Multi-session Separation of Duties (MSoD) for RBAC

David W Chadwick[1], Wensheng Xu[2], Sassa Otenko[1], Romain Laborde[3], Bassem Nasser[1]
[1]*University of Kent, UK;* [2]*Beijing Jiaotong University, China;* [3]*IRIT, France*
{d.w.chadwick, o.otenko, b.nasser}@kent.ac.uk; xuwsh2002@yahoo.com.cn; laborde@irit.fr

Abstract

Separation of duties (SoD) is a key security requirement for many business and information systems. Role Based Access Controls (RBAC) is a relatively new paradigm for protecting information systems. In the ANSI standard RBAC model both static and dynamic SoD are defined. However, static SoD policies assume that the system has full control over the assignment of all roles to users, whilst dynamic SoD policies assume that conflicts of interest can only arise during the simultaneous activation of a user's roles. Unfortunately neither of these assumptions hold true in dynamic virtual organisations (VOs), or in business processes that span multiple user sessions, or where users only partially disclose their roles at each session. In this paper we propose multi-session SoD (MSoD) policies for business processes which include multiple tasks enacted by multiple users over many user access control sessions. We explore the means to define MSoD policies in RBAC via multi-session mutually exclusive roles (MMER) and multi-session mutually exclusive privileges (MMEP). We propose an approach to expressing MSoD policies in XML and enforcing MSoD policies in a policy controlled RBAC infrastructure. Finally, we describe how we have implemented MSoD policies in the PERMIS Privilege Management Infrastructure

1. Introduction

Separation of duties (SoD) is widely considered to be a fundamental security principle for business and information systems [1]. The concept of SoD has long existed in the physical world. For example, staff members in a bank are assigned to different posts with different duties. This ensures that cooperation of multiple staff members is required to perform all the tasks of a complete business process, either sequentially or concurrently, so that accountability can be enforced and damage caused by either a single member's mistake, or an accident or deception can be avoided or minimized. Many organizations require that the request and approval of a major expenditure be done by two separate people. SoD is an important protection mechanism for handling important business processes or information, and it should be integrated with all access control mechanisms, such as discretionary access control (DAC), mandatory access control (MAC), and especially role based access control (RBAC) since most tasks within organizations are performed by roles.

Significant research about the use of SoD policies in RBAC has been performed and this is reflected in the ANSI standard [2]. The ANSI standard considers SoD as constraints on the roles that can be assigned to users at any time (Static SoD - SSD) or constraints on the roles that a user can activate simultaneously within one or more user sessions (Dynamic SoD - DSD). The ANSI standard assumes that static SoD policies can be enforced by the administrative function at role assignment time because the administrative system has full control over the assignment of all roles to users, whilst dynamic SoD policies can be enforced at role activation time because conflicts of interest can only arise when a user's conflicting roles are active at the same time.

Unfortunately neither of these assumptions hold true in all business environments. In dynamic virtual organisations (VOs) when multiple independent role allocating authorities exist, SSD cannot be enforced at role assignment time since no single administrative function will know all the roles that have already been assigned to any single user. In business processes where conflicts of interest exist over extended periods of time, during which a user may have invoked and terminated multiple access control sessions, DSD cannot be enforced at role activation time because a user may never activate conflicting roles simultaneously. When a user only partially discloses his roles at each session, the standard RBAC policy constraints cannot solve the SSD or DSD problems. In both these cases the current access control decision may depend on previous access control decisions that took place in some previous user access control

session. Hence the access control system needs to retain a history of previous access control decisions.

In Section 2 we analyze multi-session SoD (MSoD) policies in RBAC and propose multi-session mutually exclusive roles (MMER) and multi-session mutually exclusive privileges (MMEP) to enforce MSoD policies. In section 3 we propose a way of expressing MSoD policies in XML. In Section 4 we describe the enforcement procedure for MSoD policies in an RBAC system. In Section 5, we give a practical example of how we have integrated MSoD policies into the PERMIS [11] RBAC authorization infrastructure. Finally, in Section 6, we review related research, give the limitations of our research, and provide our conclusions. But first we give two motivating examples of SoD policies, one SSD and one DSD, in which the different tasks may be carried out in different user access control sessions by different roles. These will be used throughout the paper to highlight the concepts and designs that we have chosen.

Example 1. Cash processing in a bank. In a bank, a staff member who is authorized to be a Teller may not be allowed to be an Auditor of the same bank. That is, an employee cannot simultaneously hold the roles of auditor and teller. However, some auditing, e.g. an annual one, may take place much later than the cash handling, during which time many staff may have changed their roles, and a cashier might have been promoted to an auditor. This is an example of a SSD policy over multiple user sessions with potentially many role occupant changes during the operation of the policy. Obviously a conventional SSD policy cannot express such a constraint since no user might ever have been assigned the conflicting roles at the same period of time and so the conventional SSD policy will never have been violated.

Example 2. Tax refund taken from [12]. To carry out a tax refund process, four different tasks need to be executed sequentially:

Task T1: A clerk prepares a check for a tax refund.
Task T2: A manager can approve or disapprove the check. This task should be performed in parallel twice by two different managers.
Task T3: The decisions of the managers are collected and the final decision is made. The manager who collects the results must be different from those executing task T2.
Task T4: A clerk issues or voids the check based on the result of task T3; the clerk issuing or voiding the check must be different from the clerk who prepared the check.

