A Secure, Constraint-Aware Role-Based Access Control Interoperation Framework

Nathalie Baracaldo, Amirreza Masoumzadeh, and James Joshi
School of Information Sciences
University of Pittsburgh
Email: [nab62@, amirreza@sis., jjoshi@sis.] pitt.edu

***Abstract*—With the growing needs for and the benefits of sharing resources and information among different organizations, an interoperation framework that automatically integrates policies to facilitate such cross-domain sharing in a secure way is becoming increasingly important. To avoid security breaches, such policies must enforce the policy constraints of the individual domains. Such constraints may include temporal constraints that limit the times when the users can access the resources, and separation of duty (SoD) constraints. Existing interoperation solutions do not address such cross-domain temporal access control and SoDs requirements. In this paper, we propose a role-based framework to facilitate secure interoperation among multiple domains by ensuring the enforcement of temporal and SoD constraints of individual domains. To support interoperation, we do not modify the internal policies, as most of the current approaches do. We present experimental results to demonstrate our proposed framework is effective and easily realizable.**

I. Introduction

Nowadays, sharing computing and information resources between organizations is becoming more and more important. In fact, several businesses base their competitive advantage on being able to share relevant information and resources with their partners. Unfortunately, security concerns are responsible for the reluctance of some organizations to join such environments [6]. In multi-domain environments, it is necessary that each of the domains involved can control the extent to which it wants to share its resources with other domains. It is important that the internal policy of each organization is not violated during interoperation. If such violations occur, vital information might be lost, modified or disclosed through unauthorized accesses. Therefore, when two or more domains want to interoperate, an interoperation policy should be created. In existing approaches (e.g. [13]), an interoperation policy is usually created manually, which is cumbersome and error-prone. It is desirable to create interoperation policies automatically to avoid security breaches introduced by human mistakes while reducing the amount of time required to enable the sharing activities.

A *secure interoperation policy* follows the principles of *autonomy* and *security* [16]. The *principle of autonomy* states that accesses permitted within a domain before the integration should still be allowed after the integration. On the other hand, the *principle of security* says that accesses forbidden within a domain before the integration should still be forbidden after the integration. To ensure these, the policy constraints of individual domains need to be enforced when sharing resources with external domains. Among the constraints that should be enforced are the separation of duty constraints [5], least privilege constraints [12] and temporal constraints [9].

Role Based Access Control (RBAC) [4] has been shown to encompass discretionary and mandatory access control models and support organization or user-specific requirements [10]. Because of the benefits offered by RBAC, it is considered as a promising approach for expressing domain policies; in this paper, we assume that all the involved domains employ extended RBAC models. To facilitate cross domain accesses in such a multidomain environment, it is first necessary to establish role to role mappings between domains, so that users from a domain can acquire the desired permissions from other domains[3]. Several algorithms have been proposed to solve the role mapping problem [16], [3], [22], [2], [21], [17]. However, these solutions have three key limitations. The first limitation is the limited capability to handle Static SoD (SSoD) constraints. In [18], a role mapping algorithm that respects SSoD of the internal policy was proposed. However, this approach assumes that the policies of both domains are static and are completely known to each other before the interoperation takes place. This assumption is too strong, as the policies of the domains involved can change at any time and some organizations may not trust their partners enough to reveal their policies. The second limitation is related to handling temporal constraints while creating an interoperation policy. The role mapping algorithm proposed in [14] attempts to minimize the number of roles to be mapped to a given request. However, it does not consider the temporal constraints. As a result, mapped roles may not be available at desired times. The third limitation of existing work consists of requiring that the roles selected by the role mapping algorithm only have authorization for the exact set of required permissions. To achieve this objective, some proposed solutions either modify the policy of the domains [18], [14], [20], [19] or grant extra privileges; however, neither of these two solutions are adequate, as the former increases the policy administration difficulty and the latter violates the principle of least privilege. Furthermore, none address these three issues together.

In this paper, we propose a role based framework that addresses aforementioned limitations of existing approaches. The key contributions of this paper are:

- We propose a secure interoperation framework and formally show that it enforces temporal, SSoD and DSoD constraints of individual domains involved.
- We introduce temporal coverage as the objective function of role mapping to maximize interoperation based on temporal requirements.
- We use the notion of *filter roles* in the interoperation

policy to ensure the enforcement of the least privilege principle, and support secure interoperation even when the internal policy evolves.

The paper is organized as follows. In Section II, we present an overview of GTRBAC model, hybrid hierarchy and SoD constraints. In Section III, we present the requirements of the system. In Section IV, we present the proposed secure interoperation framework. We show that the framework complies with the requirements in Section V. In Section VI, we present the proposed role mapping algorithm and implementation results. In Section VII, we discuss related work and in Section VIII, we conclude the paper and discuss future directions.

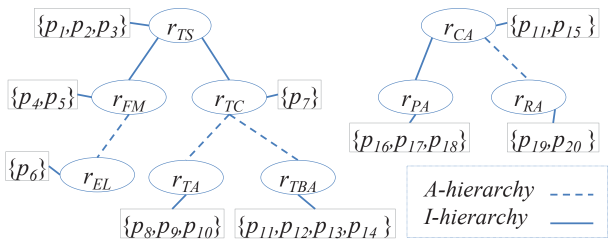
II. Background

Temporal Constraints Generalized Temporal Role Based Access Control (GTRBAC) [9] is an extension of RBAC [4] with temporal constraints that restrict the periods of time during which a user can acquire certain permissions. It allows several types of temporal constraints on different components of the model such as roles, role-permission assignment, etc. In this paper, we only consider temporal constraints for *enabling* and *disabling* roles. In GTRBAC, a role can be activated by an authorized user only if it is *enabled*. A *disabled* role cannot be activated. To express temporal constraints, GTRBAC uses periodic time expressions, which are of form $\langle[begin, end], P\rangle$, where P is a periodic expression denoting an infinite set of periodic time instants, and $[begin, end]$ is a time interval denoting the lower and upper bounds for the instants in P. We refer interested readers to [9] for further details. We define $overlapRatio(tc_1, tc_2)$ as the ratio of the time instants that the interval tc_1 has in common with tc_2 to the interval tc_1.

GTRBAC also introduces the concept of *hybrid hierarchy* [9]. Roles r_1 and r_2 can be hierarchically related in one of the following ways. (1) **I-senior** ($r_1 \geq_I r_2$) where r_1 inherits the permissions of r_2. (2) **A-senior** ($r_1 \geq_A r_2$) where users assigned to r_1 can activate r_2; however, r_1 does not inherit r_2's permissions. (3) **IA-senior** ($r_1 \geq_{IA} r_2$), in this case r_1 is I-senior and A-senior of r_2.

