Conditional Privacy-Aware Role Based Access Control

Qun Ni[1], Dan Lin[1], Elisa Bertino[1], and Jorge Lobo[2]

[1] Department of Computer Science, Purdue University, W. Lafayette, IN 47907, USA
Email: {ni, lindan, bertino}@cs.purdue.edu
[2] IBM Watson Research Center, Hawthorne, NY 10598, USA
Email: jlobo@us.ibm.com

Abstract. Privacy is considered critical for all organizations needing to manage individual related information. As such, there is an increasing need for access control models which can adequately support the specification and enforcement of privacy policies. In this paper, we propose a model, referred to as Conditional Privacy-aware Role Based Access Control (P-RBAC), which supports expressive condition languages and flexible relations among permission assignments for more complex privacy policies. Efficient algorithms for detecting conflicts, redundancies, and indeterminism for a set of permission assignments are presented. In the paper we also extend Conditional P-RBAC to Universal P-RBAC by taking into account hierarchical relations among roles, data and purposes. In comparison with other approaches, such as P3P, EPAL, and XACML, our work has achieved both expressiveness and efficiency.

1 Introduction

Privacy is today a key issue in information technology (IT)[24] and has received increasing attention from consumers, stakeholders, and legislators. Legislative acts, such as the Health Insurance Portability and Accountability Act (HIPAA) [27] for healthcare and the Gramm Leach Bliley Act (GLBA)[28] for financial institutions, require enterprises to protect the privacy of their customers. To address privacy, enterprises have adopted various strategies to protect customer data and to communicate their privacy policies to customers, such as publishing privacy policies on websites [2] possibly based on P3P, or incorporating privacy seal programs (e.g. TRUSTe [25], ESRB, BBBOnline, CPAWebTrust). Those approaches however cannot truly safeguard consumers because they do not address how consumer personal data is actually handled after it is collected. Enterprises' actual practices might intentionally or unintentionally violate the privacy policies published at their websites. Privacy protection can only be achieved by enforcing privacy policies within an enterprise's online and offline data processing systems. Therefore enforceability of privacy policies is the key to a solution for privacy protection.

Conventional access models, such as Mandatory Access Control (MAC) and Discretionary Access Control (DAC), are not designed to enforce privacy policies and barely meet the requirements of privacy protection [8]. However, existing access control technology can be used as a starting point for managing personal identifiable information in a trustworthy fashion [20]. A language used for privacy policies must be the same as or integrated with the language used for access control policies, because both types of policy usually control access to the same resources and should not conflict with one another [3]. Under this promise, we have proposed a family of Privacy-aware Role Based Access Control (P-RBAC) models (see Figure 1) [17] that naturally extend classical RBAC models [7, 23] to support privacy policies. Due to the complexity and variety of privacy policies and privacy requirements from different organizations, we employ a "Divide and Conquer"

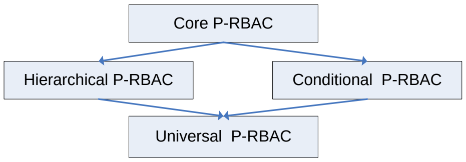

Fig. 1. A family of conceptual P-RBAC models

methodology. That is, the models in our P-RBAC family are designed to meet different levels of requirements and handle different problems. The P-RBAC family includes four models: Core P-RBAC, Hierarchical P-RBAC, Conditional P-RBAC and Universal P-RBAC. Core P-RBAC is the basic model and is able to directly represent privacy-crucial information, such as purpose of data use and obligations. However, although Core P-RBAC can be used to describe commonly used public privacy policies and some acts, the limited expressiveness of its condition language makes it not suitable for representing internally enforceable privacy policies for large scale enterprises and/or complex applications. Specifically, Core P-RBAC has the following limitations. First, Core P-RBAC only supports equality constraints on context variables in finite domains. Second, conditions are restricted to conjunctions of atomic formulas. Third, it only supports one type of relation, that we refer to as AND, among different permission assignments. The type of relation adopted by a set of permission assignments is crucial in determining which obligations need to be executed and which conditions have to be meet when several permissions may apply to the same request[3].

In this paper, we address the aforementioned shortcomings by developing two advanced models, the Conditional P-RBAC and the Universal P-RBAC. Conditional P-RBAC supports more expressive condition languages and more flexible relations between permission assignments. Moreover, we extend the limited analysis operation in [17] to redundancy check, indeterministic obligation enforcement check, conflict check and coverage queries. Universal P-RBAC adds the concept of hierarchy to Conditional P-RBAC, and it is thus able to support more complex requirements. To summarize, our current work has the following five major differences when compared to existing work: 1) Domains, atomic conditions, and relations among permission assignments are carefully crafted to meet the most demanding needs from privacy polices while keeping the complexity of policy analysis tractable; 2) Special structures are proposed to process obligations appearing in multiple permission assignments that can simultaneously apply; 3) Indeterminism in obligation enforcement among policies is identified and a solution is proposed; and 4) Efficient algorithms for detecting conflicts, indeterminism and redundancies of a new permission assignment against *all* existing permission assignments[4] are presented.

2 A Summary of Core P-RBAC

Core P-RBAC [17] is the foundation of the P-RBAC family models. It includes seven sets of entities: Users(U), Roles(R), Data(D), Actions(A), Purposes(P), Obligations(O), and Conditions (C) expressed by a customized language, referred to as LC_0. A user in the Core P-RBAC model is a

[3] In standard policy languages, such as EPAL[10] and XACML[18], the relations between rules are not clearly defined. In order to handle possible interactions or conflicts between rules, EPAL and XACML adopt a simple approach: making only one rule applicable and simply ignoring all other rules. In contrast, relations between permission assignments in P-RBAC models are explicitly defined.

[4] The significance of comparing a new permission assignment against all pre-existing assignments simultaneously as opposed to pair-wisely is elaborated in Section 4.1

human being, and a role represents a job function or job title within the organization with some associated semantics regarding the authority and responsibility conferred on a member of the role. Data in P-RBAC means any information relating to an identified or identifiable individual. An action is an executable image of a program, which upon invocation executes some function for the user. The types of action and data objects that P-RBAC controls depend on the type of system in which they are deployed.

The motivations for introducing Purposes, Conditions, and Obligations in Core P-RBAC originate from OECD Guidelines [19] on the Protection of Privacy and Transborder Flows of Personal Data, current privacy laws in the United States, and public privacy policies of some well-known organizations. The OECD guidelines are, to the best of our knowledge, the most well-known set of private information protection principles, on which many other guidelines, data-protection laws, and public privacy policies are based. Purposes which are bound to actions on data in Core P-RBAC directly reflect the OECD *Data Quality Principle*, *Purpose Specification Principle*, and *Use Limitation Principle*. Purposes are widely used for specifying privacy rules in legislative acts and actual public policies. Obligations, that is, actions to be performed after an action has been executed on data objects, are also part of many privacy policies. Conditions, that is, prerequisites to be met before any action can be executed, are frequent components of privacy policies too.

