Journal of Computer Security 22 (2014) 961–996
DOI 10.3233/JCS-140510
IOS Press

Security analysis for temporal role based access control

Emre Uzun [a], Vijayalakshmi Atluri [a], Jaideep Vaidya [a,*], Shamik Sural [b], Anna Lisa Ferrara [c], Gennaro Parlato [d] and P. Madhusudan [e]

[a] *Rutgers University, Newark, NJ, USA*
[b] *Indian Institute of Technology, Kharagpur, India*
[c] *University of Bristol, Bristol, UK*
[d] *University of Southampton, Southampton, UK*
[e] *University of Illinois at Urbana-Champaign, IL, USA*

Providing restrictive and secure access to resources is a challenging and socially important problem. Among the many formal security models, Role Based Access Control (RBAC) has become the norm in many of today's organizations for enforcing security. For every model, it is necessary to analyze and prove that the corresponding system is secure. Such analysis helps understand the implications of security policies and helps organizations gain confidence on the control they have on resources while providing access, and devise and maintain policies.

In this paper, we consider security analysis for the Temporal RBAC (TRBAC), one of the extensions of RBAC. The TRBAC considered in this paper allows temporal restrictions on roles themselves, user-permission assignments (UA), permission-role assignments (PA), as well as role hierarchies (RH). Towards this end, we first propose a suitable administrative model that governs changes to temporal policies. Then we propose our security analysis strategy, that essentially decomposes the temporal security analysis problem into smaller and more manageable RBAC security analysis sub-problems for which the existing RBAC security analysis tools can be employed. We then evaluate them from a practical perspective by evaluating their performance using simulated data sets.

Keywords: Access control, temporal RBAC, safety analysis, temporal role hierarchy

1. Introduction

Access control facilitates controlled sharing and protection of resources in an enterprise. Today, there exist a variety of formal authorization models to meet the wide needs of requirements in specifying access control policies. These include, but not limited to, Discretionary Access Control (DAC), Mandatory Access Control (MAC) and Role Based Access Control (RBAC). Due to its flexibility, ease of administration and intuitiveness, RBAC has been successfully adopted as a means to enforce security by many organizations. Recognizing the industry needs, RBAC has been widely deployed in most commercial software including operating systems, database systems, enterprise resource planning and workflow systems. Under RBAC, roles represent organizational agents that perform certain job functions, and permissions to

*Corresponding author. E-mail: jsvaidya@business.rutgers.edu.

access objects are grouped as roles. Users, in turn, are assigned appropriate roles based on their responsibilities and qualifications [10,24]. This feature immediately reduces the operational costs of the system since the number of roles is usually much smaller than that of the permissions.

The success of RBAC led the development of some useful extensions to satisfy new application domains. In particular, researchers preserve the basic idea of having roles in the model and add some additional dimensions, like time and space. Temporal RBAC (TRBAC) [8], Generalized Temporal RBAC [20], Spatio-Temporal RBAC [2] are some examples of these extensions.

Analysis is essential to understand the implications of security policies. Although each policy may appear innocent in isolation, their cumulative effect may lead to an undesirable authorization state [27]. A study of the formal behavior of RBAC models helps organizations gain confidence on the level of control they have on the resources they own. Moreover, security analysis helps them set policies so that owners do not unknowingly lose their control on resources, and aids them make changes to the policies only if the analysis yields no security property violations.

One major advantage with RBAC is that, unlike in DAC where users can grant access privileges at their own discretion, organizations have central control over their resources. Since the security configuration need not be changed when users leave or join the organization, RBAC simplifies security administration. Administrative activities include user to role assignment (UA), permission to role assignment (PA) and role to role assignment (RH, the role hierarchies). Such administration is typically performed by a system security officer (SSO). For large organizations, it is normal to have roles in the order of thousands and users in the order of tens of thousands [28]. Typically, security administration is decentralized by delegating administrative activities, as it is overwhelming for a single SSO to administer all roles. While decentralized RBAC administration enhances the flexibility and scalability, an obvious side effect of it is reduced organizational control over its resources. Therefore, certain security guarantees are essential to ensure controlled delegation and to retain the desired level of control. Such guarantees can only be ensured through a formal *security analysis* of the properties of the RBAC system.

An RBAC system can be viewed as a state transition system where state changes occur via administrative operations. Given an initial authorization state and a set of security policies specified by authorization rules, a security analysis is a query the administrator makes on the set of reachable authorization states. Oftentimes, a security analysis is a simple query that asks whether there is an unintended reachable state, and hence requires determining the set of reachable authorization states. Such a query allows the administrator to determine if any of a set of unintended states could possibly occur as the system evolves, and is extremely important to determine if the system meets its security policies.

Exclusion of unintended authorization states, known as the *safety problem*, was first identified by Harrison, Ruzzo and Ullman [14], and can be formulated as testing

the following: "*Whether* there exists a reachable authorization state in which a particular subject possesses a particular privilege for a specific object." (Note that subjects include users as well as processes (invoked by users).) While safety is one of the fundamental requirements to be analyzed, the security properties to be analyzed in this paper will be more comprehensive than those studied in prior literature. We will study several security properties: safety, availability, liveness, and mutual exclusion of privileges for TRBAC.

Specifically, in this paper, we develop a security analysis methodology for Temporal RBAC. The analysis deals with the *reachability problem*, which seeks to determine whether a potentially untrusted user will ever get access to confidential objects. Protecting the confidentiality of the data and the integrity of the system requires this analysis. In our model we have user to role, permission to role and role to role assignments (role hierarchy) defined as temporal relations. The extent of the temporal notion in the role to role assignment relation is unique in terms of the flexibility that is achieved in the role hierarchy structure. We propose an administrative model that allows us to modify the above mentioned temporal relations. This administrative model is extensive in terms of its ability to modify the more flexible role hierarchy structure. In addition to our proposed administrative model, we provide a 3-stage analysis approach that is capable of conducting the *reachability analysis* for a given TRBAC configuration. The analysis is customizable in order to fit the specific needs of the security question of interest. We propose a novel approach for security analysis of TRBAC. The main strategy we use while performing the security analysis is to decompose the TRBAC analysis problem into multiple subproblems similar to RBAC. Essentially, we decompose the problem into simpler RBAC subproblems so that deciding whether a particular target state is reachable or not can be potentially simpler. Additionally, it lends itself to employ the analysis techniques developed for traditional RBAC. We propose decomposition based on the type of the relations as well as based on time. We present two different strategies for decomposition based on time – (1) Decomposition using rule schedules and (2) Decomposition using role schedules. We perform computational experiments using the proposed analysis to demonstrate its run time performance. We also provide a discussion about how our newly defined role hierarchy structure should physically be kept in order to achieve maximum performance in terms of access decisions and hierarchy modifications.

The rest of the paper is organized as follows: in Section 2, we provide background information necessary to follow the models and analysis strategies proposed in the paper. In Section 3 we provide our Temporal RBAC model, along with its administrative model and in Section 4 we present our security analysis methodology. In Section 5, we demonstrate the runtime performance of our proposed approach. In Section 6, we provide an insight about the data structure that the dynamic temporal role hierarchies should be kept to improve the performance. In Section 7, we briefly review the related work done in the literature. Finally in Section 8, we summarize our contributions in this paper and discuss our future work.

2. Preliminaries

In this section, we introduce the preliminary definitions and concepts that are needed to develop the approaches presented in this paper. Specifically, we present the definitions for Role Based Access Control model, the extensions and the administrative model of RBAC, and the reachability problem.

Definition 1 (Role Based Access Control configuration). An RBAC configuration [11] is a tuple $\langle U, R, \mathit{PRMS}, \mathit{UA}, \mathit{PA}, \mathit{RH}\rangle$ where U, R and *PRMS* are finite sets of users, roles, and permissions, respectively, $\mathit{UA} \subseteq U \times R$ is the user to role assignment relation, $\mathit{PA} \subseteq \mathit{PRMS} \times R$ is the permission to role assignment relation and $\mathit{RH} \subseteq R \times R$ is the role to role assignment (role hierarchy) relation.

A tuple $(u, r) \in \mathit{UA}$ represents that user u belongs to role r. Similarly, $(p, r) \in \mathit{PA}$ represents that members of role r are granted permission p. A tuple $(r_1, r_2) \in \mathit{RH}$ denotes r_1 is superior to r_2, so that any user who has r_1 assigned, also has r_2 assigned, and hence the permissions that are assigned to r_2.

The *Administrative RBAC* (ARBAC97) [25] model specifies rules to modify an RBAC configuration. It is composed of three modules `URA` user to role administration, `PRA` permission to role administration, and `RRA` role hierarchy administration.

The `URA` module allows to make changes to *UA* by using assignment/revocation rules performed by administrators. Administrators are those users that belong to administrative roles. We denote the set of administrative roles as *AR*. Some policies consider the set *AR* to be disjoint from the set of roles R. Those policies are said to meet the *separate administration* constraint [30]. A user can be assigned to a role if she satisfies the *precondition* associated to that role. A *precondition* is a conjunction of literals, where each literal is either in positive form r or in negative form $\neg r$, for some role $r \in R$. Following [12], we represent preconditions by two sets of roles *Pos* and *Neg*. A user u satisfies a precondition (*Pos*, *Neg*) if u is member of all roles in *Pos* and does not belong to any role of *Neg*.

Rules to assign users to roles are specified by the set [25]:

$$\texttt{can\_assign} \subseteq \mathit{AR} \times 2^R \times 2^R \times R.$$

A `can_assign` tuple $(\mathit{admin}, \mathit{Pos}, \mathit{Neg}, r) \in$ `can_assign` allows a member of the administrative role *admin* to assign a user u to roles r provided u's current role memberships satisfies the precondition (*Pos*, *Neg*).

Rules to revoke users from roles are specified as follows:

$$\texttt{can\_revoke} \subseteq \mathit{AR} \times R.$$

If $(\mathit{admin}, r) \in$ `can_revoke`, a member of the administrative role $\mathit{admin} \in \mathit{AR}$, can revoke the membership of any user from role $r \in R$.

PRA is the module to control the permission to role assignments. The rules are similar to those in URA. These are defined as follows:

$$\texttt{can\_assignp} \subseteq \mathit{AR} \times 2^R \times 2^R \times R,$$

$$\texttt{can\_revokep} \subseteq \mathit{AR} \times R.$$

Finally the ARBAC97 has RRA component to perform operations on roles and role hierarchies. The rule defined for this context is the following:

$$\texttt{can\_modify} \subseteq \mathit{AR} \times 2^R.$$

Using this rule, authorized administrators can create and remove roles and also they can modify the relationships between the roles.

A URA can be seen as a state-transition system defined by the tuple $\mathcal{S} = \langle U, R, \mathit{UA}, \texttt{can\_assign}, \texttt{can\_revoke}\rangle$. A *configuration* of $\mathcal{S}$ is any user to role assignment relation $\mathit{UR} \subseteq U \times R$. A configuration *UR* is *initial* if $\mathit{UR} = \mathit{UA}$. Given two $\mathcal{S}$ configurations $c = \mathit{UR}$ and $c' = \mathit{UR}'$, there is a *transition* (or *move*) from c to c' with rule $m \in (\texttt{can\_assign} \cup \texttt{can\_revoke})$, denoted $c \xrightarrow{\tau_m} c'$, if there exists an *administrative* user *ad* and administrative role *admin* with $(\mathit{ad}, \mathit{admin}) \in \mathit{UR}$ and a user $u \in U$, and one of the following holds:

can-assign move: $m = (\mathit{admin}, P, N, r)$, $P \subseteq \{r' \mid (u, r') \in \mathit{UR}\}$, $N \subseteq R \setminus \{r' \mid (u, r') \in \mathit{UR}\}$, and $\mathit{UR}' = \mathit{UR} \cup \{(u, r)\}$;

can-revoke move: $m = (\mathit{admin}, r)$, $(u, r) \in \mathit{UR}$, and $\mathit{UR}' = \mathit{UR} \setminus \{(u, r)\}$.

A *run* (or *computation*) of $\mathcal{S}$ is any finite sequence of $\mathcal{S}$ transitions $\pi = c_1 \xrightarrow{\tau_{m_1}} c_2 \xrightarrow{\tau_{m_2}} \ldots \xrightarrow{\tau_{m_{n-1}}} c_n \xrightarrow{\tau_{m_n}} c_{n+1}$ for some $n \geqslant 0$, where c_1 is the *initial* configuration of $\mathcal{S}$. An $\mathcal{S}$ configuration c is *reachable* if c is the last configuration of a run of $\mathcal{S}$.

Definition 2 (Reachability problem). Given a URA system $\mathcal{S}$ over the set of roles R and a role $\texttt{goal} \in R$ and a user u, the *role-reachability problem* asks whether a configuration c with $(\texttt{u}, \texttt{goal}) \in \texttt{c}$ is reachable in $\mathcal{S}$.

