Turkish Journal of Electrical Engineering and Computer Sciences

Volume 26 | Number 5 Article 30

1-1-2018

Secure access control in multidomain environments and formal analysis of model specifications

FATEMEH NAZERIAN

HOMAYUN MOTAMENI

HOSSEIN NEMATZADEH

Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2018) 26: 2525 – 2540

doi:10.3906/elk-1802-55

Secure access control in multidomain environments and formal analysis of model specifications

Fatemeh NAZERIAN, Homayun MOTAMENI*, Hossein NEMATZADEH
Department of Computer Engineering, Sari Branch, Islamic Azad University, Sari, Iran

Received: 07.02.2018 • **Accepted/Published Online:** 08.06.2018 • **Final Version:** 28.09.2018

Abstract: Distributed multiple organizations interact with each other. If the domains employ role-based access control, one method for interaction between domains is role-mapping. However, it may violate constraints in the domains such as role hierarchy, separation of duty, and cardinality. Therefore, autonomy of the domains is lost. This paper proposes secure interoperation in multidomain environments. For this purpose, a cross-domain is created by foreign permission assignment. In an effort to maintain the autonomy of every domain, several rules are defined formally. Then, a decentralized scheme is used to provide permission mapping between domains. At the next stage, the proposed cross-domain is specified using Alloy, the first logic language. Subsequently, validity of the rules is analyzed through Alloy analyzer.

Key words: multidomain, conflict, Alloy, formal language, role-based access control

1. Introduction

With network expansion, a certain dimension can be divided into multidomains to correct management information. A large amount of information can be stored in the domains. A great deal of information is sensitive, e.g., financial information of organizations or confidential information about a patient which should be protected from unauthorized access. An effective method for data security involves access control.

One of the most widely used access control methods is role-based access control. Most organizations set roles for different tasks. Therefore, role-based access control (RBAC) is appropriate for such organizations. If every domain has an RBAC policy, one method for interoperation in multidomain environments creates a cross-domain via role mapping, where a role of a domain inherits permissions of role in other domains in the hierarchy relation between domains [1–3]. In the cross-domain created via role mapping, several constraints in the local domain may be ignored. They cause the following:

1- Role hierarchy violation: if two roles have no inheritance relation in the local domain but there is a path in hierarchy of the crossing domain between them [1, 2, 4].

2- Separation of duty (SOD) violation: SOD is divided into two parts. One is role-specific SOD constraint, where a user cannot gain access to conflict roles, and the other is user-specific SOD constraint, where a role cannot be assigned to two users. These constraints were sometimes ignored in interoperation between domains [1–6].

*Correspondence: motameni@iausari.ac.ir

3- Cardinality: in RBAC, we can consider a numerical restriction for every user or role. For example, a user u with cardinality u_n means that the number of roles simultaneously authorized for u cannot exceed u_n while role r with cardinality r_n means that the number of users simultaneously assigned to r cannot exceed r_n [1, 6].

Thus, the main contributions of this paper are as follows:

1. A permission-mapping cross-domain that solves conflicts occurring in role-mapping cross-domain is proposed.

2. Several rules are defined to maintain the autonomy and security of every domain.

3. A decentralized administrative model is considered to manage permission mapping between domains.

In this paper, a system with two domains (office staff and medical staff) is considered. In the first step, two domains interact with each other via role mapping, where conflicts such as role hierarchy, separation of duty, and cardinality occur. In the next step, foreign permission assignment is used for interoperation in multidomain environments that employ RBAC policies, so as to solve the conflicts that occur in the cross-domain created via role mapping. In other words, a cross-domain is created by foreign permission assignment and is shown how the proposed cross-domain will solve conflicts such as role hierarchy, SOD, and cardinality violations. In an effort to maintain the autonomy of local domains, three rules are defined formally: 1) not separation of duty assignment (NSODA), 2) not foreign permission assignment (NFPA), and 3) not hierarchy permission assignment (NHPA). If constraints of the local domain and the three rules (NSODA, NFPA, and NHPA) are not violated, then permission is assigned to the role of the foreign domain. Then, a decentralized scheme is explained with an algorithm to manage permission mapping between domains. The proposed cross-domain is specified with Alloy (the first-order logic language) [7]. Our motivation behind using Alloy is the automatic analysis capabilities which make Alloy applicable in analysis of system specifications [8–12].

The rest of this paper has been organized as follows: Section 2 provides the relevant literature review. Section 3 provides an overview of RBAC. In Section 4, the proposed approach is described, and Section 5 shows an example. In Section 6, the proposed approach is analyzed through Alloy for automatic verification. In Section 7, the new model is compared with the existing works in the literature. And finally, in Section 8, the paper is concluded and future work is discussed.

2. Related works

In order to prevent unauthorized access, a number of access control models have been proposed such as discretionary access control (DAC)[13], mandatory access control (MAC) [14], role-based access control (RBAC)[15, 16] and attribute-based access control (ABAC)[17]. Among these methods, RBAC is more common because it is easy to use and policy-neutral. The latter implies that RBAC supports DAC and MAC models [13, 15, 16, 18, 19]. Hence, researchers extended the RBAC model for different situations [20–22]. A survey of RBAC is available in [23]. An advantage of RBAC is self-management through RBAC. If the number of users, roles, and permissions are extremely large, then management of these components by a security administrator will be complicated. Administrative models provide a solution to this problem. ARBAC97 is the first administrative model which controls permission-to-role and role-to-user assignments by administrative role hierarchy [24]. Researchers proposed other administrative models for RBAC such as ARBAC99 [25] and ARBAC02 [26].

With the development of information processing, secure interoperation between systems has become a challenge for which certain policies should be considered. In [27], a secure interoperation model called IRBAC2000, in which a dynamic role was used for the relation between two domains, was proposed. AIRBAC2000 was proposed for dynamic role management in [28]. In [29], domain-based RBAC (D-RBAC) and access control architecture were proposed for multidomain network environments. In [30], a multiple security domain in cloud computing environment was considered while IRBAC2000 was used for cross-domain access control. Then, a centralized administrative model was considered to administer a collection of multiple hosts, router, and internet.

In the RBAC model, roles have a hierarchical structure and there are several constraints such as SOD [5] and cardinality for roles or users. In the interoperation between domains, the autonomy and security of every domain should be considered. In other words, if an access is authorized in the local domain, it is also authorized in the secure interoperation in multidomain environments, and vice versa [31]. Meanwhile, constraints of every domain are not violated but there are a few potential conflicts in the relation between domains with RBAC policies such as role hierarchy, separation of duty, and cardinality violation, the role-mapping cross-domain may violate these constraints.