This is an example of DSD which conventional RBAC policies cannot specify because a user may activate the manager or clerk role only, but this does not violate the conventional DSD policy. Whilst a manager (or clerk) is authorized to perform several tasks, in this example he may only perform one of them in one tax refund process instance. Furthermore, one tax refund process instance might span multiple user sessions, so a manager (or clerk) who has performed a task in an earlier session may not be authorised to perform any task in a subsequent session. Since the DSD policy must span several tasks over a period of time, and may involve different user sessions, then traditional RBAC DSD cannot work in this scenario because it only controls the simultaneous activation of user roles. Bertino et al solved the DSD problem in this workflow environment [12], but their solution only works in a centralised homogeneous system and is integrated with the workflow system. In this paper, we will provide a solution which can deal with DSD in a multi-session distributed heterogeneous environment that is not tied to workflows.

2. Multi-session separation of duty (MSoD)

2.1. Static and dynamic SoD in the standard RBAC model

In its simplest form, a SoD policy states that if a sensitive business process is comprised of two tasks, then different people must perform each task. More generally, when a business process is comprised of n tasks, a SoD policy requires the cooperation of at least k ($1< k \leq n$) different users to complete the process. The purpose of separation of duties is to prevent one person from doing all the tasks of a sensitive process that should require two or more people in order to prevent mistakes or fraud.

RBAC is perceived to be one of the most efficient and flexible approaches to enforcing security policies in industrial and commercial application systems. Privilege management in RBAC [2] is described in terms of users, roles, operations, and objects (Figure 1). Users are assigned to roles in a many to many relationship. If a user is a person, a role can represent a job function, a qualification or expertise. Privileges (termed permissions in ANSI RBAC [2]) represent the right to perform an operation on an object, and are granted to different roles in a many to many relationship. A user may have multiple roles but activates only a subset of these within any user access control session. A user must be active in a role before he can exercise the privileges of that role.

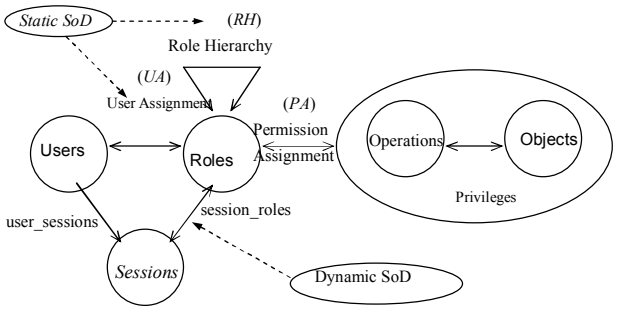


Figure 1. The ANSI RBAC Model

If a user is not allowed to be assigned conflicting roles according to the role assignment policy, this is a SSD policy. For example, in a bank, normally a staff member is not allowed to hold both the teller and auditor's roles at the same time (Example 1). If on the other hand, a user is allowed to hold but not to simultaneously activate conflicting roles (in one or more user sessions) this is a DSD policy. Standard RBAC can only provide a solution to the DSD problem described in Example 2 above by artificially defining different roles for each of the five tasks.

The above two types of SoD policy ultimately control the roles a user may activate simultaneously (in one or more sessions) and so they are not sufficient to satisfy SoD requirements for more complex situations in either a process environment or a multi domain RBAC system. In a multiple domain scenario, e.g. a virtual organization (VO), where the user roles are assigned by multiple authorities and used to access resources in different domains, a user may be assigned different roles by different domain administrators and no single RBAC system may know the full set of a user's roles. So if a user is assigned two conflicting roles (conflicting according to the SoD policy of the resource domain) by two different administrative domain authorities, but he presents only one role each time to the resource he wishes to access, then his access requests will not be against either the static or dynamic SoD policies of the resource domain. To cater for this type of security situation, we need a new SoD policy which we call a **multi-session SoD** (MSoD) policy, which can be associated with the multiple temporal user access control sessions, i.e. access control decisions in the current session may depend on accesses that were granted earlier in time in previous sessions. Thus the user who activates and utilises a role in one session will be forbidden to activate the conflicting role(s) in subsequent later sessions. For the bank cash processing example, the MSoD policy can state that, if a person has ever acted as a Teller (or an Auditor) before some event such as the annual audit, then he will no longer be authorized to activate the role of Auditor (or a Teller) now. Thus a current access control decision will depend upon earlier decisions in previous user access control sessions.

2.2. Business context of MSoD

Whilst MSoD policies implicitly concern multiple temporal user sessions, the scope of an MSoD policy is not bound to any particular set of user sessions. Some MSoD policies may apply for a long period of time, e.g. for a year, or for as long as a VO persists; but other MSoD policies may only apply for the execution of a business process instance, such as the tax refund in Example 2. In that example, the MSoD policy needs to state that, for a tax refund process instance, if a clerk has been authorized to do Task 1, then he/she will no longer be authorized to do Task 4, regardless of whether it is in the same user session or not. Note that this MSoD policy only restricts the clerk's access requests within a business process instance; the same clerk is authorized to do either Task 1 or Task 4 in a different tax refund process instance within the same or a different user session. In both cases, the history of prior access control decisions in the same or previous user sessions is needed in order to make the correct current decision.

In order to determine the scope of an MSoD policy, we introduce the concept of a business context. The scope of an MSoD policy is specified by reference to a business context, rather than to user sessions. A business context (and hence an MSoD policy) may span multiple user sessions, or a user session may span several business contexts. In this way we remove MSoD policies from any dependency on user sessions, and put the dependency back where it belongs, in the business context for which the separation of duties applies.