The reachability problem seeks to answer certain questions including and not limited to the following [22]:

- *Simple safety*: Is there a reachable state in which user u belongs to a user set s? Eventually, this can also be stated as: Can user u ever get access to the roles assigned to users that belong to set s?
- *Simple availability*: In each reachable state, does a user u always belong to a user set s? Hence this analysis questions whether user u will lose his/her privileges in the future.

- *Bounded safety*: In each reachable state, is the user set s always bounded by $\{u_1, u_2, \ldots, u_n\}$?
- *Liveness*: In every reachable state, does user set s always have at least one user?
- *Containment*: In every reachable state, does a user set s_1 always cover user set s_2.

For example, if the analysis of interest is *Simple safety*, then one should set the *goal* to the target role and check whether that state is reachable, whereas, if the analysis of interest is *Simple availability*, then the *goal* should be set to the state where the desired roles are unavailable. A *Liveness* query can be handled by performing a *Simple availability* check on the users in set s to see whether there exist at least one user in s always remain assigned to the particular role(s). Similar queries can be set for the other analysis questions.

Temporal RBAC: The basis of the temporal RBAC models in the literature relies on a *Calendar* definition, which is a periodic and duration expression given in terms of some calendars as follows [7]:

$$P = \sum_{i=1}^{n} O_i \cdot C_i \triangleright r \cdot C_d.$$

This expression is composed of two different calendar expressions split by $\triangleright$. The first part is the periodic expression which denotes the starting points of the time intervals represented by the expression. Each $C_i, C_i \sqsubseteq C_{i-1}$ is a calendar that represents a different time unit (days, weeks, minutes) so that for each $C_i \sqsubseteq C_{i-1}$ C_{i-1} can be covered by a finite number of intervals of C_i (for instance 24 hours is 1 day). The O_i's are the frequency components associated with the calendars, which are defined as $O_1 = \mathit{all}$, $O_i \in 2^{\mathbb{N}} \cup \{\mathit{all}\}$. The second part of the expression is the duration constraint which describes the time interval that the expression covers once stared with the periodic expression given in the first part. Here, $r \in \mathbb{N}$ and $C_d \sqsubseteq C_n$, meaning that the duration cannot exceed the maximum periodic time interval. An example for this expression is that $\mathit{all}.\mathit{Years} + \{3, 7\}.\mathit{Months} \triangleright 2.\mathit{Months}$ means a two month interval that starts every year at the beginning of the third and the seventh months.

The TRBAC model [8] supports role enabling, which is a tuple composed of roles and calendar expression. In GTRBAC model [20], user to role and permission to role assignments are also proposed to be temporal with the calendar expression in addition to some other components like role triggers.

Previous studies propose temporal role hierarchies [18–20] that focus on the permission and activation inheritance in the presence of temporal constraints on role enabling and disabling. Particularly, the role hierarchy is still *static*, but the other temporal components of the system have a governing effect on whether the hierarchies will provide inheritance relation at a given time. These studies propose three different types of hierarchies for temporal domain:

(1) *Inheritance-Only Hierarchy* ($\geqslant$): In this relation, the permissions in the junior role can be acquired by any user who activated a senior role, without activating the junior role. This hierarchy becomes restricted, if the enabling times of the roles are taken into account. There are two types of restrictions possible: Weak and Strong. When a hierarchy is weakly restricted, then the permission acquisition through the junior role is possible regardless of that role being enabled at that time. However, in the case of strongly restricted hierarchy, the junior role must be enabled to perform permission acquisition.
(2) *Activation-Only Hierarchy* ($\succeq$): In this relation, a user who activated a senior role can activate a junior role even if she is not explicitly assigned to it. This hierarchy becomes restricted, if the enabling times of the roles are taken into account. Similar to the Inheritance-Only case, there are two types of restrictions possible: Weak and Strong. When a hierarchy is weakly restricted, then the role activation of the junior role is possible regardless of that role being enabled at that time. However, in the case of strongly restricted hierarchy, the junior role must be enabled in order to be activated through the senior role.
(3) *Inheritance and Activation (General Inheritance) Hierarchy* ($\gg$): This relation is a combination of above two hierarchies. Senior roles can activate junior roles or just inherit some of the permissions of them. Lastly, a *Hybrid Hierarchy* exists when the pairwise relations among different roles are of different types. Hence, there can be an inheritance only relation between two roles, and an activation only relation between two other roles in the same hierarchy.

3. Temporal RBAC model and security questions

The security analysis in temporal domain requires determining how the time is embedded into the model and which components of the model are affected by this. Furthermore, an administrative model is necessary to allow certain changes in the role assignments. Then, a security analysis is possible for the TRBAC model.

3.1. Temporal components in Temporal RBAC model

In RBAC models with temporal components that are proposed in the literature, the majority of them focus extensively on the temporal user to role assignment relation and role enabling and the benefits of having temporal constraints on them. In this paper, we not only cover these two relations, but also focus on two other relations, namely, permission to role assignment and role hierarchies, as well. Now, we discuss potential benefits of having temporal permission to role assignments and temporal role hierarchies.

Temporal Permission to Role Assignments, captures the changes in *PA* with respect to time, hence, a role can have different permission assignments in different time intervals. This concept, although look similar to temporal *UA*, can have different

applications in a TRBAC model, including reducing the number of roles necessary. Let us explain this with an example:

Example 1. Consider a manufacturing company has two different production plants in different cities, one also has the headquarters of the company. The company has a CEO and a General Manager (GM) who works at both the plants; an Accounting Manager (AM), a Manufacturing Manager (MM), and a Human Resources Manager (HR) for each plant. Although the CEO works at the headquarters, GM works in both of the plants in different days of the week. When he is present at a plant, he manages the operations and audits the actions of the AM of that plant. However, when he is away (at the other plant), MM has the responsibility to audit the operations of AM without completely assuming the GM role, which is considered to have many additional permissions. In this case a TRBAC model without Temporal *PA* must have two different roles for each MM: Regular MM and Extended MM, and in Temporal *UA* the necessary assignments are done. However, presence of Temporal *PA* allows the model to have only one MM role that has different permission assignments that captures the auditing process whenever necessary.

In the Temporal RBAC model, role hierarchies can also be temporal in nature, in other words, they may change with time. Although role hierarchies in prior temporal extensions of RBAC have been specified, they do not allow temporal constraints to be specified on RH that not only *restrict the time* during which the hierarchy is valid, but also *change its structure* by shifting the position of the roles in the hierarchy. An immediate effect of this is that permission inheritance does not always hold. Essentially this means that a senior level role cannot always inherit the permissions of a junior level role. Furthermore, a role may change its level in the hierarchy, for example, a junior level role may be elevated to a higher level role during certain time periods.

Although enterprises usually specify a static hierarchy, a *dynamic temporal role hierarchy* (DTRH) comes into play in some temporary or periodical exceptional situations that are required for operational purposes. In the following, we provide such a motivating example.

Example 2. Consider once again the manufacturing company given in the previous example. The auditing tasks of MM can be modeled with DTRH, if the tasks required for auditing can be acquired through the role hierarchy given in Fig. 1. A policy which makes the Manufacturing Manager move to the second level, on top of the Accounting Manager only on the days when the General Manager is away will provide permissions needed for auditing the AM to MM.

Nevertheless, it is still possible to represent the scenario in the example above using a static role hierarchy. However, lack of temporal role hierarchies will force the system administrators to create a dummy role, like “Manager and Auditor”, that is

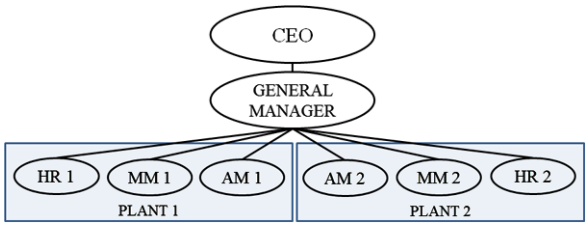


Fig. 1. The role hierarchy of the manufacturing company. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-140510.)

only enabled when necessary. Also, this role should have the required permission and hierarchy assignments that Manufacturing Manager needs. This newly created role does not essentially represent a regular job function since the Manufacturing Manager cannot assume this role all the time. Moreover, the Manufacturing Manager should be assigned to two separate roles which are enabled and disabled in regular time intervals. The situation might get even more complicated in the case of temporary changes in the system. Suppose that this auditing position is applied only when the General Manager is on vacation. Then the newly created dummy role and the necessary permission assignments are performed just for a single and temporary occurrence. Even worse, the administrators must undo the changes in the system, by revoking and deleting this role when the General Manager returns. Skipping this step would create serious safety problems. Clearly, creation of these redundant dummy roles increases the administrative burden [13].

Role delegation, which has been studied extensively in the literature [1,5,6,9,17, 33–35], is another way of handling scenarios like this. Users are delegated to the necessary roles of the users that are away. Even though our example scenario can be modeled using role delegation without imposing significant overhead, using temporal role hierarchies has still an advantage in terms of performing safety analysis. Whether handling the temporal role hierarchies is done using the specification of DTRH, using dummy roles or delegation, none of the prior work on safety analysis considers RBAC models with temporal constraints on role hierarchies.

3.2. TRBAC and its administrative model

Although temporal role based access control models have been proposed in the literature, none of them addresses the security analysis of policies. The temporal dimension of the model makes it even harder to perform security analysis, which is already proved to be intractable for the nontemporal case. Therefore, while preserving the core idea of having the temporal notion embedded into the RBAC components as in [8,20], we simplify the model to allow for a manageable security analysis. Although, our simplified model does not completely represent the previous temporal models, such as TRBAC or GTRBAC as a whole, we call this model Temporal RBAC (or TRBAC, in short) for notational simplicity. Therefore, the model referred

as TRBAC for the remainder of the paper represents our simplified model, unless otherwise noted. Now, we explain our TRBAC model in detail. We first define how the *time* is represented in the model:

Let T_{MAX} be a positive integer. A *time slot* of *Times* is a pair $(a, a+1)$, where a is an integer, and $0 \leqslant a < a+1 \leqslant T_{MAX}$. A time slot $(a, a+1)$ represents the set of all times in the set $[a, b)$, i.e., $\{t \mid a \leqslant t < b\}$. We use a *time interval*, consisting of a pair (a, b) where a, b are two integers and $0 \leqslant a < b \leqslant T_{MAX}$, to represent the set of corresponding time slots $\{(a, a+1), (a+1, a+2), \ldots, (b-1, b)\}$ succinctly. A *schedule* over T_{MAX} is a set of time slots.

For instance, consider a hospital that works for 24 hours with three shifts (between 9 am and 5 pm, between 5 pm and 1 am, and between 1 am and 9 am). If we want to have the precision of hours, we choose $T_{MAX} = 24$, and a schedule s that covers shifts 9 am–5 pm and 5 pm–1 am is represented as $s = \{(9, 10), (10, 11), \ldots, (23, 24), (24, 1)\}$. The schedule definition is a simplified version of the Calendar definition in Bertino et al. [8], where we have simpler periodic constraints and do not have duration constraints.

We assume that the system is periodic, thus the schedules repeat themselves after any T_{MAX}; in the hospital example above, time intervals are repeated each 24 h. Given a schedule s over T_{MAX} and an real number t, we say that t belongs to s, denoted $t \in s$, if there is a time interval $(a, b) \in s$ such that $t' \in [a, b)$, where $t' = t \bmod T_{MAX}$.

Definition 3 (TRBAC configuration). Let S be the set of all possible schedules over T_{MAX}. A TRBAC configuration over T_{MAX} is a tuple $M = \langle U, R, \mathit{PRMS}, \mathit{TUA}, \mathit{TPA}, \mathit{RS}, \mathit{DTRH}\rangle$ where U, R and *PRMS* are finite sets of *users*, *roles* and *permissions*, respectively, $\mathit{TUA} \subseteq (U \times R \times S)$ is the *temporal user to role assignment* relation, $\mathit{TPA} \subseteq (\mathit{PRMS} \times R \times S)$ is the *temporal permission to role assignment* relation, $\mathit{RS} \subseteq (R \times S)$ is the *role–status* relation and *DTRH* is the *dynamic temporal role hierarchy* relation.

A tuple $(u, r, s) \in \mathit{TUA}$ represents that user u is a member of the role r only during the time intervals of schedule s. During the life time of the system, a role can be either enabled or disabled. A tuple $(r, s) \in \mathit{RS}$ imposes that role r is *enabled* only during the time intervals of s (and therefore it can be assumed to be a member of r only at these times), and *disabled* otherwise. A tuple $(p, r, s) \in \mathit{TPA}$ means that permission p is associated to role r only in the time intervals denoted by s. Thus, a user u is granted permission p at time $t \in [0, T_{MAX}]$ provided that there exists a role $r \in R$ such that $(u, r, s_1) \in \mathit{TUA}$, $(r, s_2) \in \mathit{RS}$, $(p, r, s_3) \in \mathit{PA}$, and $t \in (s_1 \cap s_2 \cap s_3)$, for some time intervals s_1, s_2 and s_3.