Researchers presented different methods for solving the above conflicts. Some proposed a global policy that integrates the policies of local domains [2, 32, 33]. In this approach, every domain must expose its access control policies all to the mediator, while others proposed a decentralized policy or mediator-free scheme for secure interoperation because they considered the mediator as a bottleneck that could provide information of every domain. Meanwhile, domains may occasionally contain confidential information that should not be exposed [3, 27, 34].

3. Preliminaries

The National Institute of Standard and Technology (NIST) proposed NIST RBAC standard [18]. American National Standard for Information Technology (ANSI) has changed the NIST RBAC, defining a standard for RBAC [35]. According to the ANSI RBAC standard, RBAC includes 4 conceptual models: core RBAC, hierarchy RBAC, static separation of duty, and dynamic separation of duty.

Core RBAC: It includes 5 basic elements, i.e., USER (set of users), ROLE (set of roles in the system), OPS (system operations), OBS (set of objects that support the system), and SESSION (set of sessions that launch business) and the following relations:

- UA $\subseteq$ USERS $\times$ ROLES is a relation of users to roles.
- assigned_user(r: ROLES) $\rightarrow 2^{USERS}$, the mapping of role r onto a set of users. That means one role can be assigned to some users.
- PRMS=OPS $\times$ OBS is a set of permissions performing operation on object.
- PA $\subseteq$ PRMS $\times$ ROLES is a relation of permission to role.
- assigned_permissions(r: ROLES) $\rightarrow 2^{PRMS}$, the mapping of role r onto a set of permission. That means one role can have several permissions.

Role hierarchy: RH $\subseteq$ ROLE $\times$ ROLE is a partial order on ROLES called the inheritance relation indicated by $\geq$, where $r1 \geq r2$ means $r2$ permissions are also $r1$ permission and users of $r1$ are also users of $r2$.

In ANSI [35], there are two types of constraints: static separation of duty (SSD) and dynamic separation of duty (DSD). SSD is a collection of pair (rs,n), where each rs is a role set that $\mid rs \mid \geq 2$, while n is a natural number such that $2 \leq n \leq \mid rs \mid$, i.e., no user can be assigned to n roles in set rs. Similar to SSD, a DSD constraint is a collection of pair (rs,n), where each rs is a role set such that $\mid rs \mid \geq 2$ and n is a natural number such that $2 \leq n \leq \mid rs \mid$, i.e., no user may simultaneously activate n roles from set rs in one session. The SSD constraint limits roles that can be assigned to user, while a DSD constraint limits roles that can be activated by the user in one session. In [36], the authors changed the formal definition of DSD in [35].

4. Proposed approach

In this section, a method is proposed to solve the conflicts that may occur in a cross-domain created by role-mapping. First, the conflicts are shown in the role-mapping cross-domain. Second, permission-mapping cross-domain is proposed to solve the conflicts that occur in the role-mapping cross-domain. Third, rules are defined to keep the security of every domain, and finally, an algorithm is proposed for decentralized management of permission-mapping cross-domain. The conflicts that may occur in role-mapping cross-domain are as follows:

1- Role hierarchy violation: if role r_1 cannot access role r_2 in the local domain but in the hierarchy of cross-domain, there is a path between them.

2- Role-specific SOD violation: if r_1 and r_2 are in role-specific SOD constraint, then these roles cannot simultaneously be assigned to user u.

3- User-specific SOD violation: if u_1 and u_2 are in user-specific SOD constraint, then these users cannot simultaneously be authorized to role r.

4- Role-cardinality violation: if cardinality of role r is C_r, it implies that the maximum number of users simultaneously authorized to r must be less than C_r.

5- User-cardinality violation: if cardinality of user u is C_u, it implies that the maximum number of roles simultaneously assigned to u must be less than C_u.

Figure 1 illustrates the above conflict in a cross-domain created by role-mapping. As can be seen, there are two domains: the office staff (α) and the medical staff (β). There are five roles in the office staff domain (α): office manager (r_1), medical representative (r_2), accounts payable (r_3), secretary (r_4), and purchasing (r_5) and in the medical staff domain (β), there are two roles: doctor (r_6) and nurse (r_7). There are three users (denoted by ui, $i = 1, 2, 3$) in the system. Symbol r^{α} is used to represent role r in domain α. In Figure 1, r_1^{α} inherits all permissions of r_6^{β}, r_6^{β} inherits all permissions of r_2^{α}, r_7^{β} inherits all permissions of r_4^{α}, and r_5^{α} inherits all permissions of r_7^{β}. Five conflicts occur in Figure 1. These conflicts are shown in Table 1 . Two role hierarchy violations occur because r_1^{α} and r_2^{α} are not related in the local domain but in the role hierarchy of the cross-domain, r_1^{α} can access r_2^{α} via r_6^{β} . Similarly, r_5^{α} can access self-senior role, i.e., r_4^{α} via r_7^{β}. Violation of role-specific SOD occurs because $(r_2^{\alpha}, r_3^{\alpha}) \in$ role-specific SOD, but u_1 can simultaneously access r_2^{α} and r_3^{α}. u_1 can access r_3^{α} with inheritance relationship and r_2^{α} via r_6^{β}. Violation of user-specific SOD occurs because $(u_1, u_2) \in$ user-specific SOD, but u_1 and u_2 can access r_2 simultaneously (u_1 can access r_2^{α} via r_6^{β}). Role cardinality violation occurs because cardinality of r_2^{α} is one, but u_1 and u_2 can access r_2 simultaneously. User

cardinality violation occurs because cardinality of u_3 is three, but u_3 can access four roles: r_6^β, r_7^β, r_4^α, and r_2^α.

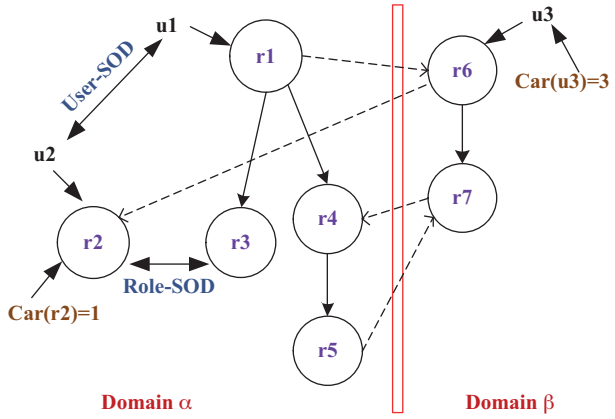

Figure 1. Cross-domain created by role mapping.

