steps 1-5 is dominated by the conversion of rule Boolean expressions to their corresponding DNF which is $O(2^{n_B})$. Note however that in practice n_B is reasonably small [8]. Steps 6-25 involve scanning the local policies and applying the heuristics. We can see that at each step each policy is compared with all other policies and the application of the heuristics results in at most one new policy in each step. This means that the number of steps is proportional to m^2, where m is the total number of local policies in a PDP. The actual application of the heuristics can be executed in time linear in n_B which is the maximum number of atomic Boolean expressions in a local policy. Therefore the complexity here is $O(n_B m^2)$. Coming back to Figure 4, steps 6-9 can be done in time $O(m)$. Hence the overall complexity of the Policy_Decomposition procedure is $O(2^{n_B} + n_R n_B m^2)$. Note that above procedure is executed once for each policy and does not incur any overhead during the request evaluation.

For the Request_Evaluation algorithm (Figure 6), the reordering of rules based on evaluation cost (Steps 5-6) can be done offline in time $O(n_R \log n_R)$. Steps 1-4 can be executed in time $O(n_R)$. Although step 8 will take $O(n_R m \log m)$, we will not consider this in the request evaluation time since it can be executed offline. Steps 9-18, in the worst case where all local policies are unique, the time will be $O(n_R m)$. Hence the request evaluation will be $O(n_R m)$. Note that because of the use of the IRE table as the number of local policies shared by the rules increases the time for request evaluation decreases.

5. RELATED WORK

Several access control models [6, 7, 12, 14, 16] have been proposed for collaborative systems. Akenti [16] uses an authorization model use of authenticated X.509 certificates, in which there is a trusted Akenti server (a PDP) which is contacted by the resource gatekeeper (PEP) when a request is made to access protected resources. The PDP finds all the (possibly distributed) relevant policy certificates from all stakeholders and evaluates them to make a decision. The PRIMA [12] system adopts a decentralized privilege management model in which the main focus is on enabling fine-grained privilege delegation among subjects and on establishing trust relationships among entities from different administrative domains. Approaches like those described in [14] mainly focus on group authorization in grid communities. Cohen et al. [6] propose a family of *coalition-based access control(CBAC)* models, wherein elements required for a

coalition-based access control are layered on top of a simple role-based access control model. Their models also incorporate team-based and task-based access control. Edwards [7] proposed the use of roles to associate policies regarding resources with users for achieving access control in a collaborative system. However, compared to our approach, those approaches have one or more of the following shortcomings: lack of approaches for dealing with sensitive information required for access control; use of simple access control languages; lack of execution strategies for the efficient enforcement of distributed access control.

Lorch et al. [13] experiment with using XACML for providing access control in distributed systems. They describe the use of XACML for implementing the PRIMA authorization model discussed above. Jin and Ahn [10] propose a role-based access control for ad-hoc collaborative environments. They use XACML to express the policies needed to support their framework. In contrast to these approaches, we propose an extension to the reference architecture of XACML in order to support the decomposition of policies for collaborative access control. In addition we provide policy decomposition techniques and strategies for efficient access control enforcement.

The notion of policy decomposition has been mainly used in the context of policy refinement [5, 15]. These approaches typically consider multiple distributed resources in an application as one single abstract hierarchical resource. A high-level access control policy for the abstract resource is transformed to produce local policies that govern access to concrete resources. These local policies are then stored at the PDPs controlling each resource. Unlike our approach in which the policy decomposition is guided by the sensitivity/availability of attribute information necessary for access control and/or user defined constraints at each PDP, the policy decomposition in these approaches is mainly guided by the resource type hierarchy.

Works in the area of privacy preserving access control [11, 9] have proposed techniques to hide sensitive credentials of the request owner and the policies of a policy owner from each other during the request evaluation. Li et al. [11] have proposed a OSBE (Oblivious Signature-Based Envelope) scheme to protect the sensitive credentials of a request owner but the policies are revealed. Frikken et al. [9] have proposed cryptographic protocols that hides both policies and credentials during access control. If we directly apply these approaches to a global policy that involves predicates on sensitive information from multiple parties, during a request evaluation, we can only prevent a requester from knowing the global policy but we need a central PEP to collect sensitive attribute values from each participating party which does not want to share such sensitive information. In comparison, the coordinator in our system only collects decisions from each party but not any sensitive attribute values. In other words, their approaches are not feasible for multi-party collaborative access control in which each party is not willing to disclose its own sensitive information to other parties or the central PEP.

6. CONCLUSION

In this paper, we have proposed a novel access control model for collaborative access control. Our architecture is developed based on the XACML framework which allows our technique to be easily integrated into existing systems. The main idea is to properly decompose a global policy and distribute it to each collaborating party. The decomposition ensures the autonomy and confidentiality of each involved party and guarantees the consistency of the decisions. Additionally, we also propose an algorithm to optimize request evaluation in our system.

Several future research directions exist, one of which is to conduct a simulation study to evaluate the efficiency and effectiveness of our approach. Also, we plan to consider hierarchical relationships among PDPs where each PDP reports the decision to his parent PDP. Another interesting direction is to take into account more complex decomposition constraints which may require the integration of the decomposed policies with policies originally residing at the local PDP.

7. REFERENCES

[1] Parthenon XACML evaluation engine. *http://www.parthenoncomputing.com/xacml_toolkit.html*.

[2] Sun's XACML open source implementation. *http://sunxacml.sourceforge.net*.

[3] Extensible access control markup language (XACML) version 2.0. *OASIS Standard*, 2005.

[4] A. Anderson. Evaluating XACML as a policy language. *Technical report, OASIS*, 2003.

[5] A. K. Bandara, E. C. Lupu, J. Moffett, and A. Russo. A goal-based approach to policy refinement. In *Proc. of the International Workshop on Policies for Distributed Systems and Networks*, page 229, 2004.

[6] E. Cohen, R. K. Thomas, W. Winsborough, and D. Shands. Models for coalition-based access control. In *Proc. of ACM Symposium on Access Control Models and Technologies*, pages 97–106, 2002.

[7] W. K. Edwards. Policies and roles in collaborative applications. In *Proc. of ACM conference on computer supported cooperative work*, pages 11–20, 1996.

[8] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *Proc. of International Conference on Software Engineering*, pages 196–205, 2005.

[9] K. Frikken, M. Atallah, and J. Li. Attribute-based access control with hidden policies and hidden credentials. *IEEE Transaction on Computers*, 55(10):1259–1270, 2006.

[10] J. Jin and G. Ahn. Role-based access management for ad-hoc collaborative sharing. In *Proc. of ACM symposium on Access control models and technologies*, pages 200–209, 2006.

[11] N. Li, W. Du, and D. Boneh. Oblivious signature-based envelope. In *In Proc. of ACM Symposium on Principles of Distributed Computing*, 2003.

[12] M. Lorch, D. B. Adams, D. Kafura, M. S. R. Koneni, A. Rathi, and S. Shah. The prima system for privilege management, authorization and enforcement in grid environments. In *Proc. of International Workshop on Grid Computing*, page 109, 2003.

[13] M. Lorch, S. Proctor, R. Lepro, D. Kafura, and S. Shah. First experiences using XACML for access control in distributed systems. In *Proc. of ACM workshop on XML security*, pages 25–37, 2003.

[14] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In *Proc. of Workshop on Policies for Distributed Systems and Networks*, pages 50–59, 2002.

[15] L. Su, D. W. Chadwick, A. Basden, and J. A. Cunningham. Automated decomposition of access control policies. In *Proc. of International Workshop on Policies for Distributed Systems and Networks*, pages 3–13, 2005.

[16] M. Thompson, A. Essiari, and S. Mudumbai. Certificate-based authorization policy in a pki environment. *ACM Trans. Inf. Syst. Secur.*, 6(4):566–588, 2003.