We assume that relation *RS* for each role $r \in R$ contains always exactly one pair with first component r. Similarly, the relation *TUA* contains exactly one tuple for each pair in $U \times R$. Thus, if a role r is disabled in any time interval, we require that *RS* relates r with the empty schedule. Similarly, if a user u does not belong to a role r

in any time interval, the pair (u, r) is associated to the empty schedule by the relation *TUA*.

Permission inheritance and role activation through role hierarchies require additional definitions. In our model, *DTRH* is represented as a collection of dynamic temporal role hierarchy policies, which are tuples consisted of a pair of roles associated with a schedule that denotes the time slots that the policy is valid. In our model, we have dynamic temporal role hierarchy for inheritance only relation $DTRH_I$, for activation only relation $DTRH_A$ and for general inheritance relation $DTRH_{IA}$, all comprises as $DTRH = DTRH_I \cup DTRH_A \cup DTRH_{IA}$.

Definition 4. A *dynamic temporal role hierarchy policy* $(r_1 \geqslant_{s,weak} r_2) \in DTRH_I$ between roles r_1 and r_2 is an *inheritance-only weak temporal hierarchy relation*, that is valid in the time slots specified by a schedule s. Under this policy, a user u who can activate r_1 can inherit permissions of r_2 at time t if (1) $(u, r_1, s_1) \in TUA$, (2) $(r_1, s_2) \in RS$ and (3) $t \in (s_1 \cap s_2 \cap s)$, provided that there exists schedules s_1 and s_2 that determine the time slots that u is assigned to r_1 and r_1 is enabled, respectively.

Definition 5. A *dynamic temporal role hierarchy policy* $(r_1 \succeq_{s,weak} r_2) \in DTRH_A$ between roles r_1 and r_2 is an *activation-only weak temporal hierarchy relation*, that is valid in the time slots specified by a schedule s. Under this policy, a user u can activate r_2 at time t if (1) $(u, r_1, s_1) \in TUA$, (2) $(r_2, s_2) \in RS$ and (3) $t \in (s_1 \cap s_2 \cap s)$, provided that there exists schedules s_1 and s_2 that determine the time slots that u is assigned to r_1, and r_2 is enabled, respectively.

Definition 6. A *dynamic temporal role hierarchy policy* $(r_1 \gg_{s,weak} r_2) \in DTRH_{IA}$ between roles r_1 and r_2 is a *general weak temporal hierarchy relation*, that is valid in the time slots specified by a schedule s. Under this policy, a user u can activate r_2 at time t, or inherit permissions of r_2 if (1) $(u, r_1, s_1) \in TUA$, (2) $(r_2, s_2) \in RS$ and (3) $t \in (s_1 \cap s_2 \cap s)$, provided that there exists schedules s_1, and s_2 that determine the time slots that u is assigned to r_1 and r_2 is enabled, respectively.

In the above three definitions, the relations become strong (i.e: $(r_1 \geqslant_{s,strong} r_2) \in DTRH_I$, $(r_1 \succeq_{s,strong} r_2) \in DTRH_A$ and $(r_1 \gg_{s,strong} r_2) \in DTRH_{IA}$), when (2) is replaced with $(r_1, s_2), (r_2, s_3) \in RS$ and (3) is replaced with $t \in (s_1 \cap s_2 \cap s_3 \cap s)$ where s_3 is the schedule that determine the time slots that r_2 is enabled.

Presence of more than one type of relation makes *DTRH* a *hybrid* hierarchy.

Dynamic temporal role hierarchy policies $(r_1 \geqslant_{s,weak} r_2) \in DTRH$ satisfy the following properties for a given schedule s:

(1) *Reflexive:* $(r_1 \geqslant_{s,weak} r_1) \in DTRH$.
(2) *Transitive:* If $(r_1 \geqslant_{s,weak} r_2), (r_2 \geqslant_{s,weak} r_3) \in DTRH$, then $(r_1 \geqslant_{s,weak} r_3) \in DTRH$.

(3) *Asymmetric:* If $(r_1 \geqslant_{s,weak} r_2) \in DTRH$ then $(r_2 \geqslant_{s,weak} r_1) \notin DTRH$. These properties apply for both strong and the other types of relations ($\succeq, \gg$) as well.

Different policies among different roles create derived relations. As discussed in [19] derived relations determine the scope of activation or inheritance privileges upon activating a role. We adopt these derived relations to the case of dynamic temporal role hierarchies in the following definition.

Definition 7. A derived relation among roles $x, y_1, y_2, \ldots, y_n, z \in R$ holds under any of the following conditions:

(1) $(x\langle\mathcal{F}\rangle_{s_0,type}y_1) \wedge (y_1\langle\mathcal{F}\rangle_{s_1,type}y_2) \wedge \cdots \wedge (y_{n-1}\langle\mathcal{F}\rangle_{s_{n-1},type}y_n) \wedge (y_n\langle\mathcal{F}\rangle_{s_n,type}z) \rightarrow (x\langle\mathcal{F}\rangle_{s,type}z)$ if $\mathcal{F} \in \{\geqslant, \succeq, \gg\} \wedge s = s_0 \cap \cdots \cap s_n$,
(2) $(x \geqslant_{s_0,type} y_1) \wedge (y_1\langle\mathcal{F}\rangle_{s_1,type}y_2) \wedge \cdots \wedge (y_{n-1}\langle\mathcal{F}\rangle_{s_{n-1},type}y_n) \wedge (y_n\langle\mathcal{F}\rangle_{s_n,type}z) \rightarrow (x \geqslant_{s,type} z)$ if $\mathcal{F} \in \{\geqslant, \gg\} \wedge s = s_0 \cap \cdots \cap s_n$,
(3) $(x \gg_{s_0,type} y_1) \wedge (y_1\langle\mathcal{F}\rangle_{s_1,type}y_2) \wedge \cdots \wedge (y_{n-1}\langle\mathcal{F}\rangle_{s_{n-1},type}y_n) \wedge (y_n\langle\mathcal{F}\rangle_{s_n,type}z) \rightarrow (x\langle\mathcal{F}\rangle_{s,type}z)$ if $\mathcal{F} \in \{\geqslant, \gg\} \wedge s = s_0 \cap \cdots \cap s_n$,
(4) $(x \gg_{s_0,type} y_1) \wedge (y_1 \succeq_{s_1,type} y_2) \wedge \cdots \wedge (y_{n-1} \succeq_{s_{n-1},type} y_n) \wedge (y_n \succeq_{s_n,type} z) \rightarrow (x \succeq_{s,type} z)$ if $s = s_0 \cap \cdots \cap s_n$.

The other rules stated in [19] hold as in the above definition provided that $s = (s_1 \cap s_2 \cap \cdots \cap s_n) \neq \emptyset$.

According to Definition 7, if all of the linked hierarchy policies are of same type, the derived policy is also of the same type. If the first policy is an inheritance only relation, then regardless of the other linked policies being activation only or general inheritance hierarchy, the derived relation will be an inheritance-only policy. Similarly, if the first policy is a general inheritance relation and the remaining policies are activation-only, the derived relation is an activation-only policy. Finally, if the first policy is a general inheritance relation and the other linked policies being activation only or general inheritance relations, the derived relation will be of the type of linked policies.

Now, we can present our administrative model that allows administrators to make changes to the role-status relation *RS*, temporal user to role assignment relation *TUA*, temporal permission to role assignment relation *TPA* and the dynamic temporal role hierarchy relation *DTRH* by using enable/disable, assignment/revocation and modify rules, respectively. The goal of these rules is to update the time intervals of the schedule s associated to the corresponding relation.

In the analysis of the TRBAC model, we assume that the analysis for *TPA* can be made separately, since it is not directly related to the analysis of other components in terms of the security questions in consideration. More specifically, the security questions ask whether it is possible for a user to get access to a role, which requires determining whether it is possible for the goal role to be assigned to the target user directly, or indirectly via the role hierarchy in a time interval and if the role is enabled during any portion of this time interval. On the other hand, the analysis for *TPA*

is needed to discover if there is a possibility for a permission to appear in a particular goal role. Therefore, we define the Temporal URA and Temporal PRA systems separately to observe the state transitions.

Definition 8 (Temporal User to Role Administration). A TURA system is a tuple $\mathcal{S}_T = \langle M, \texttt{can\_enable}, \texttt{can\_disable}, \texttt{t\_can\_assign}, \texttt{t\_can\_revoke}, \texttt{t\_can\_modify}\rangle$ where $M = \langle U, R, \mathit{PRMS}, \mathit{TUA}, \mathit{TPA}, \mathit{RS}, \mathit{DTRH}\rangle$ is a TRBAC policy over T_{MAX}, and $\texttt{can\_enable}, \texttt{can\_disable}, \texttt{t\_can\_assign}, \texttt{t\_can\_revoke} \subseteq (R \times S \times 2^R \times 2^R \times S \times R)$ and $\texttt{t\_can\_modify} \subseteq (R \times S \times 2^R \times 2^R \times 2^R \times 2^R \times S \times R \times R \times \{\mathit{strong}, \mathit{weak}\} \times \{\geqslant, \succeq, \gg\})$.

Definition 9 (Temporal permission to role administration). A TPRA system is a tuple $\mathcal{S}_T = \langle M, \texttt{t\_can\_assignp}, \texttt{t\_can\_revokep}\rangle$ where $M = \langle U, R, \mathit{PRMS}, \mathit{TUA}, \mathit{TPA}, \mathit{RS}, \mathit{DTRH}\rangle$ is a TRBAC policy over T_{MAX}, and $\texttt{t\_can\_assignp}, \texttt{t\_can\_revokep} \subseteq (R \times S \times 2^R \times 2^R \times S \times R)$.

A *configuration* of $\mathcal{S}_T$ for TURA is a triple $(\mathit{RS}, \mathit{TUA}, \mathit{DTRH})$, which is *initial* if $\mathit{RS} = \mathit{RS}_0$, $\mathit{TUA} = \mathit{TUA}_0$ and $\mathit{DTRH} = \mathit{DTRH}_0$. Similarly, a *configuration* of $\mathcal{S}_T$ for TPRA is a singleton (TPA), which is *initial* if $\mathit{TPA} = \mathit{TPA}_0$. Given two $\mathcal{S}_T$ configurations $c = (\mathit{RS}, \mathit{TUA}, \mathit{DTRH})$ and $c' = (\mathit{RS}', \mathit{TUA}', \mathit{DTRH}')$ for TURA' and $c = (\mathit{TPA})$ and $c' = (\mathit{TPA}')$ for TPRA, we describe below the conditions under which there is a *transition* (or *move*) from c to c' at time $t \in \mathbb{N}$ with rule $m \in \mathcal{M}_{\mathit{ALL}} = (\texttt{can\_enable} \cup \texttt{can\_disable} \cup \texttt{t\_can\_assign} \cup \texttt{t\_can\_revoke} \cup \texttt{t\_can\_modify} \cup \texttt{t\_can\_assignp} \cup \texttt{t\_can\_revokep})$, denoted $c \xrightarrow{(\tau_m, t)} c'$.

Before defining the transition relation, we first describe the components of move $m = (\mathit{admin}, s_{\mathit{rule}}, \mathit{Pos}, \mathit{Neg}, s_{\mathit{role}}, r)$. Move m can be executed only by a user, say *ad*, belonging to the *administrative role admin* $\in R$.

The times t in which *ad* can execute m are all those in which *ad* is assumed to be a member of role *admin*, and furthermore, t must also belong to the schedule s_{rule} which denotes the time intervals when m can be fired (or we say *valid*): $t \in (s_{\mathit{ad}} \cap s_{\mathit{admin}} \cap s_{\mathit{rule}})$ where $(\mathit{ad}, \mathit{admin}, s_{\mathit{ad}}) \in \mathit{TUA}$ and $(\mathit{admin}, s_{\mathit{admin}}) \in \mathit{RS}$. In the rest of the section we say that m can be *executed* at time t whenever t fulfills the above condition. The component s_{role} is used to update the schedule of a role, or the membership of a user to a role, depending on the kind of rule of m. The pair of disjoint role sets $(\mathit{Pos}, \mathit{Neg})$ is called the *precondition* of m whose fulfillment depends by the kind of the rule m.