Table 1. Conflict in Figure 1.

Conflict	Description
Role hierarchy	There are two conflicts; 1- r_1^α inherits r_2^α via r_6^β while r_1^α and r_2^α are not related in the local domain. 2- r_5^α inherits self-senior role, i.e, r_4^α via r_7^β
Role-specific SOD	There is a conflict because (r_2^α, r_3^α) are in role-specific SOD but u_1 can access r_2^α via $r_6^\beta$and can access r_3^α via r_1^α.
User-specific SOD	There is a conflict because (u_1, u_2) are in user-specific SOD but u_1 can access r_2^α via r_6^β while u_2 access r_2^α too.
Role cardinality	There is a conflict because Car(r_2^α)=1 but u_1 and u_2 simultaneously authorize r_2^α.
User cardinality	There is a conflict because Car(u_3)=3 but 4 roles r_6^β, r_7^β, $r_4^\alpha$and r_2^α are assigned to u_3.

4.1. Permission-mapping cross-domain

For solving the above conflicts, a foreign permission assignment where every role in local domain can inherit a few permissions of role in another domain and cannot inherit roles of another domain is proposed. Figure 2 shows how a cross-domain created by foreign permission assignment can solve the conflicts that may occur in a cross-domain created via role mapping. The number of domains, roles, and constraints (SOD, cardinality) in Figure 2 is the same as those in Figure 1 but in Figure 2, two domains (α and β) interact with each other via permission assignment. According to the RBAC policy, every role has one or more permissions (Section 3) and can inherit several permissions of a role in another domain. In Figure 2, r_1^α inherits only p_{20} of r_6^β and cannot access all permissions of r_6^β. In the same way, r_6^β can access only p_5 of r_2^α, r_5^α can access only p_{24} of r_7^β, and r_7^β can access only p_8 of r_4^α. The conflicts in the previous section are solved in our approach. The reason is displayed in Table 2. Role hierarchy violation cannot occur because r_1^α can access only p_{20} of r_6^β; thus, r_1^α cannot access permissions of r_2^α and also r_5^α inherits only p_{24} of r_7^β and cannot access permissions of self-senior role r_4^α. Role-specific SOD violation cannot occur because in Figure 2, (r_2^α, r_3^α) $\in$ role-specific SOD and u_1 authorizes only r_3^α and cannot access r_2^α via r_6^β. User-specific SOD cannot occur because (u_1, u_2) are in user-specific SOD (Figure 2) and u_1 and u_2 cannot access the same roles. Role cardinality violation

cannot occur in Figure 2 because cardinality r_2^{α} is one and only accessible to u_2. User cardinality violation cannot occur in Figure 2 because cardinality of u_3 is three and u_3 can access two roles, r_6^{β} and r_7^{β}.

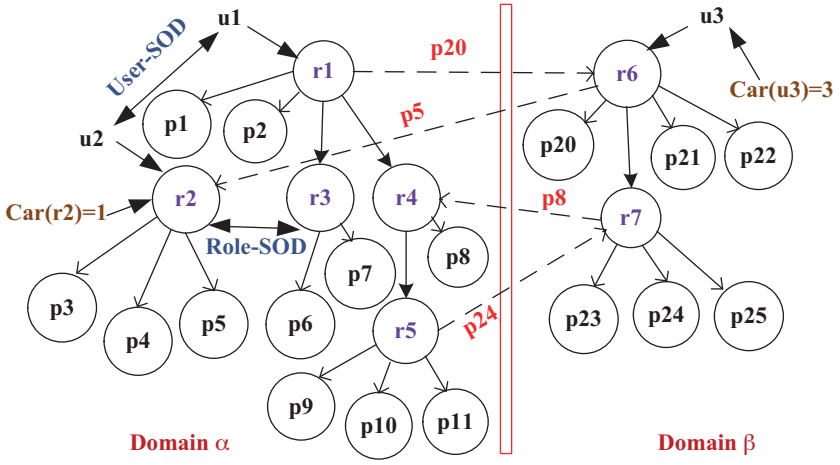

Figure 2. Cross-domain created by foreign permission assignment.

Table 2. Overview of conflicts in Figure 2.

Conflict	Description
Role hierarchy	There is no role hierarchy conflict because r_1^{α} cannot access all permissions of r_6^{β} and cannot access permissions of r_2^{α}. Meanwhile r_5^{α} inherits only p_{24} of r_7^{β} and cannot access permissions of r_4^{α}.
Role-specific SOD	There is no role-specific SOD conflict because (r_2^{α}, r_3^{α}) are in role-specific SOD and u_1 authorizes only r_3^{α} and cannot access r_2^{α} via r_6^{β}.
User-specific SOD	There is no user-specific SOD conflict because (u_1,u_2) are in user-specific SOD and u_1 cannot access r_2^{α} via r_6^{β} as u_2.
Role cardinality	There is no role cardinality conflict because Car(r_2^{α})=1 and u_2 authorizes only r_2^{α}.
User cardinality	There is no user cardinality conflict because Car(u_3)=3 and u_3 access two roles r_6^{β} and r_7^{β}.

4.2. Cross-domain rules

In this section, a number of restrictions are considered for the newly proposed cross-domain, where the autonomy of every domain is maintained and called cross-domain rules. These rules are formulated in a formal language. For this purpose, two functions namely *Irequest_permission* and *Iassign_permission* are introduced as follows:

Irequest_permission: This function returns foreign permissions requested by the role.

Irequest_permission $\subseteq ROLE1^{DOMAIN1} \times ROLE2^{DOMAIN2}$

This relation returns permissions that ROLE1 in DOMAIN1 requested from ROLE2 in DOMAIN2.

Iassign_permission: This function returns foreign permissions assigned to the role.

Iassign_permission $\subseteq ROLE1^{DOMAIN1} \times ROLE2^{DOMAIN2}$

This relation returns permissions assigned to ROLE1 in DOMAIN1 by ROLE2 in DOMAIN2.