In order to deal with the semantics of temporal constraints in presence of hybrid hierarchy, Joshi *et al.* propose two variations of hierarchical relations: *weak* and *strong* [8]. A *weak* hierarchical relation allows a senior role to activate a junior role or acquire its permissions even when the junior role is not enabled. In contrast, a *strong* hierarchical relation requires that both the senior and the junior roles be enabled for the inheritance semantics to be valid. In this paper, we use the function $P_{au}(r, t)$ to denote the permissions that can be acquired through a role r, including permissions inherited through hierarchical relations.

Separation of duty constraints Separation of duty constraints (SoD) are used to prevent fraudulent activities within an organization by preventing a unique user from playing two or more conflicting roles. There are two types of SoD constraints. The Static SoD (SSoD) and the Dynamic SoD (DSoD). SSoD restricts the authorization of users to conflicting roles [1], [5], [12], [11]. An SoD constraint is denoted by (RS, k) where $RS = \{r_1, r_2, r_3, ..., r_n\}$ is a set of roles in conflict and $2 \leq k \leq n$. We use $ssod(RS, k)$ to express SSoD constraints, which states that a user can be *authorized* to at most $k-1$ roles in RS. Similarly, a DSoD constraint $dsod(RS, k)$ states that a user can *activate* at most $k-1$ roles in RS.



Authorized Permissions and temporal constraints:
$P_{au}(r_{TS})$= $\{p_1, p_2, p_3, p_4, p_5, p_7\}$, $P_{au}(r_{FM})$=$\{p_4, p_5\}$,
$P_{au}(r_{EL})$= $\{p_6\}$, $P_{au}(r_{TC})$= $\{p_7\}$, $P_{au}(r_{RA})$=$\{p_{19}, p_{20}\}$,
$P_{au}(r_{CA})$=$\{p_{11}, p_{15}, p_{16}, p_{17}, p_{18}\}$, $P_{au}(r_{PA})$=$\{p_{16}, p_{17}, p_{18}\}$,
$P_{au}(r_{TA})$ = $\{p_8, p_9, p_{10}\}$, $TC(r_{TA})$=*Mondays to Fridays, 7am to 7pm*
$P_{au}(r_{TBA})$= $\{p_{11}, p_{12}, p_{13}, p_{14}\}$, $TC(r_{TBA})$= *Mondays to Thursdays.*
SoD Constraints: $dsod(\{r_{EL}, r_{TA}, r_{TBA}\}, 3)$, $ssod(\{r_{TS}, r_{CA}\}, 2)$

Fig. 1: TO's internal policy

III. Requirements for Secure Interoperation

An *interoperation policy* indicates which roles in an external domain will be able to acquire which privileges within a local domain. We propose a framework whose main goal is to establish an interoperation policy that respects the policies of individual domains. Here, by *external* and *internal domains* we mean the domain that issues access requests and the domain that grants/denies these requests, respectively. We assume that the domains involved use RBAC policies *extended* with SoDs, temporal constraints and hybrid hierarchy. We illustrate the requirements through the following example.

Example 1: Consider that the Treasurer Office (TO) wants to interoperate with the County Clerk Office (CCO) to increase the effectiveness of detecting tax evasions and have up-to-date and accurate information [16]. Part of TO's policy is shown in Figure 1, where each role has been directly assigned to it the permissions shown in the rectangles and the permissions that each role can acquire are shown underneath the policy. TO's policy has the following roles: *Chief auditor* (r_{CA}), *Process auditor* (r_{PA}), *Refund auditor* (r_{RA}), *Tax collector* (r_{TC}), *Fraud manager* (r_{FM}), *Tax assessor* (r_{TA}), *Tax bill approver* (r_{TBA}) and *Treasurer* (r_{TS}). TO's policy contains important information on how to manage SoD conflicts. For instance, it contains the SSoD constraint $ssod(\{r_{TS}, r_{CA}\}, 2)$ that specifies that r_{TS} and the r_{CA} cannot be assigned to the same user; i.e., the person in charge of performing the operations should not be her own auditor. In addition, DSoD constraints specify how to follow the principle of least privilege. The constraint $dsod(\{r_{EL}, r_{TA}, r_{TBA}\}, 3)$ specifies that the same user cannot activate 2 of those roles simultaneously. Finally, it also contains information on the times during which the roles are enabled; e.g., r_{TBA} can only be activated during Mondays to Thursdays and role r_{TA} can only be activated during office hours (7am to 7pm) Monday to Friday. These SSoD, DSoD and temporal constraints within a local domain also need to be enforced when users from external domains are given access through a secure interoperation policy.

In order to comply with the *principle of security*, all the constraints of the internal domain's policy should be enforced when it interoperates with external domains. Hence, it is important to create an interoperation policy that respects these constraints in local domains. This requires looking for the set of roles in the internal policy that can provide the permissions that need to be granted to users assigned to a role of the external domain without violating any constraint. When an internal domain has a large policy (in terms of the number of roles and constraints), performing this process manually is very tedious and error-prone, and hence may lead to violations of the internal policy. Our framework provides the means to automate this process.

To create an interoperation policy, the internal domain's *security administrator* issues an *interoperation query* that contains a set of desired permissions, and a periodic expression during which those permissions should be available for a particular external role. Given a set of interoperation queries, the framework should construct an interoperation policy that fulfills the following requirements:

1) It should respect the SSoD, DSoD and temporal constraints of the internal domain.
2) The time during which interoperation can happen should be maximized.
3) It should be possible to establish additional temporal restrictions on the accesses to the internal resources.
4) It should provide only the required permissions.
5) Individual domains should not be required to disclose their policies to create the interoperation policy. We assume the external domain does not trust the internal domain enough to reveal its policy and vice versa.
6) The policy of the internal domain should not be modif in order to interoperate, but it may evolve.

IV. Policy Interoperation Framework

Our approach consists of creating an intermediate layer of roles that forms the interoperation policy, which governs how a particular external domain can access the resources of the internal domain; one interoperation policy is created for each external domain and managed by the internal domain.

A. Preliminaries and Notations

We use the following notations. A GTRBAC policy is an 8-tuple $\langle U, R, P, UA, PA, RH, TC, SSoD, DSoD\rangle$ where in addition to standard GTRBAC components, TC is a function assigning enabling time constraint to roles. We use subscripts to refer to these sets in a specific GTRBAC policy, e.g., $SSoD_{\mathcal{P}}$ refers to the set of SSoDs in policy $\mathcal{P}$.

We use $\mathcal{P}$ and $\mathcal{E}$ to denote the policies of internal and external domain, respectively. The interoperation between the two domains is a triple $\langle Q, \mathcal{P}, \mathcal{E}\rangle$ where Q is a set of interoperation queries. An *interoperation query* is a triple $\langle r, PS, \tau\rangle$, indicating role $r \in R_{\mathcal{E}}$ (in the external domain) requires access to the permission set $PS \subseteq P_{\mathcal{P}}$, during a periodic time τ.

The interoperation policy is constructed using a special type of roles called *filter roles* [7]. A *filter role* is a role that has associated with it an *upper bound set (UBS)*, which is the maximum set of permissions that can be acquired through it. We formally define it as follows.

Candidate Solution	$P_{au}(r \in R_{sel})$	**Enabled time**	**CR**
1	$P_{au}(r_1) = \{p_1, p_2, p_3, p_4, p_5\}$	3pm-8pm	0.25
2	$P_{au}(r_2) = \{p_1, p_6, p_7\}$	9am-4pm	0.625
	$P_{au}(r_3) = \{p_2, p_3, p_4, p_7\}$	8am-2pm	
3	$P_{au}(r_4) = \{p_1, p_4\}$	8am-10pm, 2pm-6pm	0.5
	$P_{au}(r_5) = \{p_1, p_3, p_5, p_6\}$	11am-5pm	
	$P_{au}(r_6) = \{p_2\}$	9am-5pm	

TABLE I: Sample Coverage for $\{p_1, p_2, p_3, p_4\}$ and $\tau = \{9am - 5pm, everyday\}$

Definition 1: Let r_f be a filter role and $UBS\ (r_f)$ indicate its upper bound permission set. The authorized permissions for r_f at time t is

$$P_{au}(r_f, t) = \bigcup_{r \in I-juniors(r_f)} P_{au}(r, t)\ \cap\ UBS(r_f)$$

We use dot notation to refer to the elements of a tuple, e.g., $c.RS$ refers to the role set in the constraint $c = dsod(RS, k)$. Whenever a set is used in a formula where a single element should be normally used, we mean the union of the results of applying each member of the set in the formula. For instance, $P_{au}(RS)$ refers to the union of sets of permissions authorized for individual roles in set RS and $DSoD_{\mathcal{P}}.RS$ refers to the union of sets of roles involved in a DSoD constraint in $\mathcal{P}$.

B. The Role Selection Optimization Problem

To create a suitable interoperation policy, it is necessary to find the set of roles that provides the desired permissions and t can also be activated during the requested periods of time. We define the notion of *coverage ratio* to indicate how much of the requested interval is covered by a particular solution $R_{sel} \subseteq R_{\mathcal{P}}$.

Definition 2 (Coverage Ratio (CR)): Given a candidate solution R_{sel} for the interoperation query $\langle r, PS, \tau\rangle$, *CR($R_{sel}, PS, \tau$)* is defined as:

$$overlapRatio(\tau, \bigcap_{p \in PS} \bigcup_{r_i \in R_{sel}} \{t | p \in P_{au}(r_i, t)\})$$

The objective of our role selection process is to choose a solution that maximizes coverage ratio. In addition, it needs to consider the restrictions imposed by SoD constraints. We define the role selection problem as follows.

Definition 3: The *Role Selection Problem* (RSP) for a query $q = \langle r, PS, \tau\rangle$ is to find a solution, $S(q)$, for the following optimization problem:

$$\underset{R_{sel} \in R_{\mathcal{P}}}{\textbf{Max}}\ CR(R_{sel}, PS, \tau)$$

$$\begin{aligned} \textbf{s.t.}\ &\forall\ dsod(RS_i, k_i) \in DSoD_{\mathcal{P}}\ : |R_{sel} \cap RS_i| < k_i \\ &\forall ssod(RS_i, k_i) \in SSoD_{\mathcal{P}} : |R_{sel} \cap RS_i| < k_i \\ &P_{au}(R_{sel}) \supseteq PS \end{aligned}$$

If $S(q)$ has a coverage of zero for a given query q, it means that no set of roles in $R_{\mathcal{P}}$ can satisfy the query. i.e., the interoperation request is denied.

Example 2: Suppose that Table I lists feasible solutions for the role selection problem for query $\langle r, \{p_1, p_2, p_3, p_4\}, 9am-5pm, everyday\rangle$ and their calculated CR. Solution 2 provides

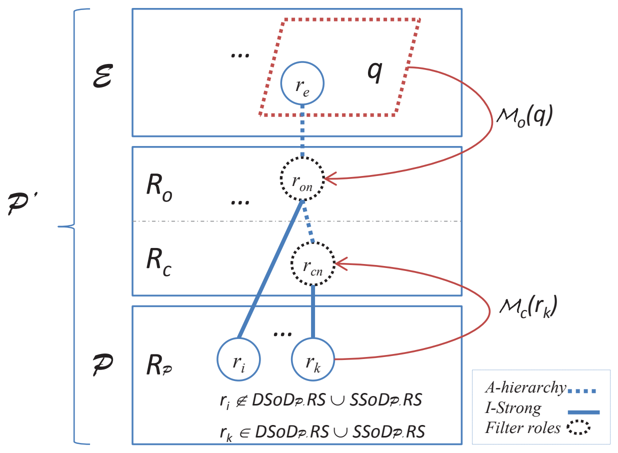


Fig. 2: Overview of *Interoperation-Augmented Policy* (IAP)

the best coverage, hence $S(q) = \{r_2, r_3\}$. Without considering temporal constraints, solution 1 would be selected, as it satisfies the request with a minimum number of roles. However, this solution is not suitable based on temporal requirements as it only covers 25% of the requested interval.

C. *Interoperation-Augmented Policy (IAP)*

In order to allow secure interopration $\langle Q, \mathcal{P}, \mathcal{E}\rangle$, we augment the internal policy with an interoperation policy. We call the result an *Interoperation-Augmented Policy (IAP)*. This process is abstractly presented in Figure 2. From top to bottom, the three layers represent external, interoperation, and internal policy, respectively. The interoperation layer consists of filter roles that are conceptually divided into two categories: *interoperation roles* (R_o) and *constrained roles* (R_c). Roles in R_o provide access to the external users, as they are made A-strong junior of the external query roles. Every query is mapped uniquely to one of these roles using $\mathcal{M}_o$. Roles in R_c ensure the enforcement of SoD constraints of the internal policy for external users. There exists a role in R_c corresponding to every role in $S(q)$ that is part of an internal SoD. The correspondence is provided by mapping function $\mathcal{M}_c$. Roles in R_o are A-strong-seniors of the corresponding roles in R_c. Finally, all hierarchical relations between roles in the interoperation layer and corresponding selected internal roles are of type I-strong. IAP is formally defined as follows.

Definition 4: For interoperation $\langle Q, \mathcal{P}, \mathcal{E}\rangle$, the *interoperation-augmented policy* $\mathcal{P}'$ augments the internal policy as follows.

Roles $(R'_\mathcal{P})$: $R'_\mathcal{P} = R_\mathcal{P} \cup R_Q \cup R_o \cup R_c$ where

- $R_Q = Q.r$ (i.e., set of all external roles in the queries)
- $R_o = \{\mathcal{M}_o(q) | \forall q \in Q\}$, where $\mathcal{M}_o$ is a one-to-one mapping function from a query role $q.r$ to its corresponding interoperation role, and $\forall r \in R_o, q = \mathcal{M}_o^{-1}(r)[UBS(r) = q.PS \wedge TC(r) = q.\tau]$.
- $R_c = \{\mathcal{M}_c(r) | \exists (c = dsod(RS, k) \in DSoD_\mathcal{P} \vee c = ssod(RS, k) \in SSoD_\mathcal{P}) \wedge r \in RS \wedge r \in S(Q)\}$, where $\mathcal{M}_c$ is a one-to-one mapping function from an internal role to its corresponding constrained filter role, and $\forall q \in Q\ \forall r \in R_c\ [r \in \mathcal{M}_c(S(q)) \rightarrow UBS(r) = q.PS \wedge TC(r) = q.\tau]$.

Users $(U_{\mathcal{P}'})$: an imaginary user u_e in the external domain is assumed to have access to all the external policy:

- $U_{\mathcal{P}'} = U_\mathcal{P} \cup \{u_e\}$.
- $UA_{\mathcal{P}'} = UA_\mathcal{P} \cup \{\langle u_e, r\rangle | r \in R_\mathcal{E}\}$.

Role Hierarchy $(RH_{\mathcal{P}'})$: Is the union of $RH_\mathcal{P}$ and the set containing the following hierarchical relations:

- Query roles A-senior to interoperation roles: $\forall q \in Q\ [q.r \geq_{A-strong} \mathcal{M}_o(q)]$
- Interoperation roles A-senior to constrained roles: $\forall q \in Q\ \forall r_c \in R_c\ [r_c \in \mathcal{M}_c(S(q)) \rightarrow \mathcal{M}_o(q) \geq_{A-strong} r_c]$
- Constrained roles I-senior to selected roles: $\forall r_c \in R_c\ [r_c \geq_{I-strong} \mathcal{M}_c^{-1}(r_c)]$
- Interoperation roles I-senior to selected roles: $\forall q \in Q\ \forall r_s \in S(q) \setminus \mathcal{M}_c^{-1}(R_c)\ [\mathcal{M}_o(q) \geq_{I-strong} r_s]$

DSoDs $(DSoD_{\mathcal{P}'})$: Is the union of $DSoD_\mathcal{P}$ and the following sets:

- Equivalent DSoDs on the constrained roles to enforce $DSoD_\mathcal{P}$: $\{dsod(\mathcal{M}_c(RS \cap S(Q)), k) | \exists c = dsod(RS, k) \in DSoD_\mathcal{P} \wedge RS \subseteq S(Q)\}$
- Equivalent DSoDs on the constrained roles to enforce $SSoD_\mathcal{P}$: $\{dsod(\mathcal{M}_c(RS \cap S(Q)), k) | \exists c = ssod(RS, k) \in SSoD_\mathcal{P} \wedge RS \subseteq S(Q)\}$

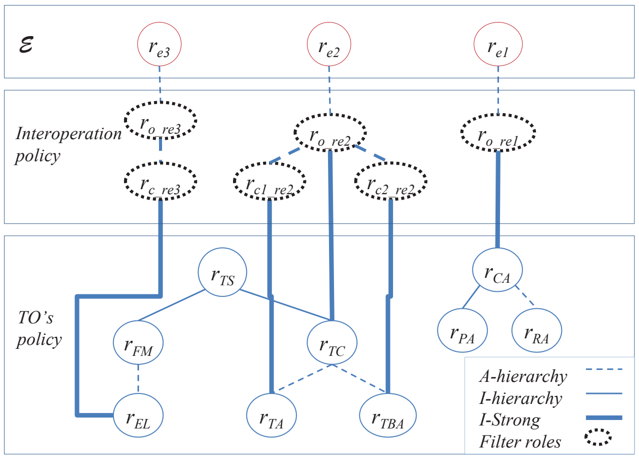
In the above definition, we consider a unique external user u_e who is authorized for all the external roles in Q. Due to the requirement of non-disclosure of the external's domain policy, we cannot assume any knowledge of authorized roles for external users. Therefore, we consider the worst case scenario, in which there exists an external user that is authorized for all the accesses that can take place in the interoperation.

Also, note that roles in R_Q do not have any permissions assigned to them, although in the external domain this might be the case. We do not consider those assignments, because the focus of our framework is to ensure the safety of the internal domain.

Example 3: Consider Example 1; suppose an interoperation query is given by the tuple $q = \langle r_{e1} \in R_Q, \{p_{11}, p_{15}, p_{16}\}, AllFridays\rangle$. First, the roles that provide the requested permissions are selected without violating any of the internal constraints (Section V explains in detail how this is done). For query q, r_{CA} is selected, as it provides the requested permissions. After this role has been selected, a filter role r_{o_re1} is created in the interoperation policy to satisfy the request. The filter roles is initialized so that $UBS(r_{o\_re1}) = \{p_{11}, p_{15}, p_{16}\}$ and $TC(r_{o\_re1}) = AllFridays$. Then, the relations $r_{e1} \geq_{A-strong} r_{o\_re1} \geq_{I-strong} r_{CA}$ are created as shown in Figure 3. Here, $P_{au}(r_{CA})$=$\{p_{11}, p_{15}, p_{16}, p_{17}, p_{18}\}$; note that role r_{CA} has two additional permissions than those strictly required for the interoperation: $\{p_{17}, p_{18}\}$. Because of Definition 1, users authorized for r_{e1} can only acquire the exact set of required permissions respecting the principle of least privilege. With respect to the temporal constraint, the external domain can only acquire permissions during Fridays, as the temporal constraint on r_{o_re1} and the internal domain does not impose any additional temporal constraint.

V. Conforming with the Requirements

In this section, we explain and prove that the proposed IAP provides only the requested permissions, respects DSoDs, SSoD and temporal constraints of the internal policy.



Internal policy

Authorized Permissions and temporal constraints:
$P_{au}(r_{TS})$= $\{p_1, p_2, p_3, p_4, p_5, p_7\}$, $P_{au}(r_{FM})$=$\{p_4, p_5\}$,
$P_{au}(r_{EL})$= $\{p_6\}$, $P_{au}(r_{TC})$= $\{p_7\}$, $P_{au}(r_{PA})$=$\{p_{16}, p_{17}, p_{18}\}$
$P_{au}(r_{CA})$=$\{p_{11}, p_{15}, p_{16}, p_{17}, p_{18}\}$, $P_{au}(r_{RA})$=$\{p_{19}, p_{20}\}$
$P_{au}(r_{TA})$ = $\{p_8, p_9, p_{10}\}$, $TC(r_{TA})$=*Mondays to Fridays, 7am to 7pm,*
$P_{au}(r_{TBA})$= $\{p_{11}, p_{12}, p_{13}, p_{14}\}$, $TC(r_{TBA})$= *Mondays to Thursdays.*
SoD Constraints: $dsod\{r_{EL}, r_{TA}, r_{TBA}\}, 3)$ $ssod(\{r_{TS}, r_{CA}\}, 2)$