The fulfillment of the precondition of a can-enable and can-disable rule depends on the current status of the other roles. Let $\hat{s} \subseteq s_{\mathit{role}}$. A can-enable or can-disable rule $m = (\mathit{admin}, s_{\mathit{rule}}, \mathit{Pos}, \mathit{Neg}, s_{\mathit{role}}, r)$ satisfies its precondition $(\mathit{Pos}, \mathit{Neg})$ w.r.t. candidate schedule $\hat{s}$, if for every time slot $\alpha \in \hat{s}$, if (1) for every role $\mathit{pos} \in \mathit{Pos}$, $\alpha \subseteq s_{\mathit{pos}}$ where $(\mathit{pos}, s_{\mathit{pos}}) \in \mathit{RS}$, (2) for every role $\mathit{neg} \in \mathit{Neg}$, $\alpha \cap s_{\mathit{neg}} = \emptyset$, where $(\mathit{neg}, s_{\mathit{neg}}) \in \mathit{RS}$, and (3) α satisfies all preconditions. In other words, a candidate schedule $\hat{s} \subseteq s_{\mathit{role}}$ satisfies a precondition only if each time slot $\alpha \in \hat{s}$ satisfies the precondition individually. Let $(r, s) \in \mathit{RS}$.

Enabling rules. A can-enable rule adds a new schedule to a specific role. A tuple $(admin, s_{rule}, Pos, Neg, s_{role}, r) \in$ `can_enable` allows to update the tuple $(r, s) \in RS$ to $(r, s \cup \hat{s})$ for some schedule $\hat{s}$, provided that m can be executed at time t and also satisfies its precondition. Formally, rule m is executable at time t, m satisfies its precondition (Pos, Neg) w.r.t. schedule $\hat{s}$, $RS' = (RS \setminus \{(r, s)\}) \cup \{(r, s \cup \hat{s})\}$, and $TUA' = TUA$.

Disabling rules. A can-disable rule removes a schedule from a designed role. A tuple $m = (admin, s_{rule}, Pos, Neg, s_{role}, r) \in$ `can_disable` allows to update the tuple $(r, s) \in RS$ to $(r, s \setminus \hat{s})$, for some schedule $\hat{s}$, provided that m can be executed at time t, and satisfies its precondition. Formally, m is executable at time t, m satisfies its precondition (Pos, Neg) w.r.t. schedule $\hat{s}$, $RS' = (RS \setminus \{(r, s)\}) \cup \{(r, s \setminus \hat{s})\}$, and $TUA' = TUA$.

The next two rules are similar to those given above with the difference that we now update the schedules associated to each element of the user to role assignment relation. Another difference is that can-assign and can-revoke rules have a different semantics to fulfill their preconditions. A user $u \in U$ satisfies a precondition (Pos, Neg) w.r.t. a schedule $\hat{s}$ if for every time slot $\alpha \in \hat{s}$, (1) for every $(u, pos, s_{pos}) \in TUR$ with $pos \in Pos$, $\alpha \subseteq s_{pos}$, (2) for every $(u, neg, s_{neg}) \in TUA$ with $neg \in Neg$, $\alpha \cap s_{neg} = \emptyset$, and (3) α satisfies all preconditions. Let $(u, r, s) \in TUA$.

Assignment rules. A tuple $(admin, s_{rule}, Pos, Neg, s_{role}, r) \in$ `t_can_assign` allows to update the user to role assignment relation for the pair (u, r) as follows. Let $\hat{s}$ be a schedule over T_{MAX} with $\hat{s} \subseteq s_{role}$. Then, if m can be executed at time t, and user u satisfies the precondition (Pos, Neg) w.r.t. schedule $\hat{s}$, then the tuple (u, r, s) is updated to $(u, r, s \cup \hat{s})$, i.e. $TUA' = (TUA \setminus \{(u, r, s)\}) \cup \{(u, r, s \cup \hat{s})\}$ and $RS' = RS$.

Revocation rules. A tuple $(admin, s_{rule}, Pos, Neg, s_{role}, r) \in$ `t_can_revoke` allows to update the user to role assignment relation for the pair (u, r) as follows. Let $\hat{s}$ be a schedule over T_{MAX} with $\hat{s} \subseteq s_{role}$. Then, if m can be executed at time t, and user u satisfies the precondition (Pos, Neg) w.r.t. schedule $\hat{s}$, then the tuple (u, r, s) is updated to $(u, r, s \setminus \hat{s})$, i.e. $TUA' = (TUA \setminus \{(u, r, s)\}) \cup \{(u, r, s \setminus \hat{s})\}$ and $RS' = RS$.

The rules for updating the permission to role assignment is again similar to the user to role assignments rules, with the difference of assigning permissions and preconditions checked against the assigned permissions. The structure of the move definition is similar to the existing model, but the assignment semantics for permissions are different. Hence, the existing move definition, $m = (admin, s_{rule}, Pos, Neg, s_{role}, r)$ remains the same, but it applies to permissions rather than users.

Intuitively, a precondition in the permission level is a verification procedure of the existing role assignments of a given permission. For instance, a positive precondition (negative, resp.) can state a permission can only be added to a given role if it has already been (has not been, resp.) assigned to another role. More formally, a

permission $p \in PRMS$ satisfies a precondition (Pos, Neg) w.r.t. a schedule $\hat{s}$ if for every time slot $\alpha \in \hat{s}$, (1) for every $(p, pos, s_{pos}) \in TPA$ with $pos \in Pos$, $\alpha \subseteq s_{pos}$, (2) for every $(p, neg, s_{neg}) \in TPA$ with $neg \in Neg$, $\alpha \cap s_{neg} = \emptyset$, and (3) α satisfies all preconditions. Let $(p, r, s) \in TPA$.

Assignment rules. A tuple $(admin, s_{rule}, Pos, Neg, s_{role}, r) \in$ `t_can_assignp` allows to update the permission to role assignment relation for the pair (p, r) as follows. Let $\hat{s}$ be a schedule over T_{MAX} with $\hat{s} \subseteq s_{role}$. Then, if m can be executed at time t, and permission p satisfies the precondition (Pos, Neg) w.r.t. schedule $\hat{s}$, then the tuple (p, r, s) is updated to $(p, r, s \cup \hat{s})$, i.e. $TPA' = (TPA \setminus \{(p, r, s)\}) \cup \{(p, r, s \cup \hat{s})\}$, $TUR' = TUR$ and $ER' = ER$.

Revocation rules. A tuple $(admin, s_{rule}, Pos, Neg, s_{role}, r) \in$ `t_can_revokep` allows to update the permission to role assignment relation for the pair (p, r) as follows. Let $\hat{s}$ be a schedule over T_{MAX} with $\hat{s} \subseteq s_{role}$. Then, if m can be executed at time t, and permission p satisfies the precondition (Pos, Neg) w.r.t. schedule $\hat{s}$, then the tuple (p, r, s) is updated to $(p, r, s \setminus \hat{s})$, i.e. $TPA' = (TPA \setminus \{(p, r, s)\}) \cup \{(p, r, s \setminus \hat{s})\}$, $TUR' = TUR$ and $ER' = ER$.

The rule structure for `t_can_modify` is different from the other rules. This rule updates the valid time slots of the dynamic temporal role hierarchy policies. Also, in contrast to precondition structures that have been proposed in the literature for other administrative rules (like `can_assign`), it has two sets of preconditions, one for senior and one for junior role in order to protect the integrity of the hierarchy. The rule is composed of eight parameters that should be satisfied to execute the rule. Three of these parameters are similar to the above mentioned moves, namely, $admin$, s_{rule} and $s_{hierarchy}$ which is declared as s_{role} in other rules defined above, but has similar semantics. Let t be the time slot that the rule is required to be executed.

- $type \in \{strong, weak\}$ denotes the type of the hierarchy relation.
- r_{sr} is the Senior Role of the hierarchy policy.
- r_{jr} is the Junior Role of the hierarchy policy.
- $SR(Pos, Neg)$ denotes the positive and negative preconditions of the senior role r_{sr}. The preconditions are satisfied in the following way: Let $\hat{s}$ denote the time slots that are intended to be modified by the rule ($\hat{s} \subseteq s_{hierarchy}$). For each $r \in Pos$, there must be a role hierarchy policy $(r \geqslant_{\hat{s},type} r_{sr}) \in DTRH$ and for each $r \in Neg$, there must not be a hierarchy policy $(r \geqslant_{\hat{s},type} r_{sr}) \in DTRH$.
- $JR(Pos, Neg)$ denotes the positive and negative preconditions of the junior role r_{jr}. The preconditions are satisfied in the following way. Let $\hat{s}$ denote the time slots that are intended to be modified by the rule ($\hat{s} \subseteq s_{hierarchy}$). For each $r \in Pos$, there must be a role hierarchy policy $(r_{jr} \geqslant_{\hat{s},type} r) \in DTRH$ and for each $r \in Neg$, there must not be a hierarchy policy $(r_{jr} \geqslant_{\hat{s},type} r) \in DTRH$.

Modification rule. Under these parameters, a tuple: $(admin, s_{rule}, SR(Pos, Neg), JR(Pos, Neg), s_{hierarchy}, r_{sr}, r_{jr}, type) \in$ `t_can_modify` allows to update the role

hierarchy relation $r_{sr} \geqslant_{s,type} r_{jr}$ as follows: Let $\hat{s}$ be a schedule over T_{MAX} with $\hat{s} \subseteq s_{hierarchy}$. Then, if this rule can be executed at time t, and the preconditions are satisfied w.r.t. schedule $\hat{s}$, then the tuple $r_{sr} \geqslant_{s,type} r_{jr}$ is updated to $r_{sr} \geqslant_{s\cup\hat{s},type} r_{jr}$ or $r_{sr} \geqslant_{s\setminus\hat{s},type} r_{jr}$, depending on the intended modification. This definition is for inheritance only hierarchies, but it also applies to activation only and general inheritance hierarchies, by replacing $\geqslant$ with $\succeq$ and $\gg$.

Example 3. Let us now consider an example of a TRBAC system deployed in a hospital. Assume that there are 7 different roles, namely, Employee (*EMP*), Day Doctor (*DDR*), Night Doctor (*NDR*), Practitioner (*PRC*), Nurse (*NRS*), Secretary (*SEC*) and Chairman (*CHR*). Hospital works for 24 hours and there are three different shifts (time slots) from 8 am to 4 pm (Time Slot 1), 4 pm to 12 am (Time Slot 2) and 12 am to 8 am (Time Slot 3). Only the Chairman role (*CHR*) has administrative privileges.

(1) $(CHR, \{(0,2)\}, \{DDR\}, \emptyset, \{(0,1)\}, PRC) \in$ `can_enable`: At time slots 1 and 2, a chairman can enable the role *Practitioner* for the first time slot if the role *Day Doctor* is also enabled during this time slot.
(2) $(CHR, \{(0,3)\}, \{EMP, NDR\}, \{(2,3)\}, NRS) \in$ `can_disable`: At time slots 1, 2 and 3, a chairman can disable the role *Nurse* for the third time slot if the roles *Employee* and *Night Doctor* are enabled at this time slot.
(3) $(CHR, \{(0,2)\}, \{EMP\}, \{NRS\}, \{(0,2)\}, DDR) \in$ `t_can_assign`: At time slots 1 and 2, a chairman can assign the role *Day Doctor* for the first and the second time slots to any user that has *Employee* role and does not have *Nurse* role during these time slots.
(4) $(CHR, \{(0,3)\}, \emptyset, \emptyset, \{(0,3)\}, SEC) \in$ `t_can_revoke`: At time slots 1, 2 and 3, a chairman can revoke the role *Secretary* for all time slots of any user that has *Secretary* role assigned in these slots.
(5) $(CHR, \{(2,3)\}, \{EMP\}, \{NRS\}, \{(2,3)\}, NDR) \in$ `t_can_assignp`: At time slot 3, a chairman can assign a permission to the role *Night Doctor* for the third time slot if that permission is also assigned to *Employee* not assigned to *Nurse* role during this time slot.
(6) $(CHR, \{(0,2)\}, \emptyset, \emptyset, \{(0,3)\}, NRS) \in$ `t_can_revokep`: At time slots 1 and 2, a chairman can revoke a permission from the role *Nurse* for all time slots.
(7) $(CHR, \{(0,2)\}, \{DDR\}, \emptyset, \{(0,1)\}, PRC) \in$ `t_can_assign`: At time slots 1 and 2, a chairman can assign the role *Practitioner* for the first time slot of any user that has *Day Doctor* role during this time slot.
(8) $(CHR, \{(0,3)\}, \{NDR\}, \emptyset, \{(2,3)\}, PRC) \in$ `t_can_assign`: At time slots 1, 2 and 3, a chairman can assign the role *Practitioner* for the third time slot to any user that has *Night Doctor* role during this time slot.

Reachability problems. A *run* (or *computation*) of $\mathcal{S}_T$ is any finite sequence of $\mathcal{S}_T$ transitions $\pi = c_1 \overset{(\tau_{m_1},t_1)}{\longrightarrow} c_2 \overset{(\tau_{m_2},t_2)}{\longrightarrow} \dots \overset{(\tau_{m_{n-1}},t_{n-1})}{\longrightarrow} c_n \overset{(\tau_{m_n},t_n)}{\longrightarrow} c_{n+1}$ for some

$n \geqslant 0$, where c_1 is an *initial* configuration of $\mathcal{S}_T$, $t_1 = 0$, and $t_i \leqslant t_{i+1}$ for every $i \in [n-1]$. An $\mathcal{S}_T$ configuration c is *reachable within time* t, if there exists a run π in which $c_{n+1} = c$ and $t_n \leqslant t$. Furthermore, c is simply *reachable* if c is reachable within time t, for some $t \geqslant 0$.