Now, using these functions, three rules are defined in the following subsections: 1) not separation of duty assignment (NSODA), 2) not foreign permission assignment (NFPA), and 3) not hierarchy permission assignment (NHPA).

4.2.1. Not separation of duty assignment (NSODA)

If two roles $(r_1,\ r_2)$ are in role-specific SOD constraint, then permissions of r_1 and r_2 cannot be assigned to a role in other domains. For further explanation, consider that r_6^{β} requests p_6 of r_3^{α} in Figure 2. This request is invalid because r_6^{β} has permission p_5 of r_2^{α} and cannot request permission of r_3^{α} because $(r_2,\ r_3)$ are in role-specific SOD constraint. The formal description of this constraint is as follows:

$\forall i \in N \wedge \forall d_1, d_2 \in DOMAINS \wedge \forall r_1, r_2, r_3, r_i \in ROLES \wedge$

$r_1, r_2 \in d_1 \wedge r_3, r_i \in d_2 \wedge (r_1, r_2) \in SOD \bullet$

$(Irequest\_permission(r_3^{d_2}, r_1^{d_1}) \cap Iassign\_permission(r_3^{d_2}, r_2^{d_1})) = \varnothing)$

$\wedge(\forall r_i \mid r_3 \geq r_i \vee \forall r_i \mid r_i \geq r_3) \Rightarrow$

$(Irequest\_permission(r_3^{d_2}, r_1^{d_1}) \cap Iassign\_permission(r_i^{d_2}, r_2^{d_1})) = \varnothing).$

In this definition, $(r_1,\ r_2)$ are in SOD constraint. Hence, foreign role r_3 cannot request permission of r_1 while it has a permission of r_2. If r_3 has a junior or senior role r_i, and r_i has a permission of $r_2^{d_1}$, then r_3 cannot request permission of $r_1^{d_1}$. In Figure 2, for example, r_7^{β} has a senior role r_6^{β}. Thus, if r_7^{β} has a permission of r_3^{α}, then r_6^{β} cannot access to permission of r_2^{α} because $(r_3^{\alpha},\ r_2^{\alpha})$ are in role-specific SOD constraint.

4.2.2. Not foreign permission assignment (NFPA)

In this rule, a role in a local domain cannot request foreign permissions assigned to a role of another domain. To gain a better understanding of NFPA, consider Figure 2. r_7^{β} requests p_8 of r_4^{α} but r_5^{α} cannot request p_8 of r_7^{β}. Under this restriction, r_5^{α} cannot access the permissions of self-senior role r_4^{α}. If the number of domains is greater than 2, NFPA, indicates a violation of hierarchy permission request between domains. Therefore, every domain must request permission of another domain directly. The formal definition of NFPA is as follows.

$\forall i \in N \wedge \forall d_1, d_2, d_i \in DOMAINS \wedge \forall r_1, r_2, r_i \in ROLES \wedge$

$r_1 \in d_1 \wedge r_2 \in d_2 \wedge r_i \in d_i \wedge d_i \neq d_2 \bullet$

$(Irequest\_permission(r_1^{d_1}, r_2^{d_2}) \cap Iassign\_permission(r_2^{d_2}, r_i^{d_i})) = \varnothing)$

In formal definition, d_i is the third domain and permissions of d_i assigned to role r_2 in domain d_2 cannot be assigned to a role of another domain.

4.2.3. Not hierarchy permission assignment (NHPA)

In this rule, a role in a local domain cannot request any inherited permissions of roles in other domains. In Figure 2, r_1^{α} inherits permissions of r_3^{α} but r_6^{β} cannot request p_6 of r_1^{α} because p_5 of r_2^{α} has been assigned to r_6^{β} and if p_6 is assigned to r_6^{β} by r_1^{α} then r_6^{β} can access permissions of r_2 and r_3 in domain α. Nonetheless, $(r_2,\ r_3)$ are in SOD constraint and permissions of these roles cannot be assigned to a role in another domain. The formal definition of NHPA is as follows:

$\forall r_1, r_2, r_3 \in ROLES \wedge \forall d_1, d_2 \in DOMAINS \bullet$

$r_1, r_2 \in d_1 \wedge r_3 \in d_2 \wedge (r_1 \geq r_2) \Rightarrow (Irequest\_permission(r_3^{d_2}, r_1^{d_1}) \cap assign\_permission(r_2)) = \varnothing).$

According to Section 3, $assign\_permission(r_2)$ returns permissions assigned to r_2, and $r_1 \geq r_2$ means that r_1 is the senior role of r_2. Thus, role $r_3^{d_2}$ cannot request a permission of $r_1^{d_1}$ inherited from $r_2^{d_1}$.

4.3. Decentralized administration

Since every domain has RBAC policy, a foreign permission assignment can be controlled by administrative role of every domain. Administrative role of every domain receives the requested permission of other domains and then verifies the rules NSODA, NFPA, and NHPA. If these rules are satisfied, then the permission is assigned to the role of the requested domain.

Algorithm is proposed to check the foreign permission request. Consider role r_1 in domain d_1 requests permission p_k of role r_2 in domain d_2, then administrative role of domain d_2 should check NSODA, NFPA, and NHPA rules. For checking the NSODA rule, the administrative role of domain d_2 checks the SOD constraint in domain d_2. If r_1 has a permission of r_j in domain d_2 and (r_j, r_2) is in SOD constraint of domain d_2, then this request is invalid. If role r_x of domain d_1 has a permission of r_j in domain d_2 and (r_j, r_2) is in SOD constraint of domain d_2, then administrative role of domain d_2 sends a message to the administrative role of domain d_1 to inform that role r_x is not junior or senior of role r_1 in role hierarchy of domain d_1. If r_x is a junior or senior of role r_1 then NSODA rule is not satisfied and this request is invalid. For NFPA rule checking, the administrative role of domain d_2 checks that p_k is not a foreign permission of domain d_2, otherwise this request is invalid. As for NHPA, the administrative role of domain d_2 checks that p_k is not a hierarchy permission of domain d_2; otherwise, this request is invalid.

Algorithm. Check permission request

1: Role r_1 of domain d_1 requests permission p_k of role r_2 in domain d_2.
2: Administrative role of domain d_2 checks NSODA rule.
3: For every (r_j,r_2) in SOD constraint in domain d_2
4: Begin
5: If r_1 has a permission of r_j
6: Return invalid request and exit.
7: If r_x of domain d_1 has a permission of r_j in domain d_2
8. begin
9: d_2 Sends a message d_1.
10: d_1 checks hierarchy relation and returns answer.
11. end
12: If there is a relation between r_1 and r_x
13: Return invalid request and exit.
14: End
15: Administrative role of domain d_2 checks NFPA rule.
16: If p_k is foreign permission of r_2
17: Return invalid request and exit.
18: Administrative role of domain d_2 checks NHPA rule.
19: If p_k is hierarchy permission of r_2
20: Return invalid request and exit.

4.3.1. Complexity analysis

In the above algorithm, every domain uses an administrative role to check the permission request. Consider two domains, d_1 and d_2. Domain d_1 has n_1 roles and p_1 permissions, whereas domain d_2 has n_2 roles and

p_2 permissions. Role r_1 in domain d_1 requests p_k of role r_2 in domain d_2. The administrative role of domain d_2 should check three rules; NSODA, NFPA, and NHPA. We calculate time complexity for domain d_2.

1. NSODA checking (lines 4–14): The maximum SOD constraint in domain d_2 that includes r_2 is (n_2-1). For every (n_2-1) pair, requester role r_1 should not have a permission of (n_2-1) roles in SOD constraint. Assume that pr is the maximum number of permission that should be checked (lines 5 and 6).

   If there is a role in domain d_1 with a permission of (n_2-1) roles in SOD constraint, then the administrative role of domain d_1 should check the hierarchy role related to it. Consider constant C for this purpose (lines 7–13). Thus, time complexity to check NSODA is $\sum_{i=1}^{n_2-1}(pr+C)$.

2. NFPA checking (lines 15–17): Pf represents the total foreign permissions in domain d_2.

3. NHPA checking (lines 18–20): Ph represents the hierarchy permissions in domain d_2.

Then, the time complexity for domain d_2 is as follows:

$$t(n_2,p_2) = (\sum_{i=1}^{n_2-1}(pr+C)) + pf + ph \leq (\sum_{i=1}^{n_2-1}(p_2+C)) + p_2 + p_2\,,$$

$$t(n_2,p_2) = (n_2-1)(p_2+C) + 2p_2\,,$$

$$t(n_2,p_2) = (n_2+1)p_2 + (n_2-1)C = n_2p_2 + n_2C + p_2 - C\,.$$

P_2 is the total permissions of domain d_2 and is greater than pr, pf, and ph. Thus, time complexity for domain d_2 is $t(n_2,p_2) = O(n_2 \times p_2)$. In the newly proposed approach, a decentralized scheme is used and interaction between domains is managed by itself. The time complexity of every domain is $O(np)$, where n is the number of roles and p is the number of permissions. If i domains interact with each other and domain d_1 has n_1 roles and p_1 permissions, domain d_2 has n_2 roles and p_2 permissions and so on, then the time complexity of the interaction between domains is $O(max(n_1p_1, n_2p_2, ..., n_ip_i))$.

In the centralized scheme, there is a mediator that gathers the policy of every domain. Therefore, the centralized scheme tends to be space-consuming. If i domains interact with each other, the mediator should collect all policies of these domains. The time complexity is $O(n_1n_2..n_i \times p_1p_2..p_i)$, i.e., time-consuming.

5. An illustrative example

Since every domain has RBAC policy, a foreign permission assignment can be controlled by the administrative role of every domain. Consider Figure 2. For domain α, $r_1 - r_5$ are organization roles while ar^α is considered an administrative role. For domain β, r_6 and r_7 are organization roles and an administrative role ar^β is considered. The administrative role of every domain includes a related cross-domain (what permissions are assigned to roles of other domains), SOD constraint, PA (permission directly assigned to the role), HR (hierarchy role), and FPA (foreign permission assignment). For every domain, SOD constraint has been provided in Table 3, domain β does not have SOD constraint and therefore is displayed empty. Cross-domain is shown in Table 4. Cross-domain (CD) of domain d_n is the sum of permissions of another domain assigned to roles in domain d_n. A foreign permission assignment is in the form $r_i^{d_m} \triangleright p_k^{r_j^{d_n}}$, in which permission p_k of role r_j in domain d_n is assigned to role r_i in domain d_m. In domain α for example, $r_6^\beta \triangleright p_5^{r_2^\alpha}, r_7^\beta \triangleright p_8^{r_4^\alpha}$ means p_5 of role r_2 in domain α is assigned to role r_6 of domain β while p_8 of role r_4 in domain α is assigned to role r_7 of domain

β. Permission assignment (PA), role hierarchy (HR), and foreign permission assignment (FPA) can be seen in Table 5.

Table 3. SOD constraint for every domain.

Domain	SOD constraint
α	$\{(r_2, r_3)\}$
β	$\{\}$

Table 4. Cross-domain for every domain.

Domain	Cross-domain
α	$r_6^{\beta} \triangleright p_5^{r_2^{\alpha}}, r_7^{\beta} \triangleright p_8^{r_4^{\alpha}}$
β	$r_1^{\alpha} \triangleright p_{20}^{r_6^{\beta}}, r_5^{\alpha} \triangleright p_{24}^{r_6^{\beta}}$

Table 5. Permission assignment and role hierarchy for every domain.

Domain	Role	PA	HR	FPA
α	r_1	p_1, p_2	$r_1 \geq r_3, r_1 \geq r_4$	p_{20}
	r_2	p_3, p_4, p_5		
	r_3	p_6, p_7	$r_1 \geq r_3$	
	r_4	p_8	$r_4 \geq r_5, r_1 \geq r_4$	
	r_5	p_9, p_{10}, p_{11}		p_{24}
β	r_6	p_{20}, p_{21}, p_{22}	$r_6 \geq r_7$	P_5
	r_7	p_{23}, p_{24}, p_{25}	$r_6 \geq r_7$	P_8

When a foreign role requests a permission of a local domain, the administrative role of the local domain checks the rules (NSODA, NHPA, and NFPA) using SOD constraint in Table 3, cross-domain in Table 4, and permission assignment (PA), role hierarchy (HR), and foreign permission assignment (FPA) in Table 5. If the requested permission does not violate these three rules (NSODA, NFPA, NHPA), then permission is assigned to the foreign role.