Components of Interoperation Policy

$UBS(r_{o\_re1}) = \{p_{11}, p_{15}, p_{16}\}$, $TC(r_{o\_re1}) = AllFridays$
$UBS(r_{o\_re2}) = \{p_7, p_8, p_9, p_{10}, p_{12}, p_{13}, p_{14}\}$
$UBS(r_{c1\_re2}) = \{p_7, p_8, p_9, p_{10}, p_{12}, p_{13}, p_{14}\}$
$UBS(r_{c2\_re2}) = \{p_7, p_8, p_9, p_{10}, p_{12}, p_{13}, p_{14}\}$
$UBS(r_{c\_re3}) = UBS(r_{o\_re3}) = \{p_6\}$
$TC(r_{o\_re2}) = TC(r_{c1\_re2}) = TC(r_{c2\_re2}) = EveryDay$,
$TC(r_{o\_re3}) = TC(r_{c\_re3}) = AllFridays$
SoD Constraints: $dsod(\{r_{c1\_re2}, r_{c2\_re2}, r_{c\_re3}\}, 3)$

Fig. 3: Example of an IAP system.

A. *Providing Only the Requested Permissions*

The permissions an external user u authorized for $r \in R_Q$ can acquire in the internal domain are bounded by the set of queries created by the system's administrator for role r_e (assuming u is only authorized for r_e). Hence, u can acquire permissions in the internal domain by activating the roles in the interoperation policy that are juniors of r_e. The following theorem shows IAP complies with these requirements.

Theorem 1: Let Q_e be a set of queries with the same external role r_e, for permissions in internal policy $\mathcal{P}$. A user u, in $\mathcal{P}$'s IAP, can only acquire the permissions in $Q_e.PS$ because of being authorized for r_e.

Proof: Being authorized for r_e allows u to activate r_e and its A-juniors to acquire permissions. By activating r_e, u does not acquire any permission because, as per Definition 4, r_e does not have assigned any permissions. The A-juniors of r_e, roles in R_o and R_c, are filter roles that have their UBS set to $q.PS$, where $q \in Q_e$. Therefore, they can acquire at most $Q_e.PS$. ■

B. *Respecting Internal Enabling Constraints*

One of the requirements of the framework is to respect the temporal constraints of the internal policy. In what follows, we show that IAP enforces temporal constraints of the internal policy and is able to enforce additional constraints during interoperation with an external domain.

Theorem 2: For an IAP policy and a corresponding query $q = \langle r, PS, \tau\rangle$, user u_e can only acquire the permissions in $q.PS$ during time τ without violating any temporal constraint of the internal domain.

Proof: Based on hierarchical relations in Definition 4, u_e can only acquire permissions by activating A-juniors of r_e. All such roles that are added to the IAP policy for the query q are assigned the time constraint τ. Therefore, u_e can only activate these roles during τ. Since these roles are linked to the internal policy roles using *I-strong*-hierarchy, they do not inherit permissions unless the internal roles are enabled. ■

C. *Respecting Dynamic Separation of Duty Constraints*

In this section, we first provide some definitions and then formally prove that IAP respects DSoDs.

Definition 5: A user *can acquire* a role if the user can activate it or one of its I-seniors.

Definition 6: A GTRBAC system *enforces* constraint $c = dsod(RS, k)$ if and only if no user can simultaneously activate k or more roles in RS.

Definition 7: A GTRBAC system *strictly enforces* constraint $c = dsod(RS, k)$ if and only if no user can simultaneously acquire k or more roles in RS.

Strict enforcement of DSoD constraints is a desired property in RBAC systems, and in fact, one of the main reasons that led to introduction of hybrid hierarchy [15]. We assume that in the internal policy, the roles involved in a DSoD constraint may only have A-senior roles, and no I-senior roles. This guarantees that when a GTRBAC system enforces a DSoD constraint, it is enforced strictly. We call such a policy *DSoD-well-formed*.

Definition 8: A policy $\mathcal{P}$ is *DSoD-well-formed* if and only if $\forall r \in RS, c = dsod(RS, k) \in DSoD_{\mathcal{P}}[\nexists r' \in R_{\mathcal{P}}, r' \geq_I r]$.

Theorem 3: Let $\mathcal{P}$ and $\mathcal{P}'$ be a DSoD-well-formed GTRBAC system and its IAP, respectively. $\mathcal{P}'$ strictly enforces constraints in $DSoD_{\mathcal{P}}$.

Proof: Assume $c = dsod(RS, k) \in DSoD_{\mathcal{P}}$ is not strictly enforced in $\mathcal{P}'$, i.e., a user can acquire $RK \subseteq RS$ with k roles in system $\mathcal{P}'$. Note that $RS \subseteq R_{\mathcal{P}}$. Since the user assignments and their respective role structure for the users in $U_{\mathcal{P}}$ is not different in $\mathcal{P}$ and $\mathcal{P}'$, the user that violates strict enforcement has to be u_e. Among the roles in $R_{\mathcal{P}}$, u_e can only acquire $S(Q)$, i.e., the set of selected roles for interoperation queries, and the roles I-junior to them. Since $\mathcal{P}$ is DSoD-well-formed, the roles I-junior to the roles in $S(Q)$ must not be involved in a DSoD. Thus, we have $RK \subseteq S(Q)$. Since roles in RK are involved in a DSoD, based on Definition 4, we have $RK' = \mathcal{M}_c(RK) \subseteq R_c$. u_e needs to activate all roles in RK' in order to acquire their corresponding roles in RK. According to Definition 4, there exists a constraint $c' = dsod(RS', k)$, which should be enforced in $\mathcal{P}'$. Since $RK \subseteq RS$, we have $RK' \subseteq RS'$. Hence, u_e needs to activate k roles in RS' which contradicts the enforcement of c'. Consequently, our assumption about non-strict enforcement of c is false. ■

Example 4: Let $q_1 = \langle\{p_6, p_8, p_9, p_{10}, p_{12}, p_{13}, p_{14}\}, r_e \in R_e, \tau = EveryDay >\rangle$ be an interoperation query. Here, roles r_{EL}, r_{TA} and r_{TBA} should be selected, but that would violate the DSoD constraint of TO's internal policy: $dsod(\{r_{EL}, r_{TA}, r_{TBA}\}, 3)$ (i.e., selected roles cannot be activated simultaneously). Hence, this request is *denied*, because

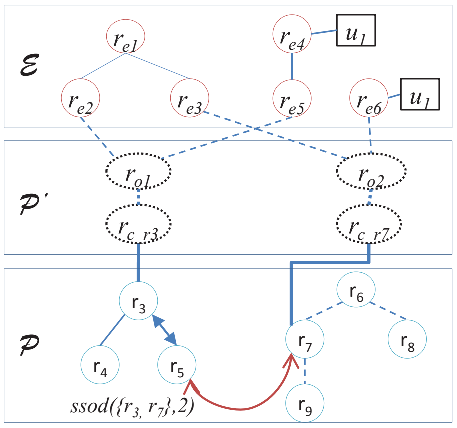


Fig. 4: An example of indirect violation of SSoD

creating a filter role that inherits from r_{EL}, r_{TA} and r_{TBA} would result in a $\mathcal{P}'$ that is not DSoD-well-formed.

Example 5: Let $q_1 \;=<\; \{p_7, p_8, p_9, p_{10}, p_{12}, p_{13}, p_{14}\}$, $r_{e2} \in R_e$, $EveryDay >$ and $q_2 \;=< \{p_6\}$, $r_{e3} \in R_e$, $AllFridays >$. Query q_1 can be granted by selecting roles r_{TA}, r_{TBA} and r_{TC}. Note that role r_{TBA} can be activated only on Monday to Thursday. Although the request was issued for *every day*, because of the *I-strong* semantics, the activation constraint of r_{TBA} is respected and the external domain cannot inherit permission from r_{TBA} on Fridays, Saturdays and Sundays. To satisfy q_1, filter roles r_{o_re2}, r_{c1_re2} and r_{c2_re2} are created as shown in Figure 3. According to Definition 4, $r_{o\_re2} \in R_o$ and r_{c1_re2}, $r_{c2\_re2} \in R_c$ because the latter roles inherit from a role in DSoD. For query q_2, role r_{EL} should be selected. Two interoperation roles $r_{o\_re3} \in R_o$ and $r_{o\_re3} \in R_c$ are created to satisfy this request as shown in Figure 3. Since r_{c_re3} is inheriting from r_{EL}, which is in a DSoD constraint, a constraint $dsod(\{r_{c1\_re2}, r_{c2\_re2}, r_{c\_re3}\}, 3)$ is created. Hence, even if an external user is authorized for roles r_{e2} and r_{e3}, he cannot activate them simultaneously, thus respecting the DSoD internal policy. Finally, note that filter roles created for q_1 and q_2 are initialized as follows $UBS(r_{o\_re2}) = UBS(r_{c\_re2}) = UBS(r_{c\_re2}) = \{p_7, p_8, p_9, p_{10}, p_{12}, p_{13}, p_{14}\}$ and $UBS(r_{o\_re3}) = UBS(r_{c\_re3}) = \{p_6\}$.

D. Respecting Static Separation of Duty Constraints

In this section, we show how the framework enforces SSoD constraints. According to the requirements presented in Section III, the policy of the external domain is not known. As a consequence, it is not possible to identify SSoD violations caused by the hierarchical structure or user assignments of the policy of the external domain. Furthermore, even if the policy of the external domain was known, in order to have a continuous enforcement of the internal SSoD constraints, the external domain should maintain its policy as it was when $\mathcal{P}'$ was constructed. Therefore, we propose to relax the SSoD to DSoD constraints to ensure that the permissions and roles in conflict are never acquired simultaneously by any user of the external domain. We capture this relaxation in Definition 4.

Example 6: Consider the policy presented in Figure 4 with $ssod(\{r_3, r_7\}, 2)$. The hierarchy paths from r_{e1} to r_3 and from r_{e1} to r_7 represent a SSoD violation. This means that any user authorized for r_{e1} is authorized for r_3 and r_7. Moreover, user u_1 assigned to roles r_{e4} and r_{e6} violates the above-mentioned SSoD, by being authorized for r_3 and r_7. To mitigate this lack of control, we add $dsod(\{r_{c3}, r_{c7}\}, 2)$ to $\mathcal{P}'$, which ensures the relaxed enforcement of $ssod(\{r_3, r_7\}, 2)$.

Now, we formally prove enforcement of SSoD constraints in our framework, based on the following definitions. We assume that there are no roles in the internal policy that inherit from the roles involved in SSoD constraints. We call such a policy *SSoD-well-formed* .

Definition 9: A GTRBAC system *satisfies* constraint $c = ssod(RS, k)$ if and only if no user is authorized to k or more roles in RS.

Definition 10: A policy $\mathcal{P}$ is *SSoD-well-formed* if and only if $\forall r \in RS, c = ssod(RS, k) \in SSoD_{\mathcal{P}} [\nexists r' \in R_{\mathcal{P}}, r' \geq_I r]$.

Theorem 4: Let $\mathcal{P}$ satisfy $SSoD_{\mathcal{P}}$ and be SSoD-well-formed and DSoD-well-formed. Also, let $\mathcal{P}'$ be its IAP. $\mathcal{P}'$ strictly enforces constraints in $SSoD_{\mathcal{P}}$ as DSoDs.

Proof: As per Definition 4, there exists a constrained role and DSoD constraint for each selected role that is in a SSoD constraint. Therefore, constraints in $SSoD_{\mathcal{P}}$ are regarded as DSoDs and strictly enforced by $\mathcal{P}'$ as proved in Theorem 3. ■

VI. Algorithms and Implementation Results

The interoperation policy is created in two phases, the *role selection* phase and the *interoperation policy construction* phase where the filter roles and the hierarchy of the interoperation policy are constructed. Algorithm 1 is used to process the queries, and Algorithm 2 presents the role mapping details. The role mapping works using a backtracking algorithm that greedily looks for the solution with the maximum possible coverage (defined in Section IV-B) given a request $\langle P_{req}, \tau, r_{ext}\rangle$. In each iteration the role with maximum average coverage per queried permission is selected and, to respect the SSoD constraints, the search space is reduced by removing the roles that cannot be authorized any longer. For instance, given a constraint $ssod(RS, k)$ if the algorithm has selected $k-1$ roles in RS, the search space is pruned accordingly. Similarly, the algorithm prunes the roles that cannot be activated at the same time due to DSoD constraints. If a solution for a request is found some post-processing is performed. Because the policy of the external domain is dynamic and unknown, we relax SSoD to DSoD as explained in Section V-D. The time complexity in the worst case is exponential to the number of roles in the internal policy, but in practice is less due to the pruning strategy used.

A. Implementation Results

We have implemented a prototype system to test the performance of the proposed algorithm. We generated 30 random policies and averaged the time required to process a query. Each role was randomly assigned a periodic time constraint. The ratios of role to SSoD and DSoD were set to 0.1, and of role to temporal constraints to 0.4. The ratio of roles to authorized permissions was kept 1:24. A randomly generated request was created. The number of requested permissions was kept to five, the number of hierarchical levels to seven and the

Algorithm 1 createInteroperationPolicy(r, PS, τ)

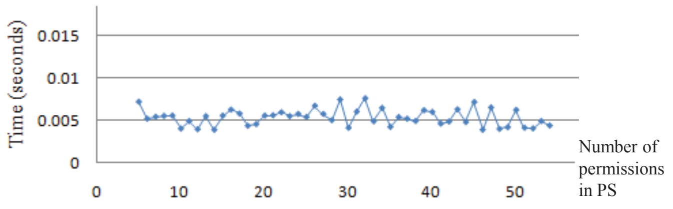
```
$R_{avail} \leftarrow R_{\mathcal{P}} \setminus juniors(r \in SSoD_{\mathcal{P}})$ {Candidate roles}
$R_{sel} \leftarrow \emptyset$ {Selected roles so far}
$P_{rem} \leftarrow P_i$ {Set of permissions that haven't been found}
$P_{part} \leftarrow \emptyset$ {Permissions partially covered}
$R_b \leftarrow \emptyset$ {Global variable to store the best found solution}
for all $p \in PS$ do
    add to $UT$ the pair $[p, \tau]$ {Initialize uncovered time per permission}
$selectRolesMaxCoverage(P_{rem}, R_{avail}, R_{sel}, P_{part}, UT)$ {See Algorithm 2}
if $R_b \neq \emptyset$ then
    Follow Definition 4
else
    return {Could not find a solution for the query}
```

Algorithm 2 selectRolesMaxCoverage($P_{rem}, R_{avail}, R_{sel}, P_{part}, UT$)

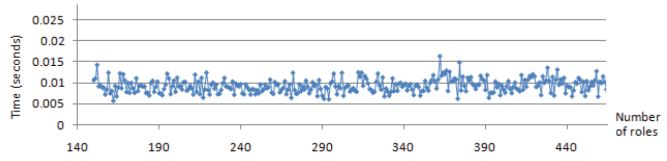
```
if $P_{rem} = \emptyset$ then
    if $R_b = \emptyset$ then
        $R_b \leftarrow R_{sel}$
    else
        if $CR(R_b, PS, \tau) < CR(R_{sel}, PS, \tau)$ then
            $R_b \leftarrow R_{sel}$
        else if $(CR(R_b, PS, \tau) = CR(R_{sel}, PS, \tau)) \wedge | R_b | > | R_{sel} |$ then
            $R_b \leftarrow R_{sel}$
    return {Found candidate solution}
if coverage($R_b$) = 1 then
    return {Objective fulfilled.}
for all $ssod(RS, k) \in SSoD_{\mathcal{P}}$ do
    if $|R_{sel} \cap RS| = (k-1)$ then
        $R_{avail} \leftarrow R_{avail} \setminus [RS \setminus (R_{sel} \cap RS)]$
if simultaneous then
    for all $dsod(RS, t) \in DSoD_{\mathcal{P}}$ do
        if $| R_{sel} \cap RS | = (k-1)$ then
            $R_{avail} \leftarrow R_{avail} \setminus [RS \setminus (R_{sel} \cap RS)]$
for all $r_i \in R_{avail}$ do
    if $P_{rem} \cap P_{au}(r_i) = \emptyset$ then
        $R_{avail} \leftarrow R_{avail} \setminus r_i$
if $R_{avail} = \emptyset$ then
    return
$r_{best} \leftarrow roleMaxCoverage(P_{rem}, R_{avail}, UT)$
$R_{avail} \leftarrow R_{avail} \setminus r_{best}$
$newUT \leftarrow UT$ {Create a copy}
updated newUT according to $r_{best}$ activation constraint
$P_{newPart} \leftarrow P_{part} \cup P_{au}(r_{best})$
$P_{newRem} \leftarrow P_{rem}$
for all $p \in P_{rem}$ do
    if $[p, \tau] \in newUT$ has $\tau = 0$ then
        $P_{newRem} \leftarrow P_{newRem} \setminus p$
    else if $\forall r \in R_{avail} : coverage(newUT(p), r) = 0 \wedge (p \in P_{newPart})$ then
        $P_{newRem} \leftarrow P_{newRem} \setminus p$
selectRolesMaxCoverage($P_{newRem}, R_{avail}, (R_{sel} \cup \{r_{best}\}), P_{newPart}, newUTime$)
selectRolesMaxCoverage($P_{rem}, R_{avail}, R_{sel}, P_{part}, UT$)
```

number of roles to 400. We used an Intel Core 2 Duo with 4GB of memory running Windows 7 to perform the tests.

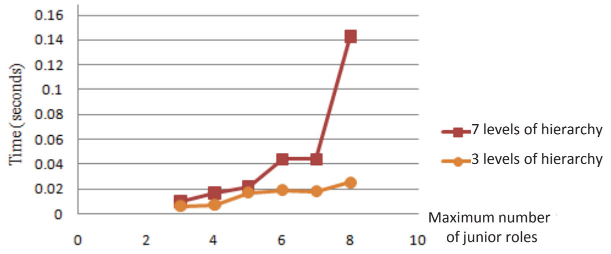
Figure 5a shows the processing time against increasing number of requested permissions, keeping other variables constant. We observe some fluctuations in the time required to process the requests because of the proposed greedy heuristic for processing different requests against different policies. The number of requested permissions does not seem to influence *per se* the time required to find a solution. The effect of augmenting the number of roles in $\mathcal{P}$ while keeping the remaining variables constant is shown in Figure 5b. The results suggest that the time required to find a solution does not increase considerably when the number of roles in the internal policy is augmented and the remaining parameters are maintained unchanged. Finally, we find that the performance of the algorithm depends on the configuration of $\mathcal{P}$, as Figure 5c suggests. The experiment was run increasing the maximum number of junior roles in $\mathcal{P}$ and maintaining the other variables fixed. Each line represents the results for policies of 3 and 7 hierarchy levels. As can be seen, augmenting the number of junior roles increases the time required to find a solution for an interoperation query. This is due to the increase of density of commonly authorized permissions per role, caused by the inheritance of permissions, which in turn increases the size of the search space. The experimental results suggest that the algorithm can find solutions in reasonable time.



(a) Effect of $|PS|$



(b) Effect of the number of roles



(c) Effect of the number of junior roles

Fig. 5: Performance of the system

VII. Related Work

Several algorithms related to the role mapping problem exist, but they do not consider temporal constraints. In [3], simulated annealing and genetic algorithms have been used to find a suboptimal mapping. In [2], an alternative objective function for the role mapping algorithm has been proposed to consider the least privilege principle by penalizing solutions that contain roles with extra permissions. In [3], [22], [2],

[21], various algorithms have been proposed to solve the role mapping problem in presence of hybrid hierarchy - none, however, consider temporal, SSoD and DSoD constraints.