Let $\mathcal{S}_T$ be a TURA system over T_{MAX}, u and r be a user and a role of $\mathcal{S}_T$, respectively, and s be a schedule over T_{MAX}. Given a time t, the *timed reachability problem* for $(\mathcal{S}_T, u, r, s, t)$ asks whether there is a reachable configuration within time t of $\mathcal{S}_T$ in which user u is a member of role r in the schedule s either explicitly or implicitly through the role hierarchy. Similarly, the *reachability problem* for $(\mathcal{S}_T, u, r, s)$ is defined as above where there is no constraint on time t. In all of the time slots of s, r must also be enabled.

For a TPRA system over T_{MAX} which is identified by $\mathcal{S}_T$, and p and r are a permission and a role of $\mathcal{S}_T$, respectively, and s be a schedule over T_{MAX}. Given a time t, the *timed reachability problem* for $(\mathcal{S}_T, p, r, s, t)$ asks whether there is a reachable configuration within time t of $\mathcal{S}_T$ in which user u is a member of role r in the schedule s. Similarly, the *reachability problem* for $(\mathcal{S}_T, u, r, s)$ is defined as above where there is no constraint on time t.

In our analysis, we assume Separate Administration, in which there is an administrative user who is assigned to the required administrative roles which are enabled all the time. Hence, the times to fire a rule is only restricted by s_{rule}.

3.3. Security analysis questions

In Temporal RBAC, the security problem is slightly different than that of RBAC. The model can have two different ranges of temporal coverage: Safety until a given period of time (or called short term safety), and the ultimate safety (or called long term safety). In short term safety, we are only interested in the safety of the system until a given fixed time. Practically, this type of an analysis is useful to track users that will have temporary presence in the system. Whereas, the long term safety is more concerned about the regular users which are likely to be active in the system for relatively longer periods of time. This analysis will yield an ultimate safety of the system in the long run. Furthermore, changes allowed in the role hierarchy require additional security questions related to implicit assignments that are possible in the future. There is no problem of this sort in the case of static role hierarchies, however a simple manipulation in the hierarchy could create a security breach, and should be detected in advance to prevent any such occurrence. Considering these definitions, some example security questions for the temporal domain can be stated as follows:

(1) *Safety:*

   (a) *(Explicit Assignment – Short Term)* Will there be no reachable state in which a user u is assigned to a role r at time t?

(b) *(Explicit Assignment – Long Term)* Will a user u ever get assigned to a role r?
(c) *(Role Enabling – Long Term)* Will an enabled role r eventually be disabled?
(d) *(Implicit Assignment – Short Term)* Will a user u get *implicitly* assigned to role r at time t?
(e) *(Implicit Assignment – Long Term)* Will a user u ever get *implicitly* assigned to role r in the future?
(f) *(Permission Assignment – Long Term)* Will a permission p ever get assigned to role r in the future?

(2) *Liveness:*

(a) *(Role Enabling – Short Term)* Will an enabled role remain enabled at time t?
(b) *(Implicit Assignment – Short Term)* Will a user u lose privileges of any role that he is *implicitly* assigned until time t?
(c) *(Explicit Assignment – Long Term)* Will a user u ever lose any role that he is assigned in the future?
(d) *(Permission Assignment – Short Term)* Will a permission p remain assigned to role r at time t?

(3) *Mutual exclusion:*

(a) *(Explicit Assignment – Long Term)* Will a user u be assigned to roles r_1 and r_2 at the same time (i.e., do the time intervals during which u is assigned to roles r_1 and r_2 overlap?)
(b) *(Implicit Assignment – Short Term)* Will users u_1 and u_2 get *implicitly* assigned to role r at the same time slot until time t?

Regarding these security questions, our aim is to analyze TRBAC model to verify that the configuration is safe in terms of the questions stated above.

4. TRBAC security analysis

Given an initial configuration c_0, rules of an administrative model, $\mathcal{M}_{ALL}$ and the target user u, who is being analyzed against the security questions of interest, our proposed security analysis methodology provides answers to various security questions outlined in Section 3.3.

Our security analysis depends on a customizable three stage decomposition strategy. First we decompose the problem into four steps based on the temporal relation that is modified (*TUA*, *RS*, *TPA*, *DTRH*). Then, we further decompose each of these subproblems into smaller ones using the time dimension in which we have two different strategies to address different security questions – Rule Schedule and Role Schedule. Finally, combining the results obtained from each of these decomposed problems provide the complete analysis.

4.1. *Stage* 1: *Relation based decomposition*

The TURA and TPRA systems are composed of a set of different type of rules that are used to generate new configurations for a security analysis. The interactions among these rules, however, have certain properties. Consider the rules grouped according to the relations that they modify, i.e., t_can_assign, t_can_revoke; can_enable, can_disable; t_can_assignp, t_can_revokep; and t_can_modify are the four groups of rules that modify different relations in TRBAC. Assuming that the administrator role and rule schedule requirements are satisfied, the execution of roles of each group is determined by the relations that they modify. For instance, the preconditions to satisfy for t_can_assign and t_can_revoke are checked against the current status of *TUA*, whereas, it is *TPA* for t_can_assignp, t_can_revokep, *RS* for can_enable, can_disable and *DTRH* for t_can_modify. Therefore, the execution rules of different groups are independent of each other. However, this property does not imply that the *relations that are modified* with these rules are also independent semantically. For instance, role assignments and revocations can be performed for a user, but these assignments are useful only if the roles are enabled. Similarly, an inheritance through the role hierarchy is only possible if the senior role of the policy is enabled. Therefore, we perform independent analysis on four different components of the TRBAC model and then we combine the results obtained from each of these four sub-analysis problems in order to interpret them correctly in Stage 3.

Hence, regarding this property, our security analysis procedure is composed of four steps (Table 1). In each of these steps, the state configurations and the administrative rule sets of the analysis problems are shaped with different relations.

For each different analysis, the rule set is composed of the following rules:

(1) User Assignment: t_can_assign, t_can_revoke
(2) Role Enabling: can_enable, can_disable
(3) Role Hierarchy: t_can_modify
(4) Permission Assignment: t_can_assignp, t_can_revokep

Table 1
Subproblems, initial configurations and the relations used

Analysis performed	State configuration represented by	Initial configuration of the analysis
1. User Assignment	TUA	TUA relation of the target user
2. Role Enabling	RS	RS relations of all roles
	✓Explicit role assignment analysis is complete	
3. Role Hierarchy	DTRH	DTRH policies
	✓Implicit role assignment analysis is complete	
4. Permission Assignment	TPA	TPA relation of the target role
	✓Full analysis is complete	

This four step procedure depicted in Table 1 might be customized with respect to the scope of the security analysis. At the end of first step, the analysis generates all possible configurations for the target user under the administrative rules. The second step declares the time slots that the roles can get enabled. Combining the results of the first and the second step produces the analysis that answers the security questions related to explicit role assignments. If the implicit assignments are also considered, the third step should be performed. In the third step, possible role hierarchy relations are generated. Combining these results with the ones from the earlier steps will determine the possibility of an implicit assignment to a role. Finally, the fourth step determines the possible permission assignments to a role (or roles), which could also be conducted as an independent analysis determining whether there is a possibility for a set of permissions to appear in a role. In summary, one can choose different combinations of the steps outlined in Table 1. For example one can choose to carry out steps 1 and 2, steps 1, 2 and 3, steps 1, 2 and 4, or steps 1, 2, 3 and 4, based on the analysis they would like to perform.

4.2. Stage 2: *Time Based Decomposition*

Time Based Decomposition further simplifies the decomposed analysis problems in the first stage. Since the time dimension is discrete, we decompose each of the four security analysis problems above into multiple subproblems, so that each instance can be treated similar to an RBAC model. We employ two different alternative decomposition strategies – the *rule schedule strategy* and the *the role schedule strategy*. These strategies, although can analyze the same problem, provide answers to different security questions. Rule schedule strategy provides analysis for short term reachability, whereas role schedule strategy provides analysis for long term reachability. Each of the four steps of Stage 1 can be analyzed by these strategies under the state configuration and administrative rule settings depicted in Table 1. The time based decomposition strategies provide flexibility so that different RBAC analysis procedures can be employed as given in Table 2.

Before we provide details of these two strategies, we give the steps for each stage to be performed for some of the example security questions that we discuss in Section 3.3 in Table 3.

Table 2
Time Based Decomposition and available analyzers

Analysis performed	Rule Schedule	Role Schedule
User Assignment	SA	Any RBAC analyzer
Role Enabling	SA	Any RBAC analyzer
Role Hierarchy	MSA[a]	MSA
Permission Assignment	SA	Any RBAC analyzer

[a]Details given in Section 4.2.3.

Table 3
The steps of analysis to be performed for different security questions given in Section 3.3

Security question	Stage 1	Stage 2
1-a	1, 2	Rule Schedule
1-c	2	Role Schedule
1-d	1, 2, 3	Rule Schedule
1-f	4	Role Schedule
2-c	1, 2	Role Schedule
3-b	1, 2, 3	Rule Schedule

4.2.1. Rule Schedule Strategy

Rule Schedule Strategy is a state space exploration approach utilizing rule schedules (s_{rule}) to decompose the analysis into smaller problems and analyze them serially with respect to time. In this strategy, we use the RBAC analysis approach by Stoller et al. [30] extensively to explore potential states reachable in different time instances.

Let $m \in \mathcal{M}_c \subseteq \mathcal{M}$ be a subset of the rules in the analysis problem. A *constant region* $\mathcal{C}(a, b, \mathcal{M}_c)$ is a bounded time interval between $t = a$ and $t = b$, $a \leqslant b$ such that $\forall m \in \mathcal{M}_c$, $(a, b) \subseteq s^m_{rule}$ and $\nexists m' \notin \mathcal{M}_c$ such that $s^{m'}_{rule} \subseteq (a, b)$. Informally, if a rule m is included in a constant region $\mathcal{C}$ then it should be valid in all time slots $\alpha \in (a, b)$, and there should not be any other rule m' that is valid in some but not all of the time slots of (a, b). In the rule schedule approach, we split the timeline from 0 to T_{MAX} into nonoverlapping constant regions $\mathcal{C}_i$ w.r.t. the s_{rule} of the roles.

In the analysis, we trace *constant regions* $\mathcal{C}_1, \mathcal{C}_2, \ldots$ serially with respect to time. These regions can be seen as separate RBAC systems. However, $\mathcal{C}_{i+1}$ depends on $\mathcal{C}_i, \forall i$, which implies the output of an RBAC reachability analysis at $\mathcal{C}_i$ is an input (or initial configuration) to $\mathcal{C}_{i+1}$. Since an RBAC analysis could result in multiple configurations, then, in each *constant region*, a separate RBAC analysis should be performed for each configuration generated by the analysis done in the previous *constant region*.

Example 4. Now, let us consider the hospital example given in Section 3.2. There are eight different administrative rules with different valid periods as depicted in Fig. 2, where the bars indicate their respective rule schedules. As can be seen from the figure, the set of valid rules does not change in interval $(0, 2)$ $\mathcal{C}_1$ and $(2, 3)$ ($\mathcal{C}_2$). More specifically, the valid rules for $\mathcal{C}_1$ are 1, 2, 3, 4, 6, 7, 8 and the valid rules for $\mathcal{C}_2$ are 2, 4, 5, 8. Essentially, we decompose the analysis problem of TRBAC into two subproblems which are similar to RBAC problems pertaining to these *constant regions*.

There are other issues related to role schedules that are assigned by the rules. Recall that all of the rules have a role schedule which denotes the time intervals that the role can be assigned. But, according to the rule definitions, the administrators are

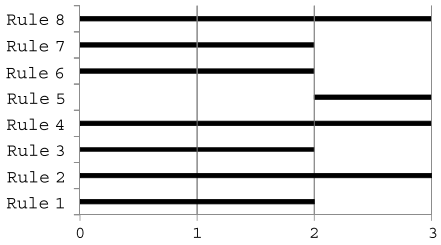


Fig. 2. Rule schedules.

free to choose a sub schedule of the role schedule and assign/revoke, enable/disable and modify the role (hierarchy) schedules only for some of the designated time intervals. This further complicates the reachability analysis, since in a serial fashion, one should keep all of the possible schedule combinations for the subsequent time intervals. Therefore we make the following assumption to simplify the analysis:

Sub-schedule assumption. For each rule, the role (or hierarchy) schedule modification operations are performed using the entire schedule s_{role} ($s_{hierarchy}$, resp.). This means that an administrator may use a rule to assign the associated role r to a user u all of the subsets of the schedule s_{role} (as long as the preconditions are satisfied). In our analysis, we assume that s_{role} ($s_{hierarchy}$, resp.) is assigned or revoked completely – no sub schedule assignments are allowed. Hence, this assumption ensures that a rule can only generate at most one (new) configuration.