Core P-RBAC directly models the above notions. In Core P-RBAC, as in classical RBAC, permissions are assigned to roles and users obtain such permissions by being assigned to roles. The distinctive feature of Core P-RBAC lies in the complex structure of privacy permissions, which reflects the highly structured ways of expressing privacy rules to represent the essences of OECD principles and Privacy acts. Hence, aside from the data and the action to be performed on it, a privacy permission explicitly states the intended purpose of the action along with the conditions under which the permission can be granted and the obligations that are to be finally performed. Conditions are represented by conjunction of equality constraints over *context variables*, which record privacy-relevant requirements taken into account when enforcing privacy permissions. The following definition introduces Core P-RBAC. We refer the readers to [17] for additional details.

Definition 1. *The Core P-RBAC model is composed of the following components:*

- *A set U of users, a set R of roles, a set D of data, a set P of purposes, a set A of actions, a set O of obligations, and a condition language LC_0.*
- *The set of Privacy-sensitive Data Permission $PDP = \{(a, d, p, c, o)|\ a \in A,\ d \in D,\ p \in P,\ c$ is an expression of $LC_0,\ o \in \mathcal{P}(O)\}$, where $\mathcal{P}(O)$ denotes the powerset of O.*
- *User Assignment $UA \subseteq U \times R$, a many-to-many mapping user to role assignment relation.*
- *Privacy-sensitive Data Permission Assignment $PDPA \subseteq R \times PDP$, a many-to-many mapping privacy-sensitive data permission to role assignment relation.* □

For simplicity, we use (r, a, d, p, c, o) to denote a permission assignment in the rest of the paper.

3 Conditional P-RBAC

A major shortcoming of Core P-RBAC is the limited expressive power of its condition language LC_0. For example, LC_0 is not able to express conditions like ($DataUser$=“Alice”) OR ($DataUser$=“Bob”) because it only supports conjunction as logical operator. LC_0 cannot deal with conditions like (8am $<$ *currentTime* $<$ 5pm) either because it only supports equality comparisons.

However, enhancing the expressiveness may result in a condition language which is not tractable. In particular, to determine whether a condition in a permission assignment can be satisfied is essentially the classic NP-complete satisfiability problem (SAT) where only a few classes of formulae are tractable. Therefore, for practical purposes, we divide our problem into two subcases, a tractable case and an intractable case, by carefully investigating commonly used conditions in privacy policies. Correspondingly, we define Conditional P-RBAC as characterized by a two-fold solution as follows.

- We define a more expressive condition language LC_1 and introduce the concept of *simple permission assignment set*, for which SAT is tractable.
- We define a fully expressive condition language LC_2 and introduce the concept of *advanced permission assignment set*, for which SAT is theoretically intractable but remains tractable in practice given a reasonable assumption.

3.1 Context Variable Domains and Atomic Conditions

Definition 2. *In both LC_1 and LC_2, conditions are expressed against context variables in the following domains with respective relational operators that have the standard semantics:*

- *Integer domain $\mathcal{I}$ with operators $<, \leq, =, \neq, >, \geq$.*
- *String domain $\mathcal{S}$ with operators $<, \leq, =, \neq, >, \geq$.*
- *Real domain $\mathcal{R}$ with operators $<, \leq, =, \neq, >, \geq$.*
- *Date domain $\mathcal{D}$ with operators $<, \leq, =, \neq, >, \geq$.*
- *Time domain $\mathcal{T}$ with operators $<, \leq, =, \neq, >, \geq$.*
- *A finite tree domain $\mathcal{H}$ with operators $<, \leq, =, \neq, >, \geq, \prec, \preceq, \succ, \succeq, \asymp, \not\asymp$.*
- *A finite partially ordered discrete domain $\mathcal{PO}$ with operators $<, \leq, =, \neq, >, \geq, \prec, \preceq, \succ, \succeq, \asymp, \not\asymp$.*
- *A finite unordered discrete domain $\mathcal{UD}$ with operators $=, \neq$.* □

These domains are commonly used in various kinds of policies including privacy policies. For example, X.500 directories and XML data are in the tree domain; some security labels and role hierarchies are in the partially ordered discrete domain; Boolean values and data subject's consent are in the unordered discrete domain. Most relational operators are easily understood and thus here we only explain some relational operators used in the tree domain and the partially ordered domain. Let x be a context variable in a tree domain T_x and let $v \in T_x$, $x < v$ denotes that x is a descendant of v, while $x \prec v$ means x is a direct descendent(child) of v. Similarly, the operator $>$ represents the ancestor relation while $\succ$ describes the direct ancestor (parent) relation. The operators $\asymp$ and $\not\asymp$ represent comparability and non-comparability tests between domain elements respectively.

Definition 3. *The atomic conditions of LC_1 and LC_2 are defined as follows:*

- *Let D_x be one of the domains introduced by Definition 2; let x_i and x_j be variables in D_x; let v be a constant in D_x; let $op_r \in \{=, \neq\}$; then x_i op_r v is an atomic condition, referred to as* ***equality atomic condition****.*
- *Let D_x be one of the domains introduced by Definition 2 different from domain $\mathcal{UD}$; let x_i and x_j be variables in D_x; let v be a constant in D_x; let $op_r \in \{<, \leq, >, \geq\}$; then x_i op_r v is an atomic condition, referred to as* ***order atomic condition****.*
- *Let D_x be domain $\mathcal{H}$ or domain $\mathcal{PO}$; let x_i and x_j be variables in D_x; let v be a constant in D_x; let $op_r \in \{\prec, \preceq, \succ, \succeq, \asymp, \not\asymp\}$; then x_i op_r v is an atomic condition, referred to as* ***hierarchy atomic condition****.* □

Note that for all domains in Definition 2, except $\mathcal{UD}$, the order atomic condition is more expressive than the equality atomic condition because the equality operation is just a special case of order relation. One typical class of condition in policies are range condition such as $x \in (0, 13]$. Ranges can be easily represented by two order atomic conditions. We also do not define negation of atomic conditions in the totally ordered domain (i.e. integer, real, string, date, and time) as atomic conditions because it can be easily expressed by using corresponding negative relational operators. For example, a negation of atomic condition (*not* OwnerAge $\leq$ 13) can be represented as (OwnerAge $>$ 13).