Example 1 *In Figure 2, if r_6^{β} requests p_6 of r_3^{α}, then the administrative role of domain α verifies the NSODA rules for domain α. For this purpose, the administrative role of domain α checks the SOD constraint in Table 3 and informs that (r_2, r_3) are in SOD constraint. Then, it checks the cross-domain of α in Table 4 while informing that r_6^{β} has a permission of r_2^{α}, indicating that the request is invalid.*

Example 2 *In Figure 2, if r_7^{β} requests p_7 of r_3^{α}, the administrative role of domain α checks Table 3 for domain α and informs that (r_2, r_3) are in SOD constraint. According to domain α of Table 4, r_7^{β} does not have a permission of r_2^{α} but r_6^{β} has a permission of r_2^{α}. Thus, the administrative role of domain α sends a message to the administrative role of domain β, asking whether or not r_6^{β} and r_7^{β} have a hierarchy relation. The administrative role of domain β verifies Table 5 for domain β informing that there is a hierarchy relation*

between r_6^β *and* r_7^β*. It therefore returns True to domain* α*, and then administrative role of domain* α *announces that request* p_7 *by* r_7^β *is invalid because the senior role of* r_7^β *signifies that* r_6^β *accesses* p_5 *of* r_2^α*. Finally, this assignment violates the NSODA rule.*

Example 3 *In Figure 2, if* r_6^β *requests* p_6 *of* r_1^α*, the administrative role of domain* α *checks Table 3 for NSODA rule and informs that the NSODA rule is valid because* r_1^α *is not in SOD constraint of domain* α*. Then, it verifies Table 4 for checking the NFPA rule, finding out that* p_6 *is not a foreign permission of* r_1^α*, rendering the NFPA valid. Then, it checks the NHPA rule by verifying Table 5 and informing that* p_6 *is a hierarchy permission of* r_1^α*, thus indicating this request is invalid.*

Example 4 *Example 4: In Figure 2, if* r_5^α *requests* p_8 *of* r_7^β*, the administrative role of domain* β *according to Table 3 does not have SOD constraint. Hence, the NSODA rule is valid. Then, it checks Table 5 for the NFPA rule of domain* β *and finds out that* p_8 *is not a direct permission of* r_7^β *and is a foreign permission of* r_7^β*. This request is thus indicated to be invalid because it cannot satisfy the NFPA rule.*

6. Analysis of the newly proposed cross-domain using Alloy

Alloy is a high-level language and a structural modeling tool based on the first-order logic. It has a mathematical base under the influence of Z notation [7]. It uses the Alloy analyzer to create a micromodel yielding automatic model validation and consists of several elements including *signatures (sig), fields, facts, assertions (assert)*, and *predicates (pred)*. In Alloy *signatures, (sig)* denotes set of atoms. Atoms are uninterpreted and immutable, consisting of fields to represent a relation between signatures; *facts* are used to define model constraints; *assertions (assert)* check if the model satisfies the desired behavior. And *predicates (pred)* perform any required operation in the model. Alloy has two commands: *run* and *check*. *run* produces instants of a model and *check* violates the instruction if a counterexample is found.

Employing Alloy, this section specifies the formal specifications of the proposed approach and the three rules (NSODA, NFPA, and NHPA). Then, the validity of the rules are analyzed using the Alloy analyzer. Entities such as domain, role, and permission are considered as *signatures (sig)*. The proposed approach and cross-domain rules are implemented in Alloy in Figures 3 and 4, respectively.

6.1. Analysis of the cross-domain rules

To automatically analyze, Figure 2 is modeled in Alloy. In the following subsections, we describe the analysis of NSODA, NFPA, and NHPA in the Alloy analyzer.

6.1.1. NSODA rule analysis

To check the NSODA rule, two predicates, *pred exist_f_p* and *pred exist_h_p*, (Figure 4) are used. *Pred exist_f_p* verifies whether or not role r has foreign permission. If role r has foreign permission, then we use *pred nsoda_check* to check the NSODA rule. *Pred exist_h_p* verifies whether or not role r has hierarchy permission. If role r has heredity role, then we use *pred nsodah_check* to check the NSODA rule. With the instruction *check{}*, we can verify the validity of *predicate*.

```
sig domains{}
sig permissions{}
sig roles{da:set domains,
pa:set permissions,
fpa:set permissions}
one sig sod{sd:set roles}
one sig hierarchy{rh:roles->roles}
fact sod_fact{all rsod:sod|#rsod.sd=2}
fact fpa_fact{all r:roles,p:permissions |
p in r.fpa implies no ((pa.p).da & r.da)}
fact hierarchy{
all r1,r2:roles|r1 in r2.^(hierarchy.rh)
implies r2 not in  r1.^(hierarchy.rh)
no r:roles|r in r.^(hierarchy.rh)}
```

Figure 3. Conceptual model in Alloy.

```
fun roleinsod[risod:sod]:set roles
{risod.sd}
pred exist_f_p[r:roles]
{some p1:permissions|p1 in r.fpa}
pred exist_h_r[r:roles]
{some r3:roles| r3 in r.^(hierarchy.rh)}
pred nsoda_check[r:roles,p:permissions]
{some r1:roles|r1 in pa.p and no sd1:sod |
( r1+ pa.(r.fpa)) in roleinsod[sd1]
and r1 not in pa.(r.fpa) }
pred nsodah_check[r:roles,p:permissions]
{(some r1,r2:roles|r1 in pa.p and r2 in r.^
(hierarchy.rh) and no sd1:sod |
( r1+ pa.(r2.fpa)) in roleinsod[sd1]
and r1 not in pa.(r2.fpa))or
(some r1,r2:roles|r1 in pa.p and r2 in
(hierarchy.rh).r and no sd1:sod |
( r1+ pa.(r2.fpa)) in roleinsod[sd1]
and r1 not in pa.(r2.fpa))}
pred nfpa_nhpa_check[r1:roles,
p:permissions,r 2:roles]
{r1.da!=r2.da and p  in r2.pa }
```

Figure 4. Cross-domain rules modeled in Alloy.