Here we provide a sketch of the algorithm. The TRBAC reachability analysis starts with an initial configuration c_0 and *constant region* $\mathcal{C}_1$. The state space is expanded using Stoller et al.'s algorithm [30] (we refer this algorithm as SA) and the rules that are valid at time $t = 0$.[1] At the end of this step, a set of reachable configurations, $\mathcal{S}_1 = \{c_1, c_2, \ldots, c_m\}$ are obtained. Afterwards, the analysis moves to $\mathcal{C}_2$. For each distinct configuration obtained so far, SA is used to expand these configurations using the valid rules in this constant region. At the end of this step, we obtain an updated set of reachable configurations $\mathcal{S}_2 \supseteq \mathcal{S}_1$. The algorithm then moves to $\mathcal{C}_3$ and the trace goes in this fashion for a specified number of cycles P of length T_{MAX} (The algorithm returns to $\mathcal{C}_1$ whenever T_{MAX} is reached). Since TURA tuple S_T is finite and since the iterations are bounded by the number of cycles, the algorithm is guaranteed to terminate. However since this approach is a greedy heuristic, we are not guaranteed to get an optimal solution.

4.2.2. Role schedule strategy

In this approach, we split the TRBAC security analysis problem into smaller RBAC security analysis subproblems using the role schedules of the rules. The main idea is to generate subproblems $\mathcal{T}(\alpha, \mathcal{M}_s)$ for each time slot $\alpha \in (0, T_{MAX})$ with nontemporal administrative rules, so that the system can be treated like an RBAC.

[1] For the analysis of Dynamic Temporal Role Hierarchies, certain modifications are required as given in Section 4.2.3.

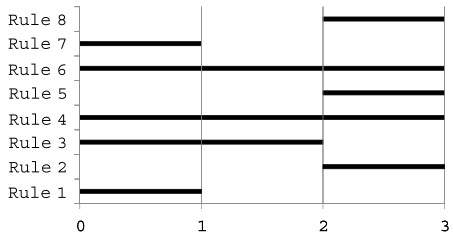


Fig. 3. Role schedules.

Example 5. Consider Fig. 3, which shows the role schedules of the rules in the hospital example given in Section 3.2. Here, we have three distinct time slots (Time Slot 1: (0, 1), Time Slot 2: (1, 2), Time Slot 3: (2, 3)) with different rules. The rules for Time Slot 1 are Rule 1, 3, 4, 6 and 7; for Time Slot 2 are Rule 3, 4 and 6; for Time Slot 3 are Rule 2, 4, 5, 6 and 8.

In order to achieve nontemporal administrative rules, (and hence an RBAC system for each time slot), we need to remove two components: Rule Schedules and Role Schedules (Hierarchy Schedules) and we need to show the inter-time slot independency. The removal of the role schedules follows the definition of subproblems $\mathcal{T}(\alpha, \mathcal{M}_s)$. For the rule schedules, we observe the Long Run Behavior property of the administrative model that we propose.

Long Run Behavior. In the long run, rule schedules of the rules can be neglected, if the system is periodic.

Here we give the intuition of this result. Rule schedules restrict the times that a particular rule can be fired. This means that if a rule m is valid in at least one time slot and if the assignment/revocation (or enabling/disabling) operation that is going to be performed m is necessary for the other rules m', one can wait until m becomes valid, and perform the necessary operation. The other rules m' can be fired next time when the system periodically repeats itself. For example, suppose that we have two roles, r_1 and r_2 and two `t_can_assign` rules $(\ldots, (4, 10), \{\}, r_1, \ldots)$ and $(\ldots, (1, 3), \{r_1\}, r_2, \ldots)$. The first rule states that we can use it only within $(4, 10)$; the second rule states that we can only use it within $(1, 3)$. Notice that if the rules are serially applied with respect to time, then since the second rule has a precondition of r_1, we cannot fire second rule if we do not have r_1 already assigned. It means that first we need to wait until first rule becomes valid (until $t = 4$) and assign r_1. Then we should wait until the system restarts from $t = 0$ (since it is periodic) to fire second rule. Then the Long Run Behavior property ensures that for the reachability analysis purposes, if one waits sufficient amount of time then the effects of these kind of rule conflicts can safely be neglected. This property allows us to treat all of the rules valid on the entire time line. Hence, the s_{rule} restrictions can be relaxed from the rules.

In order to handle the independency issues among different time slots, we need to consider preconditions. Recall that we define the preconditions as (*Pos*, *Neg*) relations to be satisfied in order to execute a rule. Now consider a rule $m \in \mathcal{M}$ which

belongs to $\mathcal{T}(\alpha, \mathcal{M})$, and $\hat{s} = \alpha$. In order to execute m, the precondition relations declared by (*Pos*, *Neg*) of m must be satisfied for $\hat{s}$. For each role *pos* $\in$ *Pos* (*neg* $\in$ *Neg*, resp.) $\hat{s} \subseteq s_{pos}$ ($\hat{s} \cap s_{pos} = \emptyset$, resp.) must be satisfied, which simply depends on the corresponding (single) time slot in s_{pos} (s_{neg}, resp.). Then it is sufficient to check the schedule only for time slot α for each rule. This implies that the preconditions do not depend on other time slots, hence the time slots are independent.

So, using the Long Run Behavior property and the independency of time slots, one can perform an RBAC reachability analysis using the rules $m \in \mathcal{M}$ for time slot α. Then, the whole TRBAC system can be analyzed by a series of independent RBAC systems $\mathcal{T}_i$ traced separately. This reduction provides usability of any RBAC reachability analysis procedure proposed in the literature.

The computational complexity of the algorithm depends on the RBAC analyzer. Suppose that the RBAC analyzer has the complexity O(·) then our approach yields a complexity of O(T_{MAX}·) since we utilize the RBAC analyzer for each time slot (totally we have T_{MAX} of them). Since the algorithm runs for T_{MAX} iterations and given that the RBAC analyzer terminates, our algorithm is guaranteed to terminate.

4.2.3. Modified SA for hierarchy analysis

The RBAC Analysis algorithm proposed by Stoller et al. [30] is a state space exploration algorithm which is proved to be fixed parameter tractable. In our decomposition approach, the subproblems obtained by the decomposition can be analyzed by SA for Role Enabling, User to Role and Permission to Role assignment relations. However, due to the precondition structure and SA not capable of handling the `can_modify` rule, SA is unable to analyze the Temporal Role Hierarchy subproblem. In this section, we make certain modifications on SA to fit the requirements of the role hierarchy analysis strategy that we propose for a `TURA` analysis instance. We call this modified algorithm as MSA (given in Algorithm 1), which is still a state space exploration algorithm, specifically designed for role hierarchies. The purpose of MSA is to generate different possible static role hierarchies given a set of `t_can_modify` rules. This algorithm can be used in both Rule Schedule and Role Schedule strategies.

The state space is composed of the TRBAC configurations c, represented by *DTRH*, generated by moves m, and authorized by the rules $\mathcal{M}$. In the configurations, the precondition statements are crucial to determine the relationship among different rules. A role is *hierarchy negative*, if it appears negated in either junior or senior preconditions of a `t_can_modify` rule. The other roles are called *hierarchy non-negative*. A role is *hierarchy positive*, if it appears positive in either junior or senior preconditions of a `t_can_modify` rule. The other roles are called *hierarchy non-positive*. Any move m related to a DTRH policy with hierarchy non-negative or hierarchy non-positive roles is called an invisible transition, the others are called visible transition. Any invisible transition that creates a conflict with the anti-symmetric property of DTRH in Section 3.2 generates a new state. Any visible transition that creates a conflict with the anti-symmetric property of DTRH in Section 3.2 is prohibited.

Algorithm 1. The modified Stoller et al.'s Algorithm (MSA)

```
1: Set S_T = {c_0} as temporary, S_P = ∅ as permanent set
2: Determine the non-positive and non-negative roles
3: while S_T ≠ ∅ do
4:    Get a state c ∈ S_T
5:    Create a temporary state c_temp = c
6:    for all Rules m ∈ S_T that generate an invisible transition do
7:       Check for hierarchy conflicts in c_temp
8:       if There exists any violation then
9:          Create a new state c'
10:         Apply rule m on c'
11:         Set S_T = S_T ∪ c'
12:      else
13:         Apply rule m on c_temp
14:      end if
15:   end for
16:   Set c = c_temp
17:   for all Rules m ∈ S_T that generate a visible transition do
18:      Create a new state c'
19:      Check for hierarchy conflicts in c'
20:      if There exists any violation then
21:         Discard c'
22:      else
23:         Set S_T = S_T ∪ c'
24:      end if
25:   end for
26:   Set S_T = S_T \ c
27:   Set S_P = S_P ∪ c
28: end while
```

In the analysis for role hierarchies, there is no goal state to be achieved, rather all possible hierarchy configurations are constructed to be used to interpret the implicit role assignments of the other steps of the analysis.

4.3. Stage 3: *Interpretation of the results*

The final step of the security analysis is to interpret the results obtained to conclude whether the access control configuration is vulnerable based on the analysis of interest. In our analysis methodology, each step of the four step analysis procedure outputs results for a different relation in TRBAC. However, these results are not sufficient individually to answer the security questions. The results of different steps

of the analysis should be utilized together to obtain the correct result. For instance, Role Assignment analysis could state that the goal role would be assigned to the target user, but that role might not get enabled at that time instance, meaning that it is not possible for that particular user to exercise the goal role. This step is crucial to interpret the security properties correctly.

Suppose that all four steps of the analysis is done. Each step outputs a set of state configurations denoted as C_1, C_2, C_3 and C_4 respectively for the four steps. Each configuration $c \in C_1$ is composed of *TUA*, $c \in C_2$ is composed of *RS*, $c \in C_3$ is composed of *DTRH* and $c \in C_4$ is composed of *DTRH* policies. For notational simplicity, we denote the relations as configurations. Under these settings a given TRBAC policies and rules create a security violation if they satisfy the following criteria for different security questions of interest:

- Explicit Role Assignment: $\exists \mathit{TUA} \in C_1, \mathit{RS} \in C_2$: $(u, r, s_1) \in \mathit{TUA} \wedge (r, s_2) \in \mathit{RS} \wedge s_1 \cap s_2 \neq \emptyset$.
- Implicit Role Assignment: $\exists \mathit{TUA} \in C_1, \mathit{RS} \in C_2, \mathit{DTRH} \in C_3$: $(u, r_1, s) \in \mathit{TUA} \wedge (r_1 \langle \mathcal{F} \rangle_{s_{i_1}} r_2), \ldots, (r_n \langle \mathcal{F} \rangle_{s_{i_n}} r) \in \mathit{DTRH} \wedge (r_1, s_{j_1}), (r_2, s_{j_2}), \ldots, (r, s_{j_n}) \in \mathit{RS} \wedge s \cap s_{i_1} \cap \cdots \cap s_{i_n} \cap s_{j_1} \cap \cdots \cap s_{j_n} \neq \emptyset$.[2]
- Role Enabling: $\exists \mathit{RS} \in C_2$: $(r, s) \in \mathit{RS} \wedge s \neq \emptyset$.
- Permission Assignment: $\exists \mathit{TPA} \in C_4$: $(p, r, s) \in \mathit{TPA} \wedge s \neq \emptyset$.
- Liveness for Explicit Role Assignment: $\forall s_1, s_2, \nexists \mathit{TUA} \in C_1, \mathit{RS} \in C_2$: $(u, r, s_1) \notin \mathit{TUA} \vee (r, s_2) \notin \mathit{RS}$.
- Mutual Exclusion for Explicit Role Assignment: $\exists \mathit{TUA} \in C_1, \mathit{RS} \in C_2$: $(u_1, r, s_1) \in \mathit{TUA} \wedge (u_2, r, s_2) \in \mathit{TUA} \wedge (r, s_3) \in \mathit{RS} \wedge s_1 \cap s_2 \cap s_3 \neq \emptyset$.

5. Computational experiments

We have performed computational experiments for the analysis of TRBAC using Rule and Role Schedule Approaches. In our experiments we demonstrated the performance of the Role Assignment (Step 1) and Role Hierarchy (Step 3), since the other steps are analogous to Step 1. In the experiments we employ SA and MSA for Role Assignment and Role Hierarchy components.

We implement our algorithm with C programming language and run it on a computer with 3 GB RAM and Intel Core2Duo 3.0 GHz processor running Debian Linux operating system. In the experiments, the initial state is set to be an empty state (meaning that there are no role assignments), and the rules and the goal are created randomly by the code with respect to the corresponding parameter values for the number of rules, number of roles, number of time slots and the number of cycles. As we discussed before, we assume separate administration. Also, for role hierarchies,

[2]Depending on the type of role hierarchy, role enabling criteria must satisfy the DTRH properties given in Definitions 4–7.

Table 4
Parameter settings

Number of roles $\|R\|$	100, 500, 900
Number of rules $\|\mathcal{M}_{ALL}\|$	100, 500, 900
Number of time slots T_{MAX}	100, 500, 900
Number of cycles P	30 for all cases

we assume a general hierarchy relation. The parameter settings are shown on Table 4. 10 replications are done for each parameter setting and their average is reported. The results are in Fig. 4(a), (b) and (c).

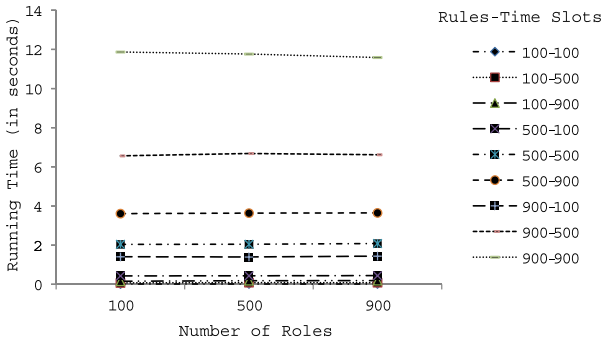
5.1. User to role assignment experiments

The complexity of the *rule schedule approach* algorithm depends not only on the number of roles and rules but also depends on the number of time slots, and the schedules (rule–role) that are assigned to the roles. The state space that is generated by this algorithm tends to be exponential in the worst case since it is a brute force state space exploration algorithm.

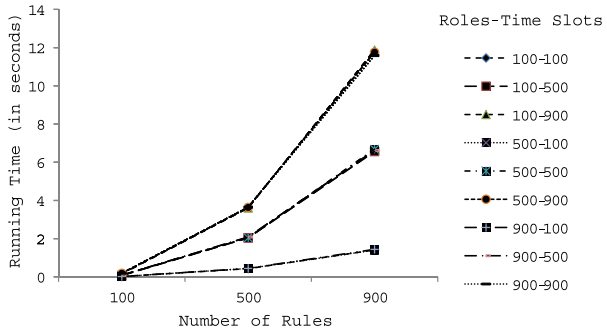
According to the results obtained for the rule schedule approach, the run time performances of the algorithms do not tend to be exponential, especially for the number of roles. A possible explanation to this situation is that the datasets are generated randomly. Hence there does not exist any "pattern" among the rules. We mean pattern in the sense that, the components that determine the usability of the rules, i.e., all of the precondition relations, rule and role schedules of the moves are generated randomly – so it might become probabilistically harder to satisfy all of these conditions. Nevertheless, the results give some insight about how the algorithm is likely to behave under different parameter settings.

The effect of number of rules while all other parameters are constant is more significant and tends to be an increasing relationship as number of rules increases (see Fig. 4(b)). Moreover, the increasing tendency becomes more significant as the number of roles and number of time slots increase. Furthermore, there is a noticeable group formation between the fixed parameters (number of roles and number of time slots). The groups are formed by different number of time slots values indicating that the effect of number of roles is comparably smaller. Finally, Fig. 4(c) denotes the relationship between different values of number of time slots parameter when the other two parameters are kept constant. The results show that for the majority of the cases, there is a linearly increasing relationship with the increasing number of rules.

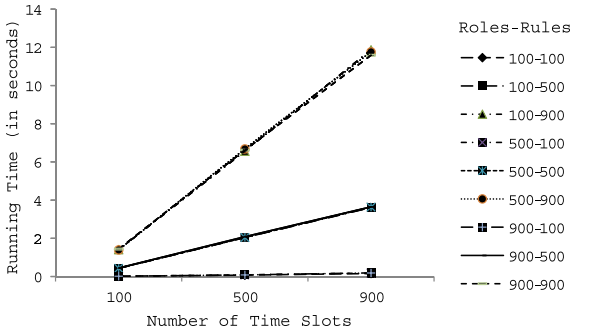
For the *role schedule approach*, we use SA. According to the results obtained, there is a linear and increasing relationship with 100, 500 and 900 roles in the system while all other parameters are constant (see Fig. 5(a)). The effect of number of rules while all other parameters are constant is very similar to the effect of roles. There is an increasing relationship in the running time as the number of rules increases (see Fig. 5(b)).



(a)

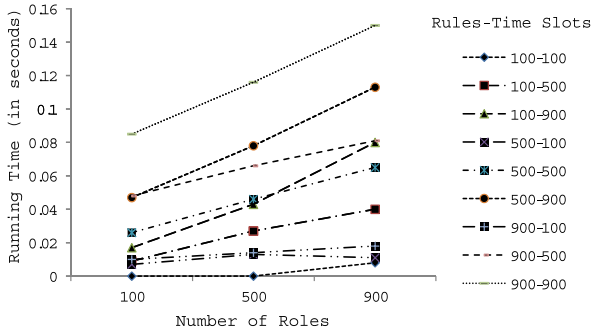


(b)

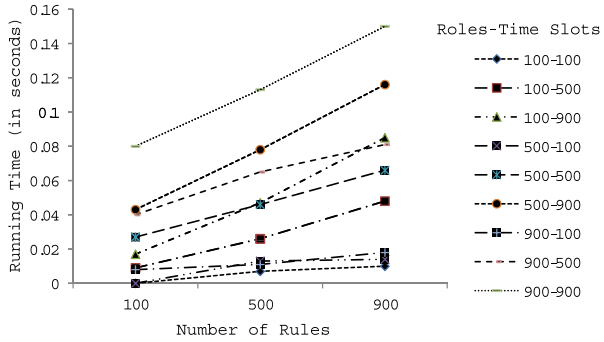


(c)

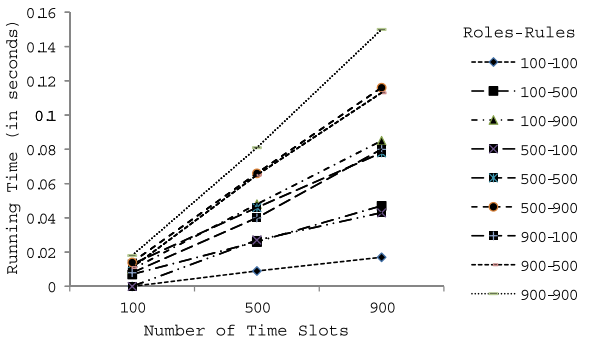
Fig. 4. Rule schedule approach for role assignment. (a) Effect of number of roles. (b) Effect of number of rules. (c) Effect of number of time slots. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-140510.)



(a)



(b)



(c)

Fig. 5. Role schedule approach for role assignment. (a) Effect of number of roles. (b) Effect of number of rules. (c) Effect of number of time slots. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-140510.)

Finally, Fig. 5(c) denotes the relationship between different values of number of time slots parameter when the other two parameters are kept constant. The results show that there is a linearly increasing behavior as the number of time slots increase. This result is expected since the complexity of the algorithm linearly depends on this parameter.

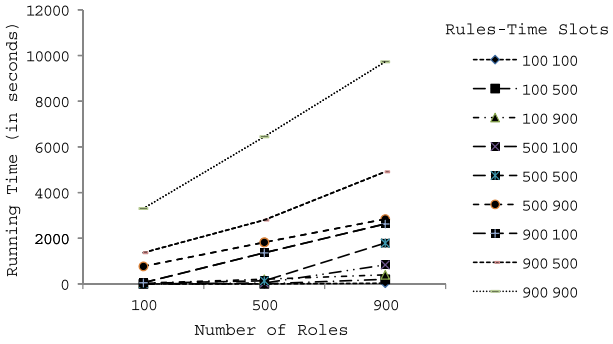
5.2. Role hierarchy experiments

In the role hierarchy experiments, we observe that the running times of both of the approaches increased significantly. Especially for higher parameter settings for Rule Schedule Approach, running times of 10,000 s, as opposed to a maximum of 12 s for User to Role Assignment experiments are observed. The underlying reasoning for this drastic increase is the fact that the state space consists of a pair of roles. Moreover, the process of determining whether an intended update in any of the role hierarchy pairs require examining the existing role hierarchy pairs to make sure that the newly imposed changes will not create a conflict.

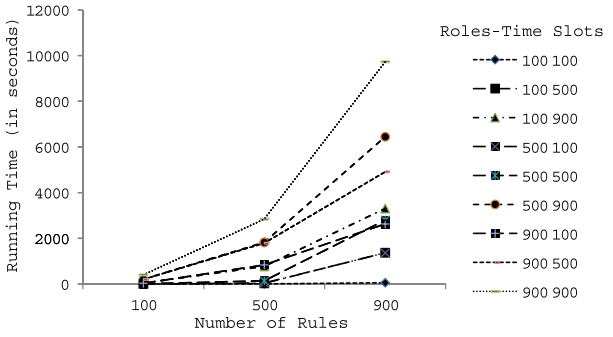
When the run time performances of rule schedule and role schedule approaches are compared, a similar pattern as in the User to Role Assignment experiments is observed. Role Schedule approach is significantly faster than the Rule Schedule approach due to the fact that the Rule Schedule approach is an exponential state space exploration algorithm. The experimental results are given in Fig. 6(a), (b) and (c) for Rule Schedule and Fig. 7(a), (b) and (c) for Role Schedule approach.

6. Temporal role hierarchies execution model

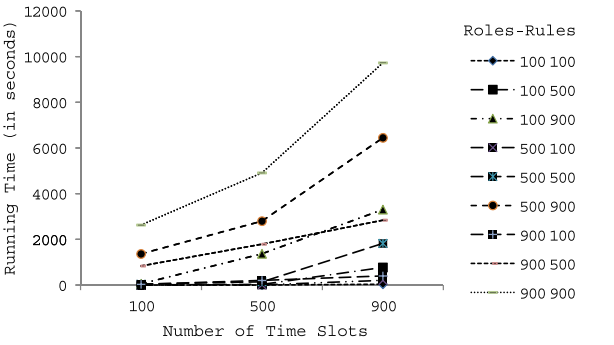
The dynamic temporal role hierarchy definition theoretically allows the access control system to have a different hierarchy at each different time slot, hence users can potentially acquire a totally different set of roles and permissions in each of these slots. Recall that, the role hierarchy set is composed of role hierarchy policies. In fact, these policies create a tree structure with roles as nodes and the policies as the directed edges. So, the hierarchy can also be represented as a tree. In an application perspective, it is necessary to determine exactly how the temporal role hierarchies are represented in the system. There are two different ways: (1) A *separate complete hierarchy* tree for each time slot. Then, the role/permission acquisition at each time slot can be determined by tracing the complete role hierarchy tree of that particular time slot. (2) Retaining the *Hierarchy Policies* with embedded schedules, and the role/permission acquisition decisions are made on demand. Both of these approaches are useful under different circumstances. Now, we provide an insight about when to use which representation to answer a query asking whether a role is senior to another role in a given time slot. Having a separate complete role hierarchy at each time slot provides faster response to any query that checks for an implicit assignment. A simple search (like depth-first search) done on this tree will provide an efficient answer



(a)

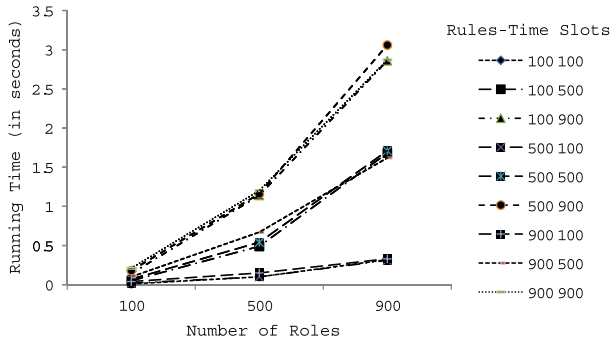


(b)

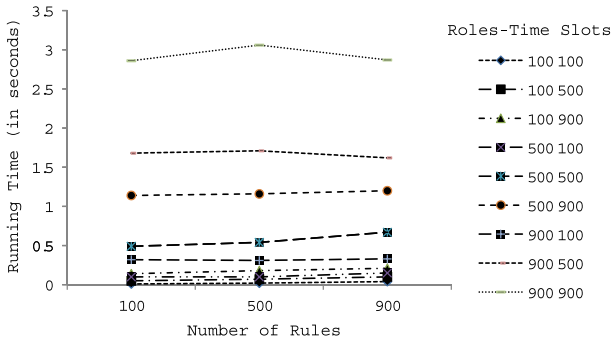


(c)

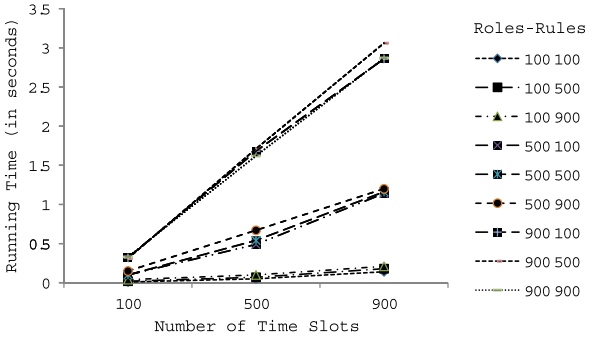
Fig. 6. Rule schedule approach for role hierarchy. (a) Effect of number of roles. (b) Effect of number of rules. (c) Effect of number of time slots. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-140510.)



(a)



(b)



(c)

Fig. 7. Role schedule approach for role hierarchy. (a) Effect of number of roles. (b) Effect of number of rules. (c) Effect of number of time slots. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-140510.)

in O($|R| \log |R|$) time. On the other hand, a search in the partial hierarchies require an exponential O($|DTRH|^{|DTRH|}$) time. However, the partial hierarchies can be beneficial if the system faces many alterations in the role hierarchies. In this case a policy change for a single time slot requires O($|DTRH|$) time for the partial hierarchies, but O($|R| \log |R|$) for separate complete hierarchy.

7. Related work

The pioneering works for the security analysis of policies are done for protection schemes with Discretionary Access Control, which is usually composed of an access control matrix and some operators to modify the scheme. Harrison et al. [14] propose a formal model for protection systems, which shows that there is an algorithm which decides whether or not a given mono-operational protection system and initial configuration is unsafe for a given generic right. However, it is undecidable whether a given configuration of a given protection system is safe for a given generic right. Jones et al. [16] state that the safety analysis of whether a user will gain an access right can be answered in linear time (for a specific class of simple policies). Sandhu [26] proposes the Schematic Protection Model (SPM) that has a high expressive power and provides an analysis which is both decidable and tractable only if the model is acyclic and attenuating. Ammann et al. [3] propose ESPM to address the limitations of SPM. In fact, the main outcome of the paper is that, it proves that ESPM is equivalent to HRU. The benefits of having strong typing in the access control schemes as depicted in SPM model can also be embedded into the basic HRU model. Sandhu [27] proposes Typed Access Matrix (TAM) model to address this issue and shows that HRU is a special case of TAM. Soshi [29] provides an extension of TAM, called Dynamic TAM (DTAM), in which changes in object types are allowed and allowing a nonmonotonic scheme and removing the restriction of strong typing can also provide a decidable safety analysis under certain conditions.

For RBAC, there are some studies exploring the security analysis. Li and Tripunitara [21,22] develop the first approach to security analysis in RBAC. Jha et al. [15] state that the security analysis problem on URA with a simple query of whether a user is a member of a particular role is PSPACE-Complete. Stoller et al. [30] consider analyzing the security problem in a parameterized complexity environment. The algorithm provided for analysis is said to be fixed parameter tractable with respect to the number of roles. Ferrara et al. [12] proposes a set and numerical abstraction based reduction of ARBAC97 policies into programs, so that a program verification tool can be used to check the security properties. According to the results they obtain, the model scales well to analyze security properties of large ARBAC policies.

The first model that embeds temporal data to access control is proposed by Bertino et al. [7] and called the Temporal Authorization Model (TAM). The model is basically built on the Discretionary Access Control model using discrete time. Atluri and Gal [4] propose another model that embeds the temporal notion into access control.

The first temporal model developed on RBAC – Temporal RBAC – is proposed by Bertino et al. [8] that has periodic role enabling and role triggers. Joshi et al. [20] propose Generalized Temporal RBAC model which considers Temporal constraints on role assignments, role activations, enabling and disabling constraints (like cardinality constraints), and temporal role hierarchies and SOD constraints in addition to Temporal RBAC. Mondal et al. [23] provide a security analysis for Generalized Temporal RBAC using timed automata to verify the safety and liveness security properties. This real time verification process is PSPACE-Complete. The important observation is that the verification process has a state space explosion for large number of users.

The work in this paper builds upon our prior work in [31,32]. Uzun et al. [31] provides an analysis for Temporal RBAC model that considers only time based decomposition for user to role assignment and role enabling relations. Uzun et al. [32] introduces the problem of security analysis for Dynamic Temporal Role Hierarchies. In this paper, we present a comprehensive approach that takes all the components of TRBAC into account and experimentally validate it with real data sets.

8. Conclusions and future work

Security analysis is vital for access control systems to capture any vulnerability that the incorrectly configured policies might cause. In this paper, we emphasize this analysis on the temporal extension of RBAC. Although there are models for the temporal extension of RBAC proposed in the literature before, none of them has an extensive analysis that captures temporal user to role and permission to role assignments, as well as temporal role hierarchies and role enabling all together. We propose an administrative model that is capable of handling authorized changes on the temporal policies. The security analysis methodology that we propose is structurally flexible to adopt itself to various different security analysis purposes as to answer different security questions of interest. Our three stage analysis procedure decomposes the analysis into relation based subproblems as well as time based sub-subproblems to obtain RBAC-like analysis problems that are easier to handle. In addition to this, we also propose an approach to analyze changes in role hierarchy in the presence of `can_modify` type administrative rules. We demonstrate the run time performances of these approaches on randomly generated data sets to show the effects of different parameters on the running times.

Our future work is to further enhance our analysis with respect to its performance, by providing an *incremental security analysis*. It is clear that the complexity of the problems affects the running times of the analysis algorithms. An incremental analysis enables faster analysis to minor modifications (introducing a new rule or a new role) done on already analyzed security problems by utilizing the previously generated state space. The help of the recycled states will eventually facilitate the analysis by generating fewer new states when compared to a security analysis with an empty initial state space.

References

[1] R. Agrawal, T. Imielinski and A. Swami, Mining association rules between sets of items in large databases, in: *Proceedings of the 1993 ACM SIGMOD Conference*, 1993.

[2] S. Aich, S. Sural and A.K. Majumdar, Starbac: spatiotemporal role based access control, in: *Proceedings of the 2007 OTM Confederated International Conference on the Move to Meaningful Internet Systems: CoopIS, DOA, ODBASE, GADA, and IS, OTM'07*, Part II, Springer, Berlin/Heidelberg, 2007, pp. 1567–1582.

[3] P. Ammann and R. Sandhu, Safety analysis for the extended schematic protection model, in: *IEEE Symposium on Security and Privacy*, 1991, pp. 87–97.

[4] V. Atluri and A. Gal, An authorization model for temporal and derived data: securing information portals, *ACM Trans. Inf. Syst. Secur.* **5**(1) (2002), 62–94.

[5] E. Barka and R. Sandhu, Framework for role-based delegation models, in: *Computer Security Applications, 2000. ACSAC'00. 16th Annual Conference*, IEEE, 2000, pp. 168–176.

[6] E. Barka, R. Sandhu et al., A role-based delegation model and some extensions, in: *Proceedings of the 23rd National Information Systems Security Conference*, Vol. 4, 2000, pp. 49–58.

[7] E. Bertino, C. Bettini and P. Samarati, A temporal authorization model, in: *Proceedings of the 2nd ACM Conference on Computer and Communications Security, CCS'94*, 1994, pp. 126–135.

[8] E. Bertino, P. Bonatti and E. Ferrari, TRBAC: A temporal role based access control model, *ACM Transactions on Information and System Security* **4**(3) (2001), 191–233.

[9] J. Crampton and H. Khambhammettu, Delegation in role-based access control, in: *Computer Security – ESORICS 2006*, 2006, pp. 174–191.

[10] D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn and R. Chandramouli, Proposed NIST standard for role-based access control, in: *TISSEC*, 2001.

[11] D.F. Ferraiolo, D.R. Kuhn, R. Chandramouli and J. Barkley, Role-based access control (RBAC), in: *15th National Computer Security Conference*, 1992, pp. 554–563.

[12] A.L. Ferrara, P. Madhusudan and G. Parlato, Security analysis of access control policies through program verification, in: *25th IEEE Computer Security Foundations Symposium*, 2012.

[13] Q. Guo, J. Vaidya and V. Atluri, The role hierarchy mining problem: Discovery of optimal role hierarchies, in: *Computer Security Applications Conference, 2008. ACSAC 2008. Annual*, IEEE, 2008, pp. 237–246.

[14] M.A. Harrison, W.L. Ruzzo and J.D. Ullman, Protection in operating systems, *Commun. ACM* **19**(8) (1976), 461–471.

[15] S. Jha, N. Li, M. Tripunitara, Q. Wang and W. Winsborough, Towards formal verification of role-based access control policies, *IEEE Trans. Dependable Secur. Comput.* **5**(4) (2008), 242–255.

[16] A.K. Jones, R.J. Lipton and L. Snyder, A linear time algorithm for deciding security, in: *Proceedings of the 17th Annual Symposium on Foundations of Computer Science, SFCS'76*, IEEE Computer Society, Washington, DC, USA, 1976, pp. 33–41.

[17] J. Joshi and E. Bertino, Fine-grained role-based delegation in presence of the hybrid role hierarchy, in: *Proceedings of the Eleventh ACM Symposium on Access Control Models and Technologies*, 2006, pp. 81–90.

[18] J. Joshi, E. Bertino and A. Ghafoor, Hybrid role hierarchy for generalized temporal role based access control model, in: *26th Annual International Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings*, IEEE, 2002, pp. 951–956.

[19] J. Joshi, E. Bertino and A. Ghafoor, Temporal hierarchies and inheritance semantics for GTRBAC, in: *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*, ACM, 2002, pp. 74–83.

[20] J. Joshi, E. Bertino, U. Latif and A. Ghafoor, A generalized temporal role based access control model, *IEEE Transactions on Knowledge and Data Engineering* **17**(1) (2005), 4–23.

[21] N. Li and M.V. Tripunitara, Security analysis in role-based access control, in: *Proceedings of the Ninth ACM Symposium on Access Control Models and Technologies, SACMAT'04*, ACM, New York, NY, USA, 2004, pp. 126–135.
[22] N. Li and M.V. Tripunitara, Security analysis in role-based access control, *ACM Transactions on Information and System Security* **9**(4) (2006), 391–420.
[23] S. Mondal, S. Sural and V. Atluri, Towards formal security analysis of GTRBAC using timed automata, in: *ACM Symposium on Access Control Models and Technologies*, 2009, pp. 33–42.
[24] R. Sandhu et al., Role-based access control models, *IEEE Computer*, February 1996, pp. 38–47.
[25] R. Sandhu, V. Bhamidipati and Q. Munawer, The ARBAC97 model for role-based administration of roles, *ACM Transactions on Information and System Security* **2**(1) (1999), 105–135.
[26] R.S. Sandhu, The schematic protection model: its definition and analysis for acyclic attenuating schemes, *J. ACM* **35**(2) (1988), 404–432.
[27] R.S. Sandhu, The typed access matrix model, in: *1992 IEEE Computer Society Symposium on Research in Security and Privacy, 1992, Proceedings*, IEEE, 1992, pp. 122–136.
[28] A. Schaad, J. Moffett and J. Jacob, The role-based access control system of a European bank: A case study and discussion, in: *Proceedings of ACM Symposium on Access Control Models and Technologies*, May 2001, pp. 3–9.
[29] M. Soshi, Safety analysis of the dynamic-typed access matrix model, in: *Computer Security – ESORICS 2000*, Lecture Notes in Computer Science, Vol. 1895, Springer, Berlin/Heidelberg, 2000, pp. 106–121.
[30] S.D. Stoller, P. Yang, C.R. Ramakrishnan and M.I. Gofman, Efficient policy analysis for administrative role based access control, in: *ACM*, 2007, pp. 445–455.
[31] E. Uzun, V. Atluri, S. Sural, J. Vaidya, G. Parlato, A.L. Ferrara and M. Parthasarathy, Analyzing temporal role based access control models, in: *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies, SACMAT'12*, ACM, New York, NY, USA, 2012.
[32] E. Uzun, V. Atluri, J. Vaidya and S. Sural, Analysis of TRBAC with dynamic temporal role hierarchies, in: *Data and Applications Security and Privacy XXVII*, Springer, 2013, pp. 297–304.
[33] J. Wainer and A. Kumar, A fine-grained, controllable, user-to-user delegation method in RBAC, in: *Symposium on Access Control Models and Technologies: Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies*, Vol. 1, 2005, pp. 59–66.
[34] L. Zhang, G. Ahn and B. Chu, A rule-based framework for role-based delegation and revocation, *ACM Transactions on Information and System Security (TISSEC)* **6**(3) (2003), 404–441.
[35] X. Zhang, S. Oh and R. Sandhu, Pbdm: a flexible delegation model in RBAC, in: *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies*, ACM, 2003, pp. 149–157.