3.2 The Condition Language LC_1 and Simple Permission Assignment Sets

Given the definition of atomic conditions, we now define LC_1 conditions.

Definition 4. *The conditions of LC_1 are defined as follows:*
- *An atomic condition is a condition of LC_1.*
- *Let c_i and c_j be conditions of LC_1; then $c_i \wedge c_j$ [5] is a condition of LC_1.* □

When dealing with multiple permission assignments including conditions and obligations, it is fundamental to understand the semantics associated with the permission when multiple assignments can be applied. For this purpose, we introduce two possible relations AND and OR. An AND relation for a set of permission assignments indicates that an access request related to these permission assignments will be authorized only if all conditions in these permission assignments are satisfied and all obligations are fulfilled thereafter. Alternatively, an OR relation for a set of permission assignments indicates that an access request related to these permission assignments will be authorized if one of the conditions in these permission assignments is satisfied and only the corresponding obligations in that permission assignment are fulfilled thereafter (more details about AND and OR relation are presented in Section 4.1). To handle AND and OR relations, we introduce the concept of Simple Permission Assignment Sets (SPAS).

Definition 5.
- *An atomic simple permission assignment set is a set $\{PA_1, PA_2, ..., PA_k\}$, such that the relation among the permission assignments in the set is AND.*
- *Let $SPAS_1$, ..., $SPAS_n$ be atomic SPASs, then $\{SPAS_1, ..., SPAS_n\}$ is a non-atomic SPAS, if (i) the relation among atomic SPAS's is OR; and (ii)$SPAS_i \cap SPAS_j = \emptyset$, $i, j \in [1..n] \wedge i \neq j$.*
- *An atomic SPAS is a SPAS; a non-atomic SPAS is a SPAS.* □

Many permission can be expressed using SPAS. e.g., SPAS allows different groups or departments to define their own permission assignments in one or several permission sets. Also, SPAS helps to specify the relation OR between permission assignments. Organizational privacy policies can then be represented by a finite number of atomic SPASs. Consider the following example: “Marketing employee can only access customers' email address for promotion if the customers are not under 13 and allow them to do so. If they are under 13, they need to get their parents' consent”. The corresponding SPAS is as follows.

Example 1. $SPAS \equiv \{SPAS_1, SPAS_2\}$; $SPAS_1 \equiv$ {(MarketingEmployee, Read, EmailAddress, Promotion, OwnerAge $>$ 13 $\wedge$ OwnerConsent=Yes, $\emptyset$)}; $SPAS_2 \equiv$ {(MarketingEmployee, Read, EmailAddress, Promotion, OwnerAge $\leq$ 13 $\wedge$ ParentalConsent=Yes, $\emptyset$)}. □

The rationale behind LC_1 and $SPAS$ is to provide good expressiveness while guaranteeing the efficient generation of disjunctive OR forms by permission assignment normalization. Disjunctive normal form and permission assignment normalization ensure the efficiency of our analysis algorithms. We will detail these concepts and analysis in Section 4.

[5] To avoid ambiguities, Boolean operators $\wedge$ and $\vee$ will be used in predicate conditions, while AND and OR will be used to denote relations between permission assignments

3.3 The Condition Language LC_2 and Advanced Permission Assignment Sets

Some applications may require the ability to specify more complex conditions that need both Boolean operators $\wedge$ and $\vee$. For example, the condition $(OwnerAge \leq 13 \wedge ParentalConsent = Yes) \vee (OwnerAge > 13 \wedge OwnerConsent = Yes)$. We define the language LC_2 to cover these cases.

Definition 6. *The conditions of LC_2 are defined as follows:*

- *An atomic condition is a condition of LC_2.*
- *Let c_i and c_j be conditions of LC_2; then $c_i \wedge c_j$ and $c_i \vee c_j$ are conditions of LC_2.* □

Along with LC_2, an Advanced Permission Assignment Set(APAS) is defined to support the representation of different relations among permission assignments.

Definition 7. *Let S be a set of all possible permission assignments.*

- *An atomic APAS is a tuple $N[rel, pas, \emptyset]$, where N is an identifier, rel $\in${AND, OR} and pas is a finite subset of S.*
- *Let rel $\in${AND,OR}, pas is a finite subset of S, and apas be a set of APAS; then a $N[rel, pas, apas]$ is an APAS.* □

Example 2. Let PA_1, PA_2, ..., PA_{14} be permission assignments. An example APAS is $APAS_1$ [AND, $\{PA_1, PA_2\}$, $\{APAS_2$ [AND, $\{PA_3, PA_4\}$, $\{APAS_3$ [OR, $\{PA_5, PA_6\}$, $\emptyset$], $APAS_4$ [OR, $\{PA_7, PA_8\}$, $\emptyset$]$\}$], $APAS_5$ [OR, $\{PA_9, PA_{10}\}$, $\{APAS_6$ [AND, $\{PA_{11}, PA_{12}\}$, $\emptyset$], $APAS_7$ [OR, $\{PA_{13}, PA_{14}\}$, $\emptyset$]$\}$]$\}$], which can be represented as a tree(see Figure 2). □

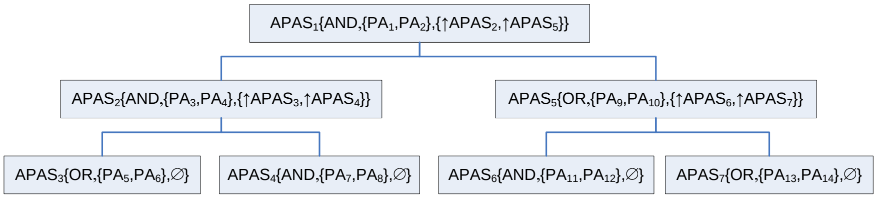
APAS$_1${AND,{PA$_1$,PA$_2$},{↑APAS$_2$,↑APAS$_5$}}
APAS$_2${AND,{PA$_3$,PA$_4$},{↑APAS$_3$,↑APAS$_4$}}
APAS$_5${OR,{PA$_9$,PA$_{10}$},{↑APAS$_6$,↑APAS$_7$}}
APAS$_3${OR,{PA$_5$,PA$_6$},∅}
APAS$_4${AND,{PA$_7$,PA$_8$},∅}
APAS$_6${AND,{PA$_{11}$,PA$_{12}$},∅}
APAS$_7${OR,{PA$_{13}$,PA$_{14}$},∅}

Fig. 2. An APAS tree

The advantage of APAS is that it provides a natural and flexible way to help administrate different levels of permission assignments. Example 2 could represent a company with two departments D_1 and D_2. D_1 has teams T_1 and T_2, and D_2 has teams T_3 and T_4. We may allow a senior privacy officer to administrate the whole APAS tree, and departmental privacy officers to maintain $APAS_2$ and $APAS_5$ respectively. If necessary, a privacy officer can also be assigned to several APAS nodes in the tree.

4 Consistency Checking in Conditional P-RBAC

In P-RBAC, when a new permission assignment is entered, the privacy officer needs to check how the new permission assignment interacts with existing ones. We refer to such task as **consistency checking** of permission assignments.

Definition 8. *A new permission assignment is consistent with pre-existing permission assignments if none of the following conditions hold:*

- ***Redundancy**. A permission assignment x is redundant with respect to a group of permission assignments $Y \equiv \{y_1, ..., y_n\}(n \geq 1)$ if the addition of x does not affect the behavior of the system governed by Y.*
- ***Conflict**. A permission assignment x conflicts with a group of permission assignments $Y \equiv \{y_1, ..., y_n\}(n \geq 1)$ if the addition of x results in that one action of the system governed by Y can be never carried out or there exists a conflict among new obligations.*
- ***Indeterminism**. A permission assignment x results in indetermination with respect to a group of permission assignments $Y \equiv \{y_1, ..., y_n\}(n \geq 1)$ if the addition of x results in that the enforcement of obligations governed by Y becomes nondeterministic.* □

Here the unchanged behavior in the definition of redundancy means given any data request, the system will make the same decision and execute the same set of obligations. Conflict happens if (i) the new condition created after the addition of x cannot be satisfied; or (ii) the new obligations introduced by x need to be added to a set of obligations of a permission assignment and the new obligations conflict with the set. Indeterminism arises because of the relations between conditions and obligations in privacy policies. For example, if there is a permission assignment ($Marketing-Employee$, *read*, *EmailAddress*, *promotion*, *ownerage* ≤ 13, $notify$(*byPhone*, *optout*)), a new permission assignment($MarketingEmployee$, *read*, *EmailAddress*, *promotion*, *ownerage* ≤ 19, $notify$(*byEmail*)) that has OR relation with respect to the original permission assignment results in indeterministic obligation enforcement. For a kid who is ten, enforcement of $notify$(*byPhone*, *optout*) or $notify$(*byEmail*) is undetermined to system. Any policy language that supports both pre-conditions and post-actions may suffer from such a problem.

Based on the result of consistency checking, the privacy officer will accept or reject new permission assignments, resolve potential conflicts, or mark certain permission assignments as being inactive. Consistency checking can also include *coverage queries*. In some cases, the privacy officer may want to know if the permission assignments have been defined for a certain range of context variables. For example, a privacy officer may want to know if third parties can access purchase order information for research purposes between 19:00 and 22:00. In what follows, we present a normalization technique to carry out the above analysis in Conditional P-RBAC.

4.1 Permission Assignment Normalization

In Conditional P-RBAC, permission assignments are maintained as a SPAS or an APAS tree. Directly using such a set or tree structure to answer data requests or to detect whether there exists a conflict between a new permission assignment and the pre-existing permission assignments, may not be efficient because sometimes the entire set or the entire tree need to be traversed to find an answer. Therefore, it would be helpful to translate a SPAS (an APAS tree) into a form better suited for analysis; we call such a translation *permission assignment normalization*.

Observe that in a group of permission assignments, either in a SPAS or in an APAS, two permission assignments may interact with each other only when they share the same role, action, data and purpose. Otherwise, the permission assignments are incomparable[6]. Therefore, the goal

[6] The statement is not true if role hierarchies, data hierarchies and purpose hierarchies are considered. Such situation is discussed in Universal P-RBAC.

of permission assignment normalization is to generate a new permission assignment set such that each combination of (role, action, data, purpose) only appears once in the set.

The benefit of the normalization for answering data access requests is obvious. Now the system can give an answer within constant time by using a hashing function $\mathcal{H}(r, a, d, p)$ to locate the permission assignment being queried. The same hashing function can be used to improve the efficiency of the consistency checking. It is worth noting that the normalization is extremely helpful in determining the relation between a new permission assignment and a group of permission assignments because a series of related permission assignments will become *one* permission assignment after the normalization. It is not sufficient to compare a new permission assignment against each existing permission assignment. For example, let D be a finite domain $\{a, b, c\}$ and x be a context variable on D, let P_1 and P_2 be two existing permission assignments with conditions $x \neq a$ and $x \neq b$ respectively, let P_3 be the new permission assignments with condition $x \neq c$, and we assume the other components of P_1, P_2 and P_3 are the same and they have an AND relation. Obviously P_3 does not conflict individually with P_1 or P_2, but conflicts with the integration of P_1 and P_2.

Definition 9. *Let S and S' be two permission assignment sets, we say the behavior of S' is equivalent to that of S if for any data access request, S' yields the same authorization decision and performs the same obligations as S.* □

The normalization is challenging because we must guarantee that the behavior of a normalized permission assignment is equivalent to the original assignments. The difficulty lies in the analysis of conditions and obligations. In the following, we discuss the procedures for normalizing SPAS and APAS separately.

Permission Assignment Normalization on SPAS To facilitate permission assignment normalization on SPAS, we first introduce the following structure.

Definition 10. *Let R be a set of roles, D be a set of data, P be a set of purposes, A be a set of actions, O be a set of obligations in Conditional P-RBAC; a condition-obligation structure is a set of tuples of the form (c, o) where c is a condition of LC_1 and $o \in \mathcal{P}(O)$; a normalized permission assignment is a 5-tuple (r, a, d, p, co) where $r \in R$, $a \in A$, $d \in D$, $p \in P$, and co is a condition-obligation structure.* □

The normalization algorithm for SPAS consists of two steps. First, for permission assignments with the same (role, action, data, purpose) in the same SPAS, we combine their conditions using the Boolean operator $\wedge$, and associate the new condition with the UNION of corresponding obligations. Second, we construct the condition-obligation structure for permission assignments with the same (role, action, data, purpose) in different SPASs. Given a normalized permission assignment (r, a, d, p, co) where $co = \{(c_i, o_i) \mid 0 < i < k\}$ and k is the number of atomic SPASs in the SPAS, if a single c_i is satisfied, the data access request is allowed and the corresponding obligations in o_i are performed later. The pseudo codes of the algorithms are shown in Figures 3 and 4. The time complexity of each algorithm is $O(n)$ assuming the number of permission assignments is n. We use Example 3 to illustrate ideas in the algorithms.

Example 3. Consider a SPAS containing the following atomic SPASs:
$SPAS_1((r_{11}, a_{11}, d_{11}, p_{11}, c_{11}, o_{11}), (r_{12}, a_{12}, d_{12}, p_{12}, c_{12}, o_{12}), (r_{13}, a_{13}, d_{13}, p_{13}, c_{13}, o_{13}))$,
$SPAS_2((r_{21}, a_{21}, d_{21}, p_{21}, c_{21}, o_{21}), (r_{22}, a_{22}, d_{22}, p_{22}, c_{22}, o_{22}), (r_{23}, a_{23}, d_{23}, p_{23}, c_{23}, o_{23}))$,
$SPAS_3((r_{31}, a_{31}, d_{31}, p_{31}, c_{31}, o_{31}), (r_{32}, a_{32}, d_{32}, p_{32}, c_{32}, o_{32}), (r_{33}, a_{33}, d_{33}, p_{33}, c_{33}, o_{33}))$.

```
**Algorithm CO-Normalization**($NSPAS$)
**Input**: $NSPAS$ is a non-atomic SPAS with respect to the same (role, action, data, purpose)
$NPAL \leftarrow$ **nil**; // $NPAL$ is a normalized permission assignment list
$ConditionObligationStructure \leftarrow \emptyset$;
**for** each atomic $SPAS$ in the $NSPAS$
    $(c, o) \leftarrow (\mathbf{true}, \emptyset)$;
    **for** each permission assignment $(r', a', d', p', c', o')$ in $SPAS$
        $(c, o) \leftarrow (c \wedge c', o \cup o')$;
    $ConditionObligationStructure \leftarrow ConditionObligationStructure \cup (c, o)$;
$NPAL \leftarrow$ List.CONS($(role, action, data, purpose, ConditionObligationStructure), NPAL$);
**return** $NPAL$.
```

Fig. 3. CO-Normalization algorithm

```
**Algorithm SPAS-Normalization**($NSPAS$)
**Input** : $NSPAS$ is a non-atomic SPAS
$NPAL \leftarrow$ **nil**; // $NPAL$ is a normalized permission assignment list;
divide $NSPAS$ into $\{NSPAS_1, NSPAS_2, ..., NSPAS_n\}$,
where $NSPAS_i$ consists of permission assignment with same (role, action, data, purpose);
**for** $i \leftarrow 1$ to $n$
    $NPAL \leftarrow$ List.CONS(CO-Normalization($NSPAS_i$),$NPAL$);
**return** $NPAL$.
```

Fig. 4. SPAS-Normalization algorithm

We assume $(r_{11}, a_{11}, d_{11}, p_{11}) = (r_{21}, a_{21}, d_{21}, p_{21}) = (r_{22}, a_{22}, d_{22}, p_{22}) = (r_{31}, a_{31}, d_{31}, p_{31}) = (r_{32}, a_{32}, d_{32}, p_{32}) = (r_{33}, a_{33}, d_{33}, p_{33})$.

Suppose there is a data request DR concerning $(r_{11}, a_{11}, d_{11}, p_{11})$. Several possible cases exist according to the definition of SPAS:

- If DR satisfies c_{11}, the request will be authorized and obligations in o_{11} will be performed.
- If DR satisfies $c_{21} \wedge c_{22}$, then the request will be authorized and obligations in $o_{21} \cup o_{22}$ will be performed. The intuition of the union of obligations is as follows:
  - Since DR satisfies c_{21} in permission $(r_{21}, a_{21}, d_{21}, p_{21}, c_{21}, o_{21})$, obligations in o_{21} should be performed.
  - Since DR satisfies c_{22} in permission $(r_{22}, a_{22}, d_{22}, p_{22}, c_{22}, o_{22})$, obligations in o_{22} should be performed.
  - Duplicated obligations should be performed only once because generally several enforcements of a same obligation do not make sense.
- If DR satisfies $c_{31} \wedge c_{32} \wedge c_{33}$, then the request will be authorized and obligations in $o_{31} \cup o_{32} \cup o_{33}$ will be performed.
- Otherwise, the request will be denied.

Then, the normalized permission assignment set is :(r', a', d', p', co'), $(r_{12}, a_{12}, d_{12}, p_{12}, \{(c_{12}, o_{12})\})$, $(r_{13}, a_{13}, d_{13}, p_{13}, \{(c_{13}, o_{13})\})$, $(r_{23}, a_{23}, d_{23}, p_{23}, \{(c_{23}, o_{23})\})$, where $r' = r_{11}$, $a' = a_{11}$, $d' = d_{11}$, $p' = p_{11}$, and $co' = \{(c_{11}, o_{11}), (c_{21} \wedge c_{22}, o_{21} \cup o_{22}), (c_{31} \wedge c_{32} \wedge c_{32}, o_{31} \cup o_{32} \cup o_{32})\}$. □

Based on the definition of condition-obligation structure, it is easy to prove the following lemma:

Lemma 1. *Algorithm CO-Normalization and SPAS-Normalization guarantee that the behavior of the normalized permission assignment set is equivalent to that of the original simple permission assignment set.* □

Permission Assignment Normalization on APAS The main difference between SPAS and APAS is the use of Boolean relation $\vee$ between conditions and the relation OR between permission assignments. However, we can still apply the same idea underlying the normalization of a SPAS to normalize an APAS tree because as in SPAS, permission assignments with different (role, action, data, purpose) in an APAS tree do not interfere with one another. The main challenge is again the processing of obligations. In order to solve the problem, we introduce a new concept, referred to as **condition-obligation binding**. The idea behind this concept is that the fact that the obligations must be fulfilled depends on the conditions satisfied by a data access request.

Definition 11. *Let c be a condition expressed according to LC_2, O be a set of obligations and $o \in \mathcal{P}(O)$. $[c, o]$ is a condition-obligation binding. If $[c_i, o_i]$ and $[c_j, o_j]$ are condition-obligation bindings, $[c_i, o_i] \wedge [c_j, o_j]$ and $[c_i, o_i] \vee [c_j, o_j]$ are condition-obligation bindings too. Further, $[c, o]$ is called a normal condition-obligation binding if c is a condition in LC_1 (i.e. a conjunction of atomic conditions).* □

Lemma 2. *A condition-obligation binding supports the following transformations:*

- $[c_i \vee c_j, o] \Leftrightarrow [c_i, o] \vee [c_j, o]$.
- $[c_i \wedge (c_j \vee c_k), o] \Leftrightarrow [c_i \wedge c_j, o] \vee [c_i \wedge c_k, o]$.
- $[c_i, o_i] \wedge [c_j, o_j] \Leftrightarrow [c_i \wedge c_j, o_i \cup o_j]$. □

The normalization algorithm for an APAS tree is as follows. First, we transform all permission assignments in the APAS tree into a new form $(r, a, d, p, [c, o])$. Second, we remove all relation operators and sub-trees by moving relation operators into condition-obligation bindings. For example, given $(r, a, d, p, [c_i, o_i])$ OR $(r, a, d, p, [c_j, o_j])$, we have $(r, a, d, p, [c_i, o_i] \vee [c_j, o_j])$. After this step, we obtain a set of permission assignments in the form of $(r, a, d, p, \sqcup_{i=1}^{n}[c_i, o_i])$ where $\sqcup \in \{\wedge, \vee\}$. Next, we convert $\sqcup_{i=1}^{n}[c_i, o_i]$ into $\vee_{j=1}^{m}[c_j, o_j]$, where $[c_j, o_j]$ is a normal condition-obligation binding, by using the transformations given in Definition 11. Finally, we transform $\vee_{j=1}^{m}[c_j, o_j]$ into a condition-obligation structure and generate a set of normalized permission assignments. The pseudo code is omitted due to space constraints. The following example illustrates the algorithm.

Example 4. Consider Example 2. Assuming that $APAS_1$ contains the following permission assignments: $PA_3 = (r_3, a_3, d_3, p_3, c_3, o_3)$, $PA_8 = (r_8, a_8, d_8, p_8, c_8, o_8)$ $PA_9 = (r_9, a_9, d_9, p_9, c_9, o_9)$, $PA_{13} = (r_{13}, a_{13}, d_{13}, p_{13}, c_{13}, o_{13})$ where $r_3 = r_8 = r_9 = r_{13}, a_3 = a_8 = a_9 = a_{13}, d_3 = d_8 = d_9 = d_{13}, p_3 = p_8 = p_9 = p_{13}$. The following steps are executed by the algorithm.

Step 1: Group permission assignments according to (role, action, data, purpose) and construct condition-obligation bindings, where we have:
$PA_3 = (r_3, a_3, d_3, p_3, [c_3, o_3])$, $PA_8 = (r_8, a_8, d_8, p_8, [c_8, o_8])$
$PA_9 = (r_9, a_9, d_9, p_9, [c_9, o_9])$, $PA_{13} = (r_{13}, a_{13}, d_{13}, p_{13}, [c_{13}, o_{13}])$

Step 2: Flatten the APAS tree by moving the relational operators into the permission assignments. We obtain NPA'=$(r_3, a_3, d_3,\ p_3, [c_3, o_3]\ \wedge[c_8, o_8]\ \wedge([c_9, o_9]\ \vee[c_{13}, o_{13}]))$. We assume that c_3, c_8, c_9 and c_{13} are atomic conditions (a more general case of conditions is shown in our technical report).

Step 3: Transform the condition-obligation bindings in NPA' into a DNF as shown below.

$[c_3, o_3] \wedge [c_8, o_8] \wedge ([c_9, o_9] \vee [c_{13}, o_{13}]) \Rightarrow [c_3 \wedge c_8 \wedge c_9, o_3 \cup o_8 \cup o_9] \vee [c_3 \wedge c_8 \wedge c_{13}, o_3 \cup o_8 \cup o_{13}]$.

Step 4: Construct the condition-obligation structure and generate the normalized permission assignment: $NPA = (r_3, a_3, d_3, p_3, \{(c_3 \wedge c_8 \wedge c_9, o_3 \cup o_8 \cup o_9),\ (c_3 \wedge c_8 \wedge c_{13}, o_3 \cup o_8 \cup o_{13})\})$ □

It is worth noting that the disjunctive normal form transformation for the condition-obligation bindings may be exponential to the number of atomic conditions. However, such situation rarely happens in practice due to the following observations. First, in real privacy policies, for each flattened permission assignment, the number of atomic conditions in the conditions is usually very small (e.g. ≤ 10). Second, the APAS-Normalization is linear with respect to the number of permission assignments, which has no direct relation with the total number of context variables. In other words, even if the total number of context variables were tens of thousands, the running time of our APAS-Normalization will still be linear in the total number of permission assignments.

4.2 Permission Assignment Maintenance

In conditional P-RBAC, we guarantee that there is no redundancy, indeterminism or conflict between a new permission and a pre-existing permission assignment set by taking the following steps when inserting a new permission assignment is issued.

1. Redundancy checking. If no error occurs, continue.
2. Conflict detection. If no error occurs, continue.
3. Indeterminism checking. If no error occurs, insert the new permission assignment.

Definition 12. *Let $NPAL$ be a normalized permission assignment set based on either a SPAS or an APAS set, PA' be a new permission assignment, NPA is the normalized permission assignment which have the same $(role, action, data, purpose)$ as PA', and NPA' is the normalized permission assignments of the addition of PA' in the pre-existing permission assignments.*

- *If either a condition of NPA' is not satisfiable or an obligation conflict is detected, we say PA' strongly conflicts with $NPAL$.*
- *Let $CO = \{(c_1, o_1), ..., (c_n, o_n)\}$ be a condition-obligation structure of NPA'. If a c_i, for $i \in [1, n]$, is not satisfiable, we say PA' weakly conflicts*[7] *with $NPAL$.*
- *If the context variable domain of NPA is the same as that of NPA' and the corresponding obligation sets are equivalent, we say PA' is redundant with respect to $NPAL$.*
- *Let $CO = \{(c_1, o_1), ..., (c_n, o_n)\}$ be a condition-obligation structure of NPA'. If there exist two tuple (c_i, o_i), $(c_j, o_j) \in CO$ such that $c_i \wedge c_j$ is satisfiable and $o_i \neq o_j$, we say PA' causes indeterminism of obligation enforcement in $NPAL$.* □

The coverage query can be very generic and depends on requirements and their implementations, therefore no formal definition is given here. The general case of coverage queries is that given some constraints on role, data, purpose, and context variables, the system checks whether they are satisfiable or unsatisfiable based on a permission assignment set.

Given the definition of redundancy, strong conflict, weak conflict, and indeterminism, our permission assignment normalization algorithms, and the domain elimination algorithms discussed in [1], the problems of redundancy checking, indeterminism checking, conflict detection and coverage queries are converted into a tractable satisfiability problem. We do not include more details due to space limitation.

[7] Weak conflict may indicate potential problems introduced by a new permission assignment because it causes some (c_i, o_i) to be totally useless.

5 Universal P-RBAC

Universal P-RBAC combines Hierarchical P-RBAC and Conditional P-RBAC, and inherits both their features. Such integration of Hierarchical P-RBAC and Conditional P-RBAC supports the specification of more complex relations between different permission assignments, which in turn raises several issues with respect to consistency check.

5.1 Hierachical P-RBAC

Hierarchical P-RBAC provides role hierarchies (RH), data hierarchies (DH) and purpose hierarchies (PH). Role hierarchies represent an important notion in RBAC [23, 7], which reflect organization's lines of authority and responsibility. Mathematically, role hierarchies are partial orders. The purpose hierarchy is represented as a tree, where each purpose (except the root purpose) has exactly one parent purpose and there are no cycles. A parent node represents a more general purpose than its children nodes. Access for a parent purpose is allowed only when the access for all its children purpose is allowed. Like the purpose hierarchy, the data hierarchy is also a tree structure. Access to a parent data object is allowed only if access to all its children is allowed. Introducing the hierarchy concept compacts permission assignments (e.g., permission assignments with different purposes may be clustered providing all the child purposes are already covered), and also complicates consistency check.

5.2 Interactions between Hierarchical P-RBAC and Conditional P-RBAC

As mentioned in previous section, inserting a new permission assignment requires checking redundancy, conflict and indeterminism. When there is no hierarchy (in the Conditional P-RBAC), those checks are carried out only on the permissions with the same (role, data, action, purpose) because each role (data, action or purpose) is independent of any of other roles. Once we introduce a hierarchy (in Universal P-RBAC), the situation becomes more complex. We now need to compare permission assignments of different roles (data, or purpose) since potential interactions may exist among these roles due to their hierarchical relations. To facilitate the discussion of such interactions, let us assume a new permission assignment to be PA_n =(r_n, a_n, d_n, p_n, c_n, o_n).

The process of issuing PA_n includes two phases. The first phase checks if PA_n causes any redundancy, conflict or indeterminism problem against the existing permission assignment sets of role r_n, r_a and r_d, respectively, which is carried out in a temporary copy of existing permission assignment sets. If PA_n passes the check, the second phase will then update all influenced permission assignments.

In the first phase, there are four steps. First, we "virtually" [8] insert PA_n into current SPAS or APAS. Let NPA_n and NPA'_n be the normalized permission assignments containing (r_n, a_n, d_n, p_n) before and after the insertion respectively. If a strong or weak conflict or indeterminism is detected during the construction of NPA'_n, or NPA'_n is redundant compared to NPA_n, the processing stops. Otherwise, we proceed to the second step which handles the effect of the data hierarchy. From here, our discussion is based on the normalized permission assignment set after the virtual insertion of PA_n. We compare $NPA'_n = \{(c_{n_1},o_{n_1}), \cdots, (c_{n_i},o_{n_i})\}$ with every such permission assignment $NPA'_x = (r_n,\ a_n,\ d_x,\ p_n,\ \{(c_{x_1},o_{x_1}), \cdots, (c_{x_j},o_{x_j})\})$, where d_x is a descendant or an ancestor of

[8] The word "virtually" means the operation does not have any real effect on the system.

d_n, denoted as $d_x \preceq d_n$ and $d_x \succeq d_n$ respectively. If NPA'_n provides broader authorizations than its previous version NPA_n, we need to correspondingly increase the authorization on the data which is a descendant of d_n. The reason is that according to the definition of the data hierarchy, if a user can access data d_n under a certain condition, he should also be able to access data d_x ($d_x \preceq d_n$) under the same condition. The increase of the authorization is achieved by combining the condition-obligation bindings of NPA'_n and NPA'_x. Specifically, the new permission assignment for d_x is $NPA''_x = (r_n,\, a_n,\, d_x,\, p_n,\, \{(c_{x_1},o_{x_1}),\, \cdots,\, (c_{x_j},o_{x_j})\} \bigcup \{(c_{n_1},o_{n_1}),\, \cdots,\, (c_{n_i},o_{n_i})\})$. During the combination, we need to check if there exists indeterminism of obligation enforcement.

In the other case when NPA'_n is stricter than before, we need to check NPA'_x with $d_x \succeq d_n$. If the solution domain of c_x is covered by c_n, no more changes are needed according to the same reason above. Otherwise, it means that c_x defines some situations which cannot be satisfied by c_n. In other words, there are some permissions authorized by NPA'_x but not authorized by NPA'_n, which conflicts with the functionality of the data hierarchy. Therefore, we remove NPA'_x and dispatch its permission to its child nodes except d_n. For example, if d_y is a child node of d_x ($d_y \neq d_n$), it will receive a permission assignment $PA_y = (r_n,\, a_n,\, d_y,\, p_n,\, \{(c_{x_1},o_{x_1}),\, \cdots,\, (c_{x_j},o_{x_j})\})$. After that, we need to normalize the permission assignment sets again.

Next, we consider the purpose hierarchy. The processing is omitted because the purpose hierarchy has the same structure as the data hierarchy.

The final step in the first phase is to propagate the changes to the ancestor roles of r_n. The basic rule is to guarantee that parent roles have all the permissions of their child roles. The specific operation is as follows. If an updated permission assignment of r_n is different from its previous version, we need to replace the correspondingly inherited permission assignment for its parent roles with the new one and renormalize permissions for its parents. After the normalization, if the parent roles obtain different permissions, we repeat the procedure for the corresponding grandparent roles. Note that these changes may be propagated all the way to the top of the role hierarchy.

All the permission assignments modified in the first phase are made in a temporary copy of the original permission assignments because the process may stop at any time due to conflict, indeterminism, or redundancy problems. We finally update all these changes to the system in the second phase.

Our maintenance algorithm may look complicated. However, it is worth noting that the frequency of policy changes (i.e. permission insertion) is much less than that of data requests. By taking care of all possible issues during the insertion phase which needs to be executed only once, we are then able to reduce response time for each data request.

6 Related Work

In this section, we compare our proposal to three proposals that are most closely related, that is P3P[29], EPAL[10] and XACML [18]. P3P enables websites to express their privacy practices in a standard format that can be retrieved automatically and interpreted easily by agents. However, P3P is not able to describe complex conditions like the age constraint, and it is also not an enforceable policy language. EPAL [4] is proposed to encode enterprise's privacy-related data-handling policies and practices, which can be imported and enforced by a privacy-enforcement system. XACML [18] is a well known access control model based on XML. Its main goal is to provide an application independent policy language which enables the use of arbitrary attributes in different types of policies. Both EPAL and XACML aim at providing large flexibilities of writing policies, but leave the policy analysis task to policy analyzers. For example, they use a very simple strategy to handle conflicts among rules. That is, when multiple rules in one policy yield different decisions for a same request, EPAL and XACML will simply choose the decision from one rule according to the rule combining algorithm and ignore the effects of other rules. One of such strategies, i.e. first applicable rule, may cause problems as discussed in [5]. In addition, obligation processing is rather preliminary in both EPAL and XACML. Unlike existing approaches, our models achieve a balance between expressiveness and tractability, and also guarantee that the insertion of a new policy will not affect the consistency of existing policies.

Besides the policy languages, we are also aware of analysis tools for XACML policies, such as [9, 14, 26]. Most of them simplify the analysis and focus on core functions only. It is not clear if they can be easily extended to support analysis on the full functionality. Since they are orthogonal to our work, we do not present the details here. In the definition of domain and atomic conditions, we refer to work on constraint databases [12, 15, 21, 22]. Compared to other works on obligations[6, 11, 16], our idea on condition-obligation binding and indeterminism is new.

7 Conclusion

In this paper, we proposed Conditional P-RBAC and Universal P-RBAC for specifying complex privacy policies. The key design criterion is to balance efficiency and expressiveness. The definition of domains and atomic conditions are carefully chosen to reflect the wide needs for enforceable privacy policies and to meet our efficiency goal, so does the design of condition languages and permission assignment sets. We have taken into account the effect of hierarchical relations among roles, data and purposes, which further enhance the expressiveness of our approach. As part of future work, we plan to introduce a sticky policy paradigm[13] into P-RBAC and develop a formal method to describe and manage obligations and to automatically detect possible conflicts between obligations and between obligations and actions.

8 Acknowledgement

The work reported in this paper has been partially supported by IBM under the OCR project “Privacy and Security Policy Management”. Participants to this project are: Carnegie Mellon University, IBM T.J. Watson Research Center, Purdue University.

References

1. D. Agrawal, J. Giles, K.-W. Lee, and J. Lobo. Policy ratification. In *POLICY ’05: Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY’05)*, pages 223–232, Stockholm Sweden, 2005. IEEE Computer Society.
2. Amazon.com. Amazon privacy notice. Available at http://www.amazon.com/exec/obidos/tg/browse/-/468496/102-8997954-0573735.
3. A. H. Anderson. A comparison of two privacy policy languages: Epal and xacml. In *SWS ’06: Proceedings of the 3rd ACM workshop on Secure web services*, pages 53–60, New York, NY, USA, 2006. ACM Press.
4. P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. Enterprise privacy authorization language (epal 1.2). W3C Member Submission 10 November 2003. Available at http://www.w3.org/Submission/EPAL/.

5. A. Barth, J. C. Mitchell, and J. Rosenstein. Conflict and combination in privacy policy languages. In *WPES '04: Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 45–46, New York, NY, USA, 2004. ACM Press.
6. C. Bettini, S. Jajodia, X. Wang, and D. Wijesekera. Obligation monitoring in policy management. In *POLICY '02: Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02)*, page 2, Washington, DC, USA, 2002. IEEE Computer Society.
7. D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001.
8. S. Fischer-Hubner. *IT-security and privacy: design and use of privacy-enhancing security mechanisms.* Springer-Verlag New York, Inc., New York, NY, USA, 2001.
9. K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 196–205, 2005.
10. IBM Zurich Research Laboratory,Switzerland. The enterprise privacy authorization language(epal 1.1). Available at http://www.zurich.ibm.com/security/enterprise-privacy/epal/.
11. K. Irwin, T. Yu, and W. H. Winsborough. On the modeling and analysis of obligations. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 134–143, New York, NY, USA, 2006. ACM Press.
12. P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query languages (preliminary report). In *PODS '90: Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 299–313, New York, NY, USA, 1990. ACM Press.
13. G. Karjoth, M. Schunter, and M. Waidner. Platform for enterprise privacy practices: Privacy-enabled management of customer data. In *Privacy Enhancing Technologies*, pages 69–84, 2002.
14. V. Kolovski, J. Hendler, and B. Parsia. Formalizing xacml using defeasible description logics. Available at http://www.mindswap.org/ kolovski/xacml_tr.pdf.
15. N. Li and J. C. Mitchell. Datalog with constraints: A foundation for trust management languages. In *PADL '03: Proceedings of the 5th International Symposium on Practical Aspects of Declarative Languages*, pages 58–73, London, UK, 2003. Springer-Verlag.
16. M. C. Mont and F. Beato. On parametric obligation policies: Enabling privacy-aware information lifecycle management in enterprises. Tech Report HPL-2007-7, Trusted Systems Laboratory, HP Laboratories Bristol. Available at http://www.hpl.hp.com/techreports/2007/HPL-2007-7.pdf.
17. Q. Ni, A. Trombetta, E. Bertino, and J. Lobo. Privcy aware role based access control. In *SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies*, New York, NY, USA, 2007. ACM Press.
18. OASIS. extensible access control markup language (xacml) 2.0. Available at http://www.oasis-open.org/.
19. Organisation for Economic Co-operation and Development. Oecd guidelines on the protection of privacy and transborder flows of personal data of 1980. Available at http://www.oecd.org/.
20. C. S. Powers. Privacy promises, access control, and privacy management. In *ISEC '02: Proceedings of the Third International Symposium on Electronic Commerce*, page 13, Washington, DC, USA, 2002. IEEE Computer Society.
21. P. Z. Revesz. Constraint databases: A survey. In *Selected Papers from a Workshop on Semantics in Databases*, pages 209–246, London, UK, 1998. Springer-Verlag.
22. P. Z. Revesz. Safe datalog queries with linear constraints. In *CP '98: Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming*, pages 355–369, London, UK, 1998. Springer-Verlag.
23. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.
24. S. W. Smith and E. H. Spafford. Grand challenges in information security: Process and output. *IEEE Security and Privacy*, pages 69–71, Jan 2004.

25. TRUSTe.org. An independent, nonprofit enabling trust based on privacy for personal information on the internet. Available at http://www.truste.org/.
26. M. C. Tschantz and S. Krishnamurthi. Towards reasonability properties for access-control policy languages with extended xacml analysis. Tech Report CS-06-04, CS, Brown University. Available at http://www.cs.brown.edu/publications/techreports/reports/CS-06-04.html.
27. United State Department of Health. Health insurance portability and accountability act of 1996. Available at http://www.hhs.gov/ocr/hipaa/.
28. U.S. Senate Committee on Banking, Housing, and Urban Affairs. Information regarding the gramm-leach-bliley act of 1999. Available at http://banking.senate.gov/conf/.
29. W3C. Platform for privacy preferences (p3p) project. Available at http://www.w3.org/P3P.