Example 5 *In Figure 2, Role r_6 has foreign permission p_5 of role r_2. With regard to check{nsoda_check[r_6,p_7]}, NSODA rule is verified for pair (r_6,p_7) in Figure 2. The result shows that a counterexample has been found. This request is invalid because role r_6 has foreign permission p_5 of role r_2, p_7 is permission of role r_3, and (r_2, r_3) are in SOD constraint. Thus, r_6 cannot have a permission of both roles. If pred nsoda_check is verified for pair (r_6, p_{10}), the result shows that there is no counterexample. Hence, this mapping does not violate the NSODA rule.*

Example 6 *In Figure 2, role r_6 has hierarchy role (r_7). Thus, check{nsodah_check[r_7,p_6]} checks the NSODA rule for pair (r_7,p_6). The result shows that a counterexample has been found and this request is invalid because r_6 has permission p_5 of role r_2, p_6 is permission of role r_3 and (r_2, r_3) are in SOD constraint.*

Example 7 *If pred nsodah_check is verified for pair (r_7, p_{10}), the result indicates that there is no counterexample. Hence, this mapping does not violate the NSODA rule. If role r does not have foreign permission and heredity role, then foreign permission can be assigned to the role.*

6.1.2. NFPA and NHPA rule analysis

To check the NFPA and NHPA rules, *pred nfpa_nhpa_check* in Figure 4 is used.

Example 8 *check{nfpa_nhpa_check[r_5,p_{20},r_6]} is used to verify NFPA and NHPA rules for Triple (r_5,p_{20},r_6), which means r_5 request p_{20} of r_6. The result of execution shows that there is no counterexample and this mapping is valid because p_{20} is a direct permission of r_6 (neither a foreign permission nor a hierarchy permission).*

Example 9 *If pred nfpa_nhpa_check is verified for Triple (r_5, p_8, r_7), then the result shows that the mapping is invalid because p_8 is foreign permission of r_7, and r_5 cannot request this permission of r_7.*

Example 10 *If pred nfpa_nhpa_check is verified for Triple* (r_5, p_{25}, r_6)*, then the result shows that this mapping is invalid because* p_{25} *is hierarchy permission of* r_6*, and* r_5 *cannot request this permission of* r_6*.*

7. Discussion

In an attempt to provide secure interopeartion between domains, this paper employed a permission-mapping cross-domain to solve conflicts such as role hierarchy, role-specific SOD, user-specific SOD, role cardinality, and user cardinality, which occur in role-mapping cross-domain. Moreover, the proposed method uses a decentralized administrative model to manage permission mapping. In this section, the proposed approach has been compared with those developed by other researchers in the relevant literature. First, we review which constraints (role hierarchy, role-specific SOD, user-specific SOD, role cardinality, and user cardinality) were investigated and solved by which researches as shown in the first two columns of Table 6. The third column of Table 6 shows the method used for creating cross-domain. In [3], researchers proposed a framework for secure interoperation in multidomain environments where role mapping and direct permission assignment were employed to create cross-domains. Direct permission assignment was used when role mapping led to unauthorized access. Thus, two classes of rules should be considered: 1) role mapping and 2) permission mapping. Park and Sandhu [37] presented the concept of usage control (UCON). Moreover, Jianfeng et al.[4] presented a framework for secure interoperation using UCON, where foreign subject attributes were mapped to local attributes, while formulating several theorems and definitions to verify SOD and cyclic hierarchy constraints. When the system is extremely large, the number of attributes will make it difficult to interpret them between domains.

There are two methods for managing the interaction between domains: centralized and decentralized. In the centralized method, the policies of local domains are integrated by a trusted third party. It should be aware of local domain policies. This will lead to security breaches because domains may sometimes contain confidential information that should not be exposed. Decentralized policy is a mediator-free scheme to maintain the security of every domain. In this paper, decentralized method is used, while the administrative role of every domain verifies permission assignment between domains. If it does not violate the constraints of any domain,

Table 6. Comparison with existing works.

Approach	Constraint	Cross-domain	Management	Tools used
Shafiq et al. [2]	1-Role hierarchy 2-User-specific SOD 3-Role-specific SOD	Role mapping	Centralized	None
Huang and Krichner [1]	1-Role hierarchy 2-User-specific SOD 3-Role-specific SOD 4- Role cardinality 5- User cardinality	Role mapping	Centralized	CPN
Jinwei et al. [3]	1-Role hierarchy 2-User-specific SOD	Role mapping and permission mapping	Decentralized	None
Lin et al. [30]	None	Role mapping	Centralized	CloudSim
This paper	1-Role hierarchy 2-User-specific SOD 3-Role-specific SOD 4- Role cardinality 5- User cardinality	Permission mapping	Decentralized	Alloy

then permission of foreign role is assigned to a local role. In this regard, the forth column displays the policy for interaction between domains.

The last column shows the tools used. In [1], CPN was used to solve cardinality and SOD constraint violation. However, if the number of roles and domains are extremely large, then the CPN model will be complicated and analysis will be extremely difficult and time-consuming.

8. Conclusions and future work

This paper intended to provide a secure interoperation in multidomain environments. For this purpose, a foreign permission assignment was used to create a cross-domain in multidomain environments. The proposed cross-domain solves conflicts such as role hierarchy, separation of duty, and cardinality found in a cross-domain created by role mapping. At the next stage, autonomy of every domain was maintained by formally defining three rules namely NSODA, NFPA, and NHPA. In this paper, every domain has RBAC policy. Therefore, decentralized administrative was used to decide which foreign permission assignment is valid, i.e. does not violate the cross-domain rules. Then, the proposed approach was illustrated through an example. Finally, formal specifications of the model were implemented by Alloy to verify the correctness of the defined policies in a formal language.

In future studies, we can focus on finding other policies that support hierarchy permission assignment between domains so that the domains can be related to each other via interface domain. Furthermore, future papers can cover temporal and partial resource sharing in secure interoperation in multidomains. Another idea involves creating a secure interoperation in multidomain environments through attribute-based access control policies.

References

[1] Huang H, Krichner H. Secure interoperation design in multi-domains environments based on colored Petri nets. Inform Sciences 2013; 221: 591-606.

[2] Shafiq B, Joshi JBD, Bertino E, Ghafoor A. Secure interoperation in a multidomain environment employing RBAC policies. IEEE T Knowl Data En 2005; 17: 1557-1577.

[3] Hu J, Li R, Lu Z. Establishing RBAC-based secure interoperability in decentralized multidomain environments. In: Proceedings of 10th International Conference on Information Security and Cryptography; 29–30 November 2007; Seoul, Korea. pp. 49-63.

[4] Lu J, Li R, Varadharajan V, Lu Z, Ma X. Secure interoperation in multidomain environment employing UCON policies. In: International Conference on Information security; 2009; Springer, Heidelberg. pp. 395-402.

[5] Ultra JD, Pancho-Festin S. A simple model of separation of duty for access control models. Comput Secur 2017; 68: 69-80.

[6] Bijon KZ, Krishnan R, Sandhu R. Toward an attribute based constraints specification language. In: International Conference on Social Computing; 8–14 September 2013; Alexandria, VA, USA. pp. 108-113.

[7] Jackson D. Software Abstraction: Logic, Language, and Analysis. Cambridge, MA, USA: MIT Press, 2006.

[8] Giammarco K. A formal method for assessing architecture model and design maturity using domain-independent patterns. Procedia Comput Sci 2014; 28: 555-564.

[9] Cunha A, Garis A, Riesco D. Translating between Alloy specification and UML class diagrams annotate with OCL. Softw Syst Model 2015; 14: 5-25.

[10] Schaad A, D.Moffett J. A lightweight approach to specification and analysis of role-based access control extensions. In: Proceedings of the 7th ACM Symposium on Access Control Models and Technologies; 3–4 June 2002; Monterey, CA, USA. pp. 13-22.

[11] Jha S, Sural S, Vaidya J, Atluri V. Security analysis of temporal RBAC under an administrative model. Comput Secur 2014; 46: 154-172.

[12] Massoni T, Gheyi R, Borba P. A UML class diagram analyzer. In: 3rd International Workshop on Critical System Development with UML; January 2004; Lisbon, Portugal. pp. 100-114.

[13] Osborn S, Sandhu R, Munawer Q. Configuring role-based access control to enforce mandatory and discretionary access control policies. ACM T Inform Syst Se 2000; 3: 85-106.

[14] Osborn S. Mandatory access control and role-based access control revisited. In: RBAC '97 Proceedings of the second ACM workshop on role-based access control; 6-7 November 1997; ACM New York, NY, USA. pp. 31-40.

[15] Sandhu R, Coyne EJ, Feinstein HL, Youman CE. Role-based access control models. IEEE Comput Soc 1996; 29: 38-47. doi.10.1109/2.485845.

[16] Bertino E. RBAC models – concepts and trend. Comput Secur 2003; 22: 511-514.

[17] Jin X, Krishnan R, Sandhu R. A unified attribute-based access control model covering DAC, MAC and RBAC. In: Proceedings of the 26th Aunal IFIP WG 11.3 conference on Data and Application Security and Privacy; 11–13 July 2012; Paris, France. pp. 41-55.

[18] Ferraiolo DF, Sandhu R, Gavrila S, Kuhn DR, Chandramouli R. Proposed NIST standard for role-based access control. ACM T Inform Syst Se 2001; 4: 224-74.

[19] Uğur A, Soğukpınar İ. Multilayer authorization model and analysis of authorization method. Turk J Electr Eng Co 2016; 24: 4915-4934. doi:10.3906/elk-1403-200.

[20] Bertino E, Bonatti PA, Ferrari E. TRBAC: a temporal role-based access control model. ACM T Inform Syst Se 2001; 4: 191-233. doi.10.1145/5019 78.501979.

[21] Ferreira A, Chadwick D, Farinha P, Correia R, Zao G, Chilro R, Antunes L. How to security break into RBAC: the BTG-RBAC model. In: Annual Computer Security Applications Conference; 7–11 December 2009; Honolulu, HI, USA. pp. 23-31. doi.10.1109/ACSAC.2009.12.

[22] Liu G, Zhang R, Song H, Wang C, Liu J. Ts-RBAC: A RBAC model with transformation. Comput Secur 2016; 60: 52-61.

[23] Fuchs L, Pernul G, Sandhu R. Roles in information security – a survey and classification of the research area. Comput Secur 2011; 30: 748-769.

[24] Sandhu R, Bhamidipati V, Munawer Q. The ARBAC97 model for role-based administration of roles. ACM T Inform Syst Se 1996; 2: 105-135.

[25] Sandhu R, Munawer Q. The ARBAC99 model for administration of roles. In: Proceeding 15th Annual computer security applications conference; 6–10 December 1999; Phoenix, AZ, USA. pp. 229-38.

[26] Oh S, Sandhu R. A model for role administration using organization structure. In: Proceedings of the 7th ACM Symposium on Access Control Models and Technologies; 3–4 June 2002; Monterey, California, USA. pp. 155-62.

[27] Kapadia A, Al-Muhtadi J, Campbell RH, Mickunas D. IRBAC 2000: Secure interoperability using dynamic role translation. In: Proceedings of the 1st International Conference on Internet Computing; 26-29 June 2000; Las Vegas, NV, USA. pp. 231-238.

[28] Al-Muhtadi J, Kapadia A, Campbell R, Mickunas D. The A-IRBAC2000 model: administrative interoperable role-base access control. ACM T Inform Syst Se 2001; 3: 173-182.

[29] Yang Z, Wang J, Yang L, Yang R, Kou B, Chen J, Yang S. The RBAC model and implementation architecture in multi-domain environment. Electron Commer Res 2013; 13: 273-289.

[30] Lin G, Bie Y, Lei M. Trust based access control policy in multi-domain of cloud computing. J Comput 2013; 8: 1357-1365.

[31] Gong L, Qian X. Computational issues in secure interoperation. IEEE T Software Eng 1996; 22: 43-52. doi. 10.1109/32.481533.

[32] Zhu H, Duan S, Hong F, Lu K. An access-control policy based on sharing resource management for a multi-domains environment. In: Proceedings of the Third International conference on Autonomic and Trusted Computing; 3–6 September 2006; Wuhan, China. pp. 439-448.

[33] Piromruen S, Joshi JBD. An RBAC framework for time constrained secure interoperation in multi-domain environments. In: Proceeding of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems; 2–4 Feb 2005; Sedona, AZ, USA. pp. 36-45.

[34] Wang X, Feng D, Xu Z, Hu H. Mediator-free secure policy interoperation of exclusively-trusted multiple domain. In: Proceedings of the International Conference on Information Security Practice and Experience; 21–23 April 2008; Sydney, Australia. pp. 248-262.

[35] ANSI, American National standard for Information Technology "Role-Based Access Control". American National Standard Institute, 2004.

[36] Esna-Ashari M, Rabiee HR, MirianHosseinabadi SH. Reliability of separation of duty in ANSI standard role-based access control. Sci Iran 2011; 18: 1416-1424.

[37] Park J, Sandhu R. The UCONABC usage control model. ACM T Inform Syst Se 2004; 7: 128-174. doi.10.1145/984334.984339.