Noname manuscript No.
(will be inserted by the editor)

Security policy verification for multi-domains in cloud systems

Antonios Gouglidis · Ioannis Mavridis · Vincent C. Hu

Received: date / Accepted: date

Abstract The cloud is a modern computing paradigm with the ability to support a business model by providing multitenacy, scalability, elasticity, pay as you go and self provisioning of resources by using broad network access. Yet, cloud systems are mostly bounded to single domains and collaboration among different cloud systems is an active area of research. Over time, such collaboration schemas are becoming of vital importance since they allow companies to diversify their services on multiple cloud systems to increase both up-time and usage of services. The existence of an efficient management process for the enforcement of security policies among the participating cloud systems would facilitate the adoption of multi-domain cloud systems. An important issue in collaborative environments is secure inter-operation. Stemmed from the absence of relevant work in the area of cloud computing, we define a model checking technique that can be used as a management service/tool for the verification of multi-domain cloud policies. Our proposal is based on NIST's (National Institute of Standards and Technology) generic model checking technique and has been enriched with RBAC reasoning. Current approaches, in Grid systems, are capable of verifying and detect only conflicts and redundancies between two policies. However, the latter cannot overcome the risk of privileged user access in multi-domain cloud systems. In this paper, we provide the formal definition of the proposed technique and security properties that have to be verified in multi-domain cloud systems. Furthermore, an evaluation of the technique through a series of performance tests is provided.

Keywords cloud computing · collaboration · multi-domain · RBAC · secure inter-operation · verification

A. Gouglidis (✉) · I. Mavridis
Department of Applied Informatics
University of Macedonia
156 Egnatia Str., 54006, Thessaloniki, Greece
E-mail: agougl@uom.gr

I. Mavridis
E-mail: mavridis@uom.gr

V.C. Hu
National Institute of Standards and Technology
Gaithersburg, MD 20899-8930, USA
E-mail: vincent.hu@nist.gov

1 Introduction

Access control (AC) in modern distributed systems has become even more challenging since they are complicated and require the collaboration among domains. A domain can be defined as a protected computing environment, consisted of users and resources under a same AC policy. AC is an essential process in all systems. The role of an AC system is to control and limit the actions or operations in a system that are performed by a user on a set of resources. Nevertheless, an AC system is considered of three abstractions of control, namely AC policies, AC models, and AC mechanisms. A policy can be defined as a high-level requirement that specifies how a user may access a specific resource and when. AC policies can be enforced in a system through an AC mechanism that is responsible for permitting or denying a user access upon a resource. An AC model can be defined as an abstract container of a collection of AC mechanism implementations, which are capable of preserving support for the reasoning of the system policies through a conceptual framework. Consequently, the AC model is capable of bridging the existing abstraction gap between the mechanism and the policy in a system [9], [48].

The cloud is a fairly new and emergent technology and its definition is a topic for discussion in several research papers [14]. Nevertheless, cloud computing is defined in [31] using five attributes viz. multitenancy, massive scalability, elasticity, pay as you go and self-provisioning of resources. These attributes successfully imprint the distinctive characteristics of the cloud and differentiate it from similar technologies, as the Grid computing paradigm. Multitenancy refers to the business model implemented by the cloud, where a single shared resource can be used from multiple users. Massive scalability refers to the potential of the system to scale (i.e. increase or decrease) in resources. The on-demand and rapid increment or decrement of computing resources is translated as elasticity of the cloud. Thus, more storage space or bandwidth can be allocated when required, and vice versa. Pay as you go is the process of paying for the resources that are used. Lastly, the users are provided with the ability to self-provision resources, namely storage space, processing power, network resources and so on. An additional characteristic defined in [43] by the National Institute of Standards and Technology (NIST) is Broad Network Access, which states that available capabilities can be accessed using standard mechanisms over the network, and promote their use by heterogeneous clients.