A business context is therefore the set of business processes throughout which an MSoD policy must persist. An MSoD policy can apply across all the instances of a business context or to each separate instance of a business context. The former is equivalent to SSD within a business context, the latter to DSD within a business context instance. For example, an MSoD policy for the business context of issuing cheques, may say either that one group of people must complete the amounts on cheques and a different group sign the cheques (i.e. SSD across all business context instances) or different people must complete and sign the same cheque (i.e. DSD per business context instance). We propose that all business contexts are related together in a context hierarchy. The most generic business context is the universal context for an organization (or VO), and this contains all the business

processes carried out by that organization (or VO). The most refined business context is a business process that contains at least 2 tasks that have separation of duties constraints (as in Example 1), or 1 task that cannot be repeated by the same person (as in task T2 of Example 2). The mechanism we use to relate one business context to another is by hierarchically naming them, using a set of type-value pairs, where *type* is the business context and *value* is an instance of the context. The universal context forms the root of this context hierarchy and its name is null. Consequently, there can only be one active instance of the universal context of an organization (or VO) at any point in time, but subordinate contexts can have multiple active instances at any time.

When writing MSoD policies we need to be able to specify both SSD across all instances of a business context and DSD per business context instance. We use the value notation * to signify the former, and ! to signify the latter. In Example 1 the MSoD policy applies to the entire bank (the universal context) across all its branches for each period of an audit, and so the business context may be referred to in the MSoD policy as "Branch=*, Period=!" (see Figure 2). If instead the MSoD policy had applied to each separate branch of the bank (meaning that an employee could be a teller in one branch and an auditor in another branch), then the policy business context for the bank might be referred to as "Branch=!, Period=!". If the policy had applied only to the York branch, the policy business context might be named "Branch=York, Period=!".

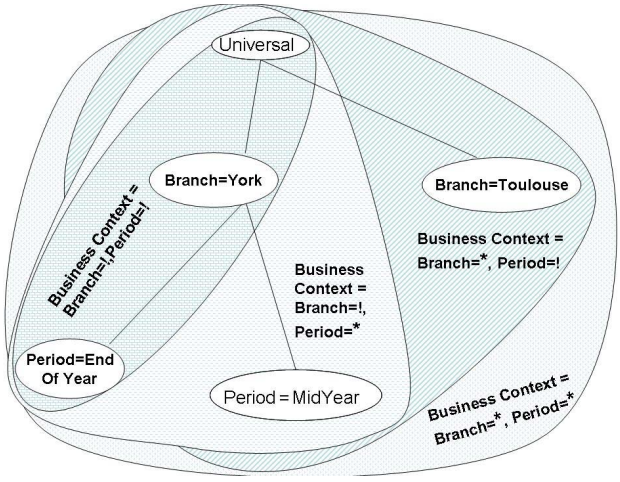


Figure 2. Various Policy Business Contexts applied to a Business Context Instance Hierarchy

If an organization (or VO) needs to enforce MSoD policies across two sequential business processes, e.g. cash dispensing followed by cash reconciliation, then there is always a super-context that joins them together e.g. cash processing (i.e. there is some super-goal that the organization wants to achieve) since all business contexts for an organization (or VO) are always part of the same universal hierarchy. Note that knowledge of how the different business contexts relate together within the hierarchy is part of the application schema that stores the hierarchy of contexts. The security policy and access control system do not need to know this. The security policy contains sufficient knowledge of how two or more business contexts relate to each other via the hierarchical names of each context.

When evaluating an MSoD policy, the access control system needs to know which particular business context is currently active. Consequently we propose that each access control request carries the value of the current business context instance. This is sufficient for the access control system to evaluate the MSoD policy (see Section 4). But it is not the most efficient method of evaluation since a large amount of unnecessary history information will need to be retained by the access control system. Consequently we propose that an MSoD policy optionally contains the starting task and ending task of a business context, to indicate during which part of the business process the context is active from an MSoD perspective, and during which time history information needs to be retained. If the starting and ending tasks are missing from the policy, the access control system must record history information for each instance of an active business context from either the first time an instance is mentioned or the system can infer it has started (because a contained business context has started), until either the system can infer that the instance has finished (because a containing business context completes, or no longer exists), or management procedures delete the history information.

2.3. Multi-session mutually exclusive roles

A convenient way to specify SoD policies in RBAC is by defining mutually exclusive roles (MER). In the RBAC standard model [2], a MER constraint forbids a user from holding (SSD) or activating (DSD) m from n $(n \geq m)$ $(n \geq 2)$ different roles $(r_1, r_2, \ldots, r_n)$ **at the same time**, so as to enforce a SoD policy. For a set of n SoD roles $(n \geq 2)$, a MER constraint can be expressed as m-out-of-n mutually exclusive roles: MER($\{r_1,\ldots,r_n\}$, m), where each r_i is a role, n and m are integers, $1 < m \leq n$. A MER constraint defines a set of conflicting roles in an organization, for example MER({teller, auditor}, 2) will forbid a user from possessing or activating 2 conflicting roles in the SoD role set. Whilst separation of duties is the security objective and a SoD policy defines which duties must

be separate, MER is a constraint imposed on users' roles in an RBAC system as a means of implementing SoD policies. MER obviously fails when it is not known how many roles a user possesses and the user chooses to selectively activate different roles in different sessions, or when there are constraints on what tasks a role can perform which depend upon the previous tasks already undertaken by the user.

To express and implement MSoD, we propose a new type of role constraint, which we call multi-session mutually exclusive roles (MMER). A MMER constraint can be denoted as an m-out-of-n constraint, which contains *n* MSoD roles in which *m* or more are conflicting with each other and cannot be activated by a user **in a particular business context**.

For each r_i (i=1, .. *n)* MSoD roles ($n \geq 2$), a MMER rule can be expressed as a set of *n* MSoD roles with a forbidden role cardinality *m*:

MMER($\{r_1,\ldots,r_n\}$, *m*, BC)

where BC identifies the particular (hierarchically named) business context to which the *m* mutually exclusive roles apply, in which each r_i (i=1, .. n, n≥2) is a role, and $1 < m \leq n$. In this case, a user is forbidden to activate *m* or more roles among $\{r_1,\ldots,r_n\}$ **in the same business context [instance],** so as to enforce an MSoD policy. For the cash processing example, Teller and Auditor are mutually exclusive within the bank in any auditing period, so the policy can be denoted as MMER({Teller, Auditor}, 2, "Branch=*, Period=!"). Since contexts are hierarchically related, all contexts which are equal or subordinate to the context in the MMER rule should be applied with the MMER rule.

2.4. Multi-session mutually exclusive privileges

To achieve more flexible and finer grained access controls and MSoD, only part of the privileges (i.e. operations on certain targets) granted to a role can be made mutually exclusive. E.g. in example 2, the role of a manager can have both the privileges of approving/disapproving a tax refund application and summarizing tax refund decisions. But approving/disapproving a tax refund application and summarizing tax refund decisions for the same tax refund application are two mutually exclusive privileges, so the user who has the role of a manager should not be allowed to perform both Tasks T2 and T3 for the same tax refund process instance. Furthermore, Task T2 needs to be performed twice, and any manager should only perform this task once. Mutually exclusive privileges for a process instance are called multi-session mutually exclusive privileges (MMEP) in this paper.

MMEP constraints are defined as follows. For *n* MSoD privileges ($n \geq 2$), a MMEP constraint can be expressed as a set of *n* MSoD privileges with a forbidden privilege cardinality *m*:

MMEP($\{p_1,\ldots,p_n\}$, *m*, BC)

where BC is the business context [instance] containing the *m* mutually exclusive privileges p_i (i = 1, .. n), where $1 < m \leq n$. In this case, a user is forbidden to perform *m* or more privileges among $\{p_1,\ldots, p_n\}$ in the same business context [instance]. For the tax refund example, approving/disapproving a tax refund application (p1) and summarizing tax refund decisions (p2) are two mutually exclusive privileges in any single instance. We may denote this as MMEP($\{p_1$, $p_2\}$, 2, "…., taxRefundProcess=!"). (Note. Each privilege will be expressed as an operation and associated object in a complete policy). A manager is also forbidden from exercising the same approving/disapproving tax refund privilege (p1) more than once, so this privilege is also mutually exclusive. We can denote this as MMEP($\{p_1$, $p_1\}$, 2, "…., taxRefundProcess=!").

3. MSoD policies in XML

Many modern day access control policies are now being written in XML e.g. XACML [7], X-Sec [14], PERMIS [11] and Akenti [6]. In this section we describe a generic XML MSoD policy for RBAC systems based on MMER and MMEP constraints. The business context is identified by its unique hierarchical name, and optionally by its first and last steps (Operations on Objects i.e. tasks) that are subject to the MSoD constraints. The first step tells the policy decision point (PDP) when to start enforcing MSoD and the last step when to stop enforcing it, for each business context [instance]. If the first step is omitted, the PDP must start to enforce MSoD from whatever is the first operation that is invoked inside the business context [instance] or any contained context [instance]. If the last step is omitted, the PDP may infer that a business context is no longer active if a containing business context [instance] is terminated (since all the contained ones must also be terminated). Otherwise termination of the MSoD policy enforcement must be done by administrative means. We suggest how this can be implemented in section 4.3. One or more MMER and/or MMEP constraints can be listed for each MSoD policy. The MMEP (MMER) constraints contain sufficient information for the PDP to know if two or more task steps (roles) are conflicting for the particular business context instance. The MSoD policies for the two examples – bank cash processing and tax refund process, are as follows.

```
<MSoDPolicySet>
    <MSoDPolicy BusinessContext="Branch=*, Period=!">
<!-- policy applies for each instance of period across all
branches of the bank -->
    <LastStep operation="CommitAudit"
     targetURI="http://audit.location.com/audit"/>
    <MMER ForbiddenCardinality = "2">
            <Role type="employee" value="Teller"/>
            <Role type="employee" value="Auditor"/>
     </MMER>
     </MSoDPolicy>
     <MSoDPolicy BusinessContext="TaxOffice=!,
    taxRefundProcess=!" />
    <!-- policy applies for each instance of taxRefundProcess
    in each tax office -->
   <FirstStep operation= "prepareCheck"
     targetURI="http://www.myTaxOffice.com/Check"/>
   <LastStep operation="confirmCheck"
     targetURI="http://secret.location.com/audit"/>
   <MMEP ForbiddenCardinality= "2">
            <Operation value= "prepareCheck"
target="http://www.myTaxOffice.com/Check"/>
            <Operation value= "confirmCheck"
target="http://secret.location.com/audit"/>
    </MMEP>
    <MMEP ForbiddenCardinality= "2">
            <Operation value= "approve/disapproveCheck"
target="http://www.myTaxOffice.com/Check"/>
            <Operation value= "approve/disapproveCheck"
target="http://www.myTaxOffice.com/Check"/>
            <Operation value="combineResults"
target="http://secret.location.com/results"/>
     </MMEP>
     </MSoDPolicy>
</MSoDPolicySet>
```

In the first example, the business context is every audit period across the entire bank. The first step of the business context is missing, meaning that the PDP will start to enforce MMER as soon as any operation in any period is invoked that contains this or any subordinate business context. This will forbid any user to invoke both Auditor and Teller roles until CommitAudit is invoked in the same period, regardless of which branch the operation was invoked in. After auditing has been completed, and the CommitAudit operation is invoked, MMER enforcement for this business context instance is finished, and the history information is deleted.

In the second example, the mutually exclusive privilege constraints contain the same privilege (approve/disapproveCheck) twice, which means that a user is only allowed to perform this privilege at most once in each context instance.

4. Enforcement of MSoD policies

4.1. Access control framework for MSoD enforcement

A standard RBAC system knows all the roles assigned to each user, therefore SoD constraints can be imposed either at the role assignment stage when roles are being assigned to users, or at the role activation stage when roles are being activated for a user session, as depicted in Figure 1. Since MSoD constraints are associated with multiple tasks in business contexts that span multiple user sessions, and/or user roles that are allocated by multiple domain authorities, enforcement of MSoD constraints is only possible at the access control decision making stage. This requires the current access control decision to be dependent upon the previous access control decisions for the same business context. The ISO Access Control Framework [13] has this feature built into its model. To make history dependent access control decisions, a Retained Access control Decision Information (ADI) component is specified. The Retained ADI is responsible for recording and maintaining information about all previous access control decisions, so that current decisions made by the Access control Decision Function (ADF) – the ISO term for the Policy Decision Point (PDP) – can properly take into account policy constraints such as MSoD (see Figure 3).

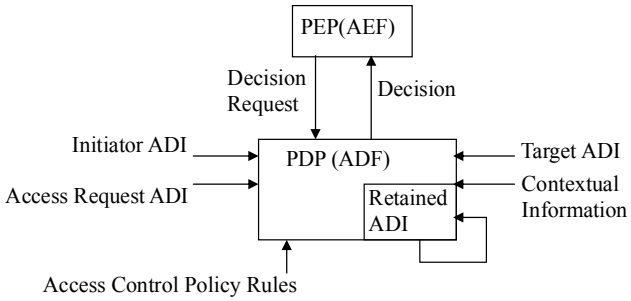


Figure 3. Access control framework from ISO 10181-3 with the Retained ADI

Normally the following information needs to be passed from the Access control Enforcement Function (AEF) – the ISO term for the Policy Enforcement Point (PEP) – to the ADF/PDP in order to make an RBAC decision for a user: 1) the user's attributes/roles (optionally including the user's ID), 2) the requested operation and its parameters, 3) the requested target object (identified by a set of attributes) and 4) any environmental or contextual information such as the time of day.

In order to make multi-session access control decisions, the user's ID becomes mandatory so that the ADF/PDP can link together the user's sessions. We also require 5) the business context instance so that the

ADF/PDP can determine which MSoD policy applies. Conceptually, the business context instance could be regarded as part of the contextual information, but we prefer to keep it as a separate parameter because special matching rules apply to it (see later). The PEP, being part of the application, is easily able to identify the business context instance of each user request[1]. Based on all these access request parameters, and the information from previous access control sessions stored in the retained ADI, MSoD policies can now be enforced by the PDP.

4.2 MSoD policy enforcement procedure

MSoD policies are a component of RBAC policies. When a PDP first initialises, it must read in the RBAC policy including the MSoD component. It also needs to initialise the retained ADI from previous sessions. The retained ADI is conceptually secure stable storage holding previous access control decisions, and this can be implemented in a variety of ways e.g. an encrypted secure database, or a tamperproof audit trail such as [5]. When the PDP initializes, it needs to reconstruct the retained ADI from this physically secure stable storage. For MSoD, each record in the retained ADI needs to contain 1) user's ID, 2) user's activated role(s), 3) operation granted, 4) target accessed, 5) business context instance, and 6) time/date of grant decision. The latter parameter is needed for administrative purposes (see later).

As shown in Figure 3, when the PEP requests the PDP to make an access control decision, the 5 sets of parameters mentioned in 4.1 above are passed from the PEP to the PDP. The PDP first performs its normal checking against the RBAC policy, and if the interim result is grant, then the PDP will further perform the following algorithm to check each user's access request against the MSoD set of policies. Input to the algorithm comprises the user's ID, user's activated role(s), operation granted, target to be accessed and business context instance name. The return value is unaltered or set to Deny access.

1) Match the input business context instance against the business contexts in the MSoD set of policies. If there is no match EXIT. (Matching is based on the context naming hierarchy. If the input context instance is equal to or subordinate to any of the contexts in the set of MSoD policies, then a match with a policy business context is flagged[2]. If there are multiple matches then all policies apply and are selected.) If a matched policy pertains to a single business context instance (!), replace policy business context with the instance of the input business context.
2) For each matched MSoD policy do the following. When no more, EXIT.
3) Match the policy business context against the business context instances stored in the retained ADI. (Retained ADI context instance matches if it is equal or subordinate to policy context, noting that policy context of * matches all instance values.) If there are one or more matches goto 5).
4) Check if the requested operation is the first step in the matched policy business context. If it is, or if there is no first step in the policy, add a new entry to the retainedADIlist then goto 7).
5) For each MMER in the policy, do
   i. Match activated role(s) against MMER role(s). Number of matched roles = n_r
   ii. If no match goto next MMER.
   iii. Ignoring n_r current matched role(s) in MMER, count number of remaining roles in the MMER that match roles from retained ADI for this user ID and matched policy business context.
   iv. If count LT (ForbiddenCardinality-n_r) add n_r new records to retainedADIlist for activated role(s) and goto next MMER, else set DENY and EXIT.
6) For each MMEP in the policy, do
   i. Match requested operation and target against MMEP privilege(s).
   ii. If no match goto next MMEP.
   iii. Ignoring current matched operation and target in MMEP, count number of remaining operation and targets in the MMEP that match an operation and target from retained ADI for this user ID and matched policy business context. If count LT (ForbiddenCardinality-1) add a new record to retainedADIlist for current operation and target and goto next MMEP, else set DENY and EXIT.
7) If requested operation equals last step for this MSoD policy business context (i.e. business context is terminated), then delete every record from retained ADI that has a matching business context instance (i.e. equal or subordinate to policy business context), else store retainedADIlist in retained ADI.

[1] The policy writer also needs to know what the business contexts are in order to construct a correct policy, but this is no different from the current requirement of needing to know the correct operations, targets, roles etc.

[2] Note that we do not need to be concerned with business contexts that are superior to ones in the MSoD policy, since their absence from the policy means that there are no constraints at this higher level.

8) Goto 2).

Note that if the access request is denied, then no change needs to be made to the retained ADI database, as it has no effect on future RBAC with MSoD decisions. Only granted decisions are stored in the retained ADI.

4.3. Explicit management of the retained ADI

The retained ADI is a core component for implementing MSoD in an RBAC PDP. Entries for a business context instance are added to the retained ADI after the first step of the business context is initiated, and they are removed from the retained ADI after the last step of the business context has finished. Providing the policy contains the last step of a business context, or it can be implied, then no administrative management of the retained ADI is needed. But for cases where a business context has no defined or implied last step, then a control mechanism is needed to manage the retained ADI, otherwise it will get too large and performance will be degraded. (Note that there are no security implications from not purging the retained ADI, only performance implications.) We propose that a management port on the PDP can be used to manage the retained ADI, by treating the retained ADI as a target resource that only trusted administrators are allowed to access via the PDP's management port. We can securely maintain the retained ADI, by defining an RBAC policy to protect it. A new role of say "RetainedADIController" is created with privileges to perform some operations on the retained ADI such as "remove record" or "purge". We plan to implement this feature next.

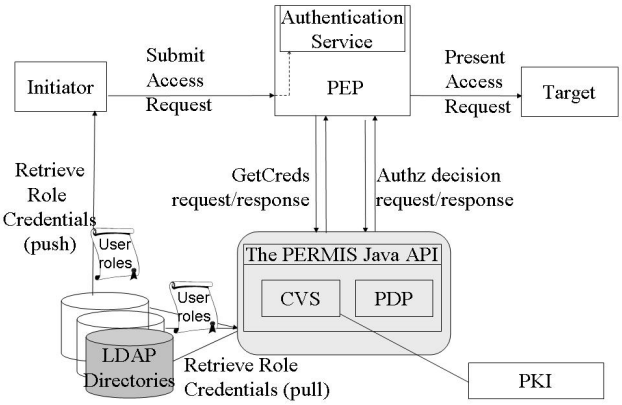
5. Implementation of MSoD in PERMIS

PERMIS is a Privilege Management Infrastructure whose core component is an RBAC decision making PDP [11]. Here we show how MSoD policies have been implemented in PERMIS.

5.1. Structure of PERMIS

The PERMIS infrastructure comprises three sub-systems: a privilege allocation (PA) sub-system for allocating roles to users, a policy management sub-system for creating RBAC policies, and a credential verification service/policy decision point (CVS/PDP) sub-system for granting or denying user's access to resources. We are primarily concerned with the latter in this paper. User's roles and attributes are typically stored in one or more LDAP directories. They are usually transported as digitally signed credentials, encoded as either SAML assertions [19] or X.509 attribute certificates [20]. The function of the CVS is to validate these credentials and extract the valid roles and attributes from them, so that the PDP can make an access control decision. It is at this point that MSoD policies can be imposed by the PDP. The PERMIS CVS/PDP sub-system structure is shown in Figure 4.

Figure 4. Sub-system structure of PERMIS CVS-PDP



5.2. Implementation

Every time an access control decision request is passed to the PERMIS PDP, the request and the response are logged in a secure audit trail [5]. This creates a cryptographically protected log of events in stable storage. Any granted decision that involves an MSoD policy is stored as retained ADI in memory as the 6 tuple defined in Section 4.2, as well as in the secure audit trail. When a granted last step is recorded in the audit trail, the retained ADI records for that business context instance and any subordinate instances are flushed from memory (but not from the audit trail). At start up, the PDP reads in its policy, and then processes the last *n* audit trails starting from time *t* (where *t* and *n* are administrative parameters). It extracts the retained ADI from these according to its current set of MSoD policies. Once its retained ADI is recovered to memory, the PDP is ready to start making access control decisions again.

By adding the business context instance to the list of environmental parameters that are already passed to the PERMIS PDP, we have not needed to alter the Java API to PERMIS in order to support multiple session separation of duties policies. The code is expected to be publicly released as open source via the US NMI release [17] in 2007.

6. Related Work, Limitations and Conclusions

SoD has been widely studied by many researchers. Sandhu [4] presented one of the earliest papers to explicitly identify SoD as an issue in business transactions. His work predated that on RBAC, and his solution used a history based transaction control model and expressed SoD rules by assigning conflicting tasks to differently named roles in order to avoid collusion or conflict of interest. The expression and enforcement of the transaction control model and the SoD rules are application specific and authorization for SoD is based on users' identities, not on roles – i.e. users with different user identities are required to execute conflicting tasks, so it is not based on RBAC, and does not form part of a generic RBAC model. In contrast we have provided an RBAC mechanism for SoD, and we have expressed the SoD rules as a generic RBAC sub policy in XML, whilst authorization is based on users' roles, not on their identities (although SoD enforcement is based on their identities).

Kuhn [3] discussed mutual exclusion of roles for SoD and analyzed the properties of different situations of mutually exclusive roles – partial and complete exclusion, authorization-time and run-time exclusion. But Kuhn's work is only on static and dynamic SoD, it doesn't solve the problem in a business process environment when authorization spans multiple user sessions. In contrast, our work has solved the multi-session SoD problem in RBAC.

Simon and Zurko [8] proposed SoD policies in a role-based environment in the form of condition rules. They also discussed the concept of history based SoD, but no XML policy was discussed nor how the rules could be integrated with RBAC policies. Finally no enforcement or implementation mechanism was discussed.

Gligor et al. [9] discussed SoD policies in RBAC and gave formalized expressions of SoD policies in different situations – Static SoD, Dynamic SoD, Object-based Static SoD, Object-based Dynamic SoD, Operational Dynamic SoD, History based Dynamic SoD, etc. Their work provides an excellent formalization of SoD policies at the conceptual level. But business process contexts are not explicitly expressed in their work, and no XML policy was presented. Finally no enforcement or implementation mechanism of SoD in RBAC was discussed in their work.

Ahn et al. proposed a constraint language – RCL 2000 (Role based Constraint Language 2000), to support role-based SoD constraints in RBAC [10], and static and dynamic SoD can be supported by their language. But RCL 2000 is a proprietary notation, and it needs to be further extended to support history based SoD. Finally, no enforcement mechanism for applications was presented in their work.

In [18], Crampton treated SoD in RBAC systems. He analyzed different types of separation of duty as user based separation (no role in a specified set can be assigned to a set of users), role-based (no user should be assigned a set of roles), permission-based, and object-based. Compared to this taxonomy, our work focuses on role-based and permission-based SoD. Crampton proposes to enforce SoD via an anti-role. As a role is associated with a set of permissions, an anti-role is associated with a set of prohibitions that constitute a blacklist for each user. Crampton proposes that implementations should periodically purge the assignments of sanitized permissions, thus deleting the anti-role effect. In comparison we have proposed a better solution using the business context to define the scope of an MSoD policy, and deletion of the retained ADI only after a business context has terminated.

In [12] Bertino et al propose a solution for SoD in workflow applications that is not history based, but rather computes the set of all possible role and user assignments that dont violate the SoD policy and other constraints, prior to workflow commencing. Then when a user asks to activate a role, it checks if this is possible, and after the task finishes prunes the rules to make future evaluations faster. However the solution is based on a central authority that knows all the users, roles and user role assignments whilst our solution does not have this restriction and can work in a distributed environment. Bertino's solution focuses solely on SoD within workflows and requires prior specification and knowledge of the workflow and its tasks. In contrast our approach does not require knowledge of all (or any of) the workflow tasks. Furthermore some examples of SoD are not related to workflows, as in Example 1. Our approach can cater for this whilst Bertino's cannot.

Our work is not without its limitations though. Firstly, the PDP needs to know the name or ID of the user who has activated the roles, in order to link the different access control sessions of the user together. In a pure RBAC system the PDP does not need to know the name of the user, and can make access control decisions based solely on the user's roles. The original PERMIS PDP can do this, but the MSoD PERMIS PDP needs the user's ID in order to enforce multiple session SoD. Secondly, we have assumed that the user will have the same ID for each session, and that each role or attribute is linked to the same user ID. The first assumption does not always hold true, for example, in Shibboleth [15] a user is given a different handle ID for each session. If this was the only ID ever delivered

to the PDP it would not be possible to support MSoD with Shibboleth. However, it is possible to configure Shibboleth to return the user's ID along with their other attributes, in which case MSoD can be supported. The second assumption does not always hold true either. In a multi-authority VO, each authority may use different identifiers for identifying the same user. The Liberty Alliance model [16] works on this basis. Thus a user could use one identity from one authority to activate one role e.g. clerk, and another identity from another authority to activate a second role e.g. auditor. Our MSoD procedure would not be able to detect this. However the Liberty Model supports identity linking between pairs of authorities, providing each service provider with a one way alias for identifying the same user in a different authority (even though it does not know the user's true identity in each authority). In this way MSoD can be enforced by linking the user's aliases to the local identity, and basing the MSoD policy on the local identity. Finally, we anticipate that our current implementation will not be scalable, due to the time taken to initialize the retained ADI from the secure audit trails. Thus our next implementation will use a secure relational database to store the retained ADI instead of in-core memory. Nevertheless the concepts and policies described in this paper will remain the same.

7. Acknowledgments

The authors would like to thank UK JISC for funding this work under the DyCom project.

8. References

[1] N. Li, Z. Bizri, M. V. Tripunitara. "On mutually-exclusive roles and separation of duty". CCS'04, October 25-29, 2004, Washington, DC, USA. pp.42-51.
[2] American National Standards Institute, International Committee for Information Technology Standards (ANSI/INCITS). "Information Technology - Role Based Access Control" ANSI INCITS 359-2004
[3] D. Richard Kuhn. "Mutual exclusion of roles as a means of implementing separation of duty in role-based access control systems". Proceedings of the second ACM workshop on Role-based access control, pp.23-30, 1997
[4] R.S.Sandhu. "Transaction control expressions for separation of duties". In Proceedings of the Fourth Annual Computer Security Applications Conference (ACSAC'88), Dec. 1988.
[5] W.Xu, D.Chadwick, S.Otenko. "A PKI-based Secure Audit Web Service". Proc. IASTED Int. Conf. on Communication, Network, and Information Security (CNIS 2005). Phoenix, AZ, USA.14-16 November, 2005.
[6] Johnston, W., Mudumbai, S., Thompson, M. "Authorization and Attribute Certificates for Widely Distributed Access Control." IEEE 7th Int. workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, Stanford, CA. June, 1998. Page(s): 340 -345
[7] OASIS "eXtensible Access Control Markup Language (XACML) Version 2.0". OASIS Standard, 1 Feb 2005
[8] T.T.Simon and M.E.Zurko. "Separation of duty in role-based environments". Proc. 10th Computer Security Foundations Workshop, pp.183-194. IEEE Computer Society Press, June 1997.
[9] V.D.Gligor, S.I.Gavrila, and D.Ferraiolo. "On the formal definition of separation-of-duty policies and their composition". Proc. IEEE Symp. on Research in Security and Privacy, pp.172-183, May 1998.
[10] G.-J. Ahn and R.S.Sandhu. "Role-based authorisation constraints specification". ACM Trans. on Information and System Security, 3(4):207-226, Nov.2000.
[11] D. W. Chadwick, A. Otenko, E. Ball. "Role-based access control with X.509 attribute certificates". IEEE Internet Computing, March-April 2003, pp.62-69.
[12] E. Bertino, E. Ferrari and V. Atluri. "The specification and enforcement of authorization constraints in workflow management systems". ACM Trans. on Information and System Security, Vol.2, No.1, February 1999, pp. 65-104.
[13] ITU-T Recommendation X.812 (1996) | ISO/IEC 10181-3:1996. "Information technology – Open systems interconnection – Security frameworks for open systems: Access control framework".
[14] Bertino, E., Castano, S., Farrari, E. "On specifying security policies for web documents with an XML-based language". Proc. 6th ACM Symp. Access Control Models and Technologies, ACM Press, 2001, pp. 41-52.
[15] Scott Cantor. "Shibboleth Architecture, Protocols and Profiles", Working Draft 10 September 2005.
[16] Thomas Wason. "Liberty ID-FF Architecture Overview." Version: 1.2-errata-v1.0.
[17] http://www.nmi-edit.org/releases/index.cfm#PERMIS
[18] J. Crampton, "Specifying and Enforcing Constraints in Role-Based Access Control", Proc. of the 8th ACM symp. on Access control models and Technologies (SACMAT 2003), pages 43–50, Como, Italy, June 2003.
[19] OASIS. "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard, 15 March 2005
[20] ISO 9594-8/ITU-T Rec. X.509 (2001) The Directory: Public-key and attribute certificate frameworks.

Appendix A: The MSoD policy schema

```
<?xml version="1.0" >
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
 <xs:element name="MSoDPolicySet">
  <xs:complexType>
   <xs:sequence>
    <xs:element maxOccurs="unbounded"
ref="MSoDPolicy"/>
   </xs:sequence>
  </xs:complexType>
```

```
  </xs:element>
  <xs:element name="MSoDPolicy">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="FirstStep" minOccurs="0" />
       <xs:element ref="LastStep" minOccurs="0" />
        <xs:choice>
          <xs:element maxOccurs="unbounded"
ref="MMER"/>
          <xs:element maxOccurs="unbounded"
ref="MMEP"/>
        </xs:choice>
      </xs:sequence>
      <xs:attribute name="BusinessContext"
use="required" type="xs:NCName"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="FirstStep">
    <xs:complexType>
      <xs:attribute name="operation" use="required"
type="xs:NCName"/>
      <xs:attribute name="targetURI" use="required"
type="xs:anyURI"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="LastStep">
    <xs:complexType>
      <xs:attribute name="operation" use="required"
type="xs:NCName"/>
      <xs:attribute name="targetURI" use="required"
type="xs:anyURI"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="MMER">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded"
minOccurs= “2” ref="Role"/>
      </xs:sequence>
    <xs:attribute name="ForbiddenCardinality"
use="required" type="xs:integer"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Role">
    <xs:complexType>
      <xs:attribute name="type" use="required"
type="xs:NCName"/>
      <xs:attribute name="value" use="required"
type="xs:NCName"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="MMEP">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded"
ref="Privilege"/>
      </xs:sequence>
    <xs:attribute name="ForbiddenCardinality"
use="required" type="xs:integer"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Privilege">
    <xs:complexType>
      <xs:attribute name="target" use="required"
type="xs:anyURI"/>
      <xs:attribute name="operation" use="required"
type="xs:NCName"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```