In [14], an approach to consider temporal constraints has been proposed. Their approach may override the internal temporal constraints as the configuration of the internal domain is not considered. Wickramaarachchi *et. al* [19] propose heuristics to solve the role mapping problem considering the DSoD constraints. However, their work does not consider the SSoD or temporal constraints, and their algorithms may provide more permissions than those strictly required for the interoperation. Algorithms proposed in [18] aim to respect several constraints in presence of hybrid hierarchy, yet their approach has some important limitations. The roles of the internal and the external domains are mapped directly. Since they do not use an intermediate layer, additional constraints cannot be established on the interoperation. In addition, to respect SSoD constraints, their algorithm assumes that the policy of the external domain is known and static and may modify their policies during interoperation periods.

Using filter roles allows our proposed approach to provide only the permission needed without modifying the internal policy. To the best of our knowledge, two alternative solutions exist. In [18], [14], [20], the roles of the internal policy are split to a degree in which the selection of a particular role provides the exact set of required permissions. However, splitting the roles affects the organization and the original semantics of the hierarchy making it unmanageable. In the approach proposed in [22], [21], [19] the principle of least privilege is sacrificed when interoperation needs are more important. The solution we propose avoids having to make such a compromise by filtering out the permissions that should not be provided to the external domain, without modifying the policy of the internal domain. In [7], a similar notion of filter role is used, yet it is limited to the semantics of the intra-domain delegation. To the best of our knowledge, this paper is the first to introduce filter roles in the context of multi-domain environments.

In addition to practical and secure support for SoD and temporal constraints, our framework is distinguished from the existing approaches in the following aspects: *1)* The proposed framework respects the principle of least privilege, as it provides the exact set of required permissions, unlike other approaches [22], [21], [19] that provide more permissions than those requested. *2)* The proposed solution maintains the internal policy unmodified, while the existing approaches [18], [14], [20] can potentially modify the internal policy for every interoperation request. *3)* To the best of our knowledge, our work is the first one to propose an algorithm that respects the temporal constraints of the internal domain.

VIII. Conclusions

In this paper, we have proposed a secure role-based framework that allows time-constrained interoperation between domains, and complies with SoD and temporal constraints of the internal domain policy. In addition, we consider a number of other requirements such as maximizing the available interoperation time, assuming lack of knowledge and control over the policy of the external domain, and avoiding changing in the internal policy as the result of interoperation. We also prove formally that our framework meets our security requirements such as conforming with the constraints. As future work, we plan to investigate reuse of the same interoperation policy for multiple interoperations, in order to minimize the enforcement load on the internal domain. We also plan to incorporate additional temporal constraints proposed in GTRBAC.

Acknowledgment

This research has been supported by the US National Science Foundation award IIS-0545912.

References

[1] Ahn, G.J., Sandhu, R.: Role-based authorization constraints specification. ACM Trans. Inf. Syst. Secur. 3, 207–226 (2000)

[2] Chen, L., Crampton, J.: Inter-domain role mapping and least privilege. The 12th ACM SACMAT. pp. 157–162. (2007)

[3] Du, S., Joshi, J.B.D.: Supporting authorization query and inter-domain role mapping in presence of hybrid role hierarchy. The 11th ACM SACMAT. pp. 228–236. (2006)

[4] Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed nist standard for role-based access control. ACM Trans. Inf. Syst. Secur. 4, 224–274 (2001)

[5] Gligor, V., Gavrila, S.L., Ferraiolo, D.: On the formal definition of separation-of-duty policies and their composition. IEEE Symposium on Research in Security and Privacy. pp. 172–183 (1998)

[6] Grimshaw, A., Morgan, M., Merrill, D., Kishimoto, H., Savva, A., Snelling, D., Smith, C., Berry, D.: An open grid services architecture primer. Computer 42, 27–34 (2009)

[7] Joshi, J.B.D., Bertino, E.: Fine-grained role-based delegation in presence of the hybrid role hierarchy. The 11th ACM SACMAT. pp. 81–90. (2006)

[8] Joshi, J.B.D., Bertino, E., Ghafoor, A.: Temporal hierarchies and inheritance semantics for gtrbac. In: Proc. of the 7th ACM SACMAT. pp. 74–83. (2002)

[9] Joshi, J.B.D., Bertino, E., Latif, U., Ghafoor, A.: A generalized temporal role-based access control model. IEEE Trans. on Knowl. and Data Eng. 17, 4–23 (2005)

[10] Q. M. S. Osborn, R. Sandhu, "Configuring role-based access control to enforce mandatory and discretionary access control policies," 2000.

[11] Kuhn, D.R.: Mutual exclusion of roles as a means of implementing separation of duty in role-based access control systems. The 2nd ACM workshop on Role-based access control. pp. 23–30. RBAC '97(1997)

[12] Li, N., Tripunitara, M.V., Bizri, Z.: On mutually exclusive roles and separation-of-duty. ACM Trans. Inf. Syst. Secur. 10 (2007)

[13] Martino, L.D., Tech, I., Ni, Q.: Multi-domain and privacy-aware role based access control in ehealth. The 2nd International Conference on In Pervasive Computing Technologies for Healthcare (2008)

[14] Piromruen, S., Joshi, J.B.D.: An rbac framework for time constrained secure interoperation in multi-domain environments. IEEE Workshop on Object-oriented Real-time Databases (2005)

[15] Sandhu, R.: Role Activation Hierarchies. In Proc. of the 2rd ACM Workshop on Role-based Access Control, Fairfax, Virginia, 22-23, 1998.

[16] Shafiq, B., Joshi, J.B.D., Bertino, E., Ghafoor, A.: Secure interoperation in a multidomain environment employing rbac policies. IEEE Trans. on Knowl. and Data Eng. 17, 1557–1577 (2005)

[17] Shehab, M., Bertino, E., Ghafoor, A.: Serat: Secure role mapping technique for decentralized secure interoperability. In: Proc. of the 10th ACM SACMAT. pp. 159–167. (2005)

[18] Tang, Z., Li, R., Lu, Z.: A request-driven role mapping for secure interoperation in multi-domain environment. The 2007 IFIP International Conference on Network and Parallel Computing Workshops. pp. 83–90. (2007)

[19] Wickramaarachchi, G.T., Qardaji, W.H., Li, N.: An efficient framework for user authorization queries in rbac systems. In: Proc. of the 14th ACM SACMAT. pp. 23–32. (2009)

[20] Zhang, S., Kong, X., Wang, B.: Study on role-spliting and its ontology-based evaluation methods during role mapping of inter-domain. In: Proc. of the 2008 International Conference on Computer Science and Software Engineering - Volume 03. pp. 642–645. (2008)

[21] Zhang, Y., Joshi, J.B.D.: A request-driven secure interoperation framework in loosely-coupled multi-domain environments employing rbac policies. In: Collaboratecom: Networking, Applications and Worksharing. pp. 25–32. IEEE Computer Society, (2007)

[22] Zhang, Y., Joshi, J.B.D.: Uaq: a framework for user authorization query processing in rbac extended with hybrid hierarchy and constraints. In: The 13th ACM SACMAT. pp. 83–92. (2008)