The service model of cloud computing is based on the SPI framework [31], [43]. SPI stands for Software-as-a-service (SaaS), Platform-as-a-service (PaaS) and Infrastructure-as-a-service (IaaS). Specifically, the SaaS provides software that is used under a business model, namely the usage-based pricing. The PaaS offers the platform for the development of the applications, and lastly, the IaaS handles the provision of the required hardware, software and equipment, in order to deliver a resource usage-based pricing model. Moreover, the aforementioned service models are provided under three deployment models viz. public, private and hybrid cloud [31]. The public cloud provision resources over the Internet and are accessible via a web application. A third-party operates as the host and performs all the required operations (e.g. management, security). The private cloud provides the same functionality as the public deployment model within internal and private networks. This model requires the acquisition of the appropriate hardware and software. The hybrid model refers to the combination of the public and private deployment models. Usually, the latter model is used to keep sensitive data in the private network and deploy non-core applications to the public. An additional service model proposed by NIST is the community cloud [43], which refers to infrastructure exclusive used by a specific community of consumers from organizations that have shared concerns.

Currently, collaboration among different cloud systems is a challenge since in addition to heterogeneity issues among different domain policies that must be addressed, it must also be ensured that collaborations are handled securely and that security breaches are effectively monitored during the secure interoperation process [53]. The work described in the following is motivated by the desire to be able to ensure that security requirements hold in the presence of dynamic coalitions [8] (i.e. virtual organizations or virtual enterprises). Similar research work in [44] identified the need for performing conformance checking of dynamic access control policies since even when the security requirements are well understood for an AC policy, it is still possible for mistakes to be made. Therefore, formal and tool-supported approaches for designing and maintaining access control policies in dynamic coalitions are required, as examined in [8]. Our research provides techniques that will facilitate multi-domain cloud collaborations. Specifically, we investigated a management service/tool that will maintain secure inter-operability among cloud systems. Our application of secure inter-operability overcomes the risk of privileged user access in multi-domain cloud systems, as stated and required in [14]. Privileged user access is concerned with the assurance that sensitive data processed outside the enterprise must be only accessible and propagate to privileged users. In addition, we demonstrated a management service/tool that is able to successfully verify the correctness of the implemented AC mechanism based on the requirement in [9], which states that a system can be argued to be secure only if the model is secure and the mechanism correctly implements the model.

RBAC has received considerable attention from researchers for its capabilities of abstraction, generalization and support of various access control principles [36]. Also, it is argued to be best suited for government agencies where well-defined organizational hierarchies exist for grouping responsibilities and privileges into roles [34]. Additionally, RBAC copes well with entrepreneurial applications in cloud computing systems where users or applications can be clearly separated according to their job functions [30]. Furthermore, as stated in [5], RBAC is a key technology for cloud infrastructures, well-suited for multi-domain architecture, and applicable in cloud systems relevant to health records, stock trading and pairing, and social networking. As a result, the RBAC model is employed by most of the cloud computing platforms, namely, OpenStack [41], [42], Xen [10], Windows Azure [32] and so on. Also, research in [33] examined a realistic medical scenario using a middleware that enforces security policy in multi-domain applications, which further gives rise to verification of AC policies in multi-domain cloud environments.

Therefore, driven by the need for supporting the requirements of multi-domain cloud collaborations and application of RBAC, we examine in the rest of this paper the usage of a RBAC model capable of supporting and enforcing secure inter-operation of RBAC policies in collaborative systems (i.e. domRBAC [16]).

The structure of the remainder of this paper is: Section 2 provides preliminary information regarding the RBAC model, secure inter-operation, and the applied technique. Section 3 describes our proposed management service/tool based on the verification of security properties in RBAC policies. Implementation aspects are discussed in Section 4. The performance of the applied technique and a number of issues are discussed in Section 5. Section 6 presents related work. Finally, we conclude this paper in Section 7.

2 Preliminaries

This section provides prerequisite information about the Role Based Access Control (RBAC) model since it is being used as our basis model for achieving collaboration among cloud systems. Secure inter-operation is being discussed in the context of RBAC since it is an issue of vital importance in collaborative environments. Furthermore, a brief information is provided regarding the applied model checking technique for the verification of secure inter-operation properties.

2.1 Role Based Access Control

The RBAC model has received considerable attentions from researchers for its capabilities of abstraction and generalization. Abstraction, because it includes only properties that are relevant to security, and generalization, because many designs could be considered valid interpretations of the model [12]. In addition, RBAC supports various AC principles, such as Least Privilege, and Separation of Duties (SoD)/Administrations [47]. The RBAC model consists of four components with different functionalities. The components are Core RBAC, Hierarchical RBAC, Static Separation of Duty (SSD), and Dynamic Separation of Duty (DSD).

The core RBAC model, depicted in Figure 1, has five static elements: users, roles, and permissions, which contain operations and objects. The relations between the elements are straightforward. Roles are assigned to users and permissions are assigned to roles, and the mapping of relations between them are many-to-many (i.e. one user can be assigned to many roles) and many users can be assigned to one role. The same applies to the role to permission assignment. However, negative permissions are not supported in RBAC. This indirect assignment of users to permissions greatly enhances the administration in RBAC, and revocation of assignments can be done easily. Moreover, we distinguish design and run-time phases in RBAC implementation. System administrators define assignments between the elements in the design phase, and the model enforces the assignments in the run-time phase.

The run-time phase is achieved by the concept of the session. This unique (among other group-based AC mechanisms) feature allows a set of users' roles to be activated. This means a user could be assigned to various roles during the design phase, but do not need to be always or simultaneously activated (by the principle of least privilege). However, the capability of sessions has been questioned with suggestion of replacement for them [29].

The Hierarchical RBAC enhances administration flexibility through the capability of permission (operations to objects) inheritance. Permissions (assigned to a role) can be inherited to another role through hierarchical relation assignments without reassigning the same permissions to the inherit role. For instance, let's assume two roles r_1 and r_2 and two permission sets $PRMS_1 = (p_1, p_2)$ and $PRMS_2 = (p_3, p_4)$, which are initially assigned to roles r_1 and r_2, respectively. Role r_1 inherits role r_2 means all permissions of r_2 are also available to r_1. The inherited permission can be expressed by the union of $PRMS_1$ and $PRMS_2$. The immediate inheritance relation is denoted by the $\rightarrow$, for example, $r_1 \rightarrow r_2$. User membership refers to the assignment of users to roles in a hierarchy, thus, users are authorized to have all the permissions assigned to roles either directly or via inheritance. The Hierarchical RBAC supports general and limited role hierarchies. General hierarchies are comprised of partial order sets of common inheritance relations. And in more restrictive environments, limited hierarchies require the existence of either a single immediate ascendant or descendant role in the hierarchy. Mathematically hierarchy is a partial order defining seniority relations between roles, whereby senior roles acquire the permissions from their juniors and junior roles acquire users from their seniors [3].

Another virtue of RBAC is to constrain authorization with SSD and DSD relationships to prevent the Conflict Of Interest (COI), which is common from business requirements. SSD handles the enforcement of static COI policies. For example, let r_1 and r_2 be two conflicting roles, and user u_1 assigned to role r_1. By enforcing

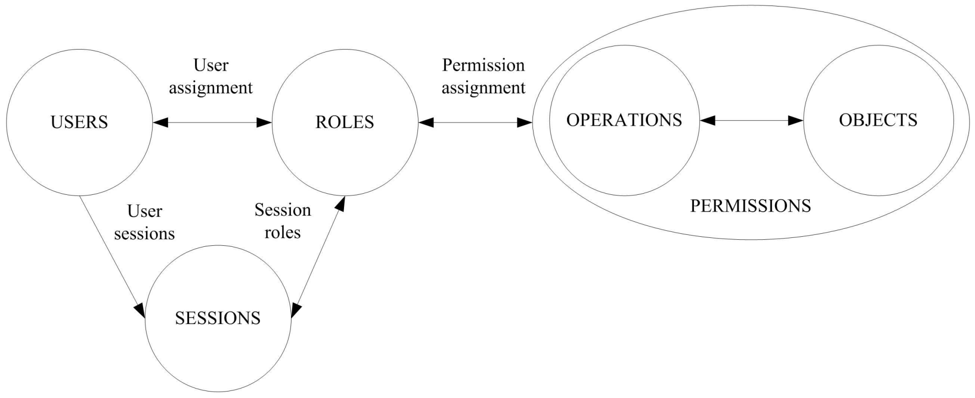

Fig. 1 The core RBAC model.

a SSD constraint between roles r_1 and r_2, RBAC prohibits the assignment of user u_1 to role r_2, since the two roles are COI. The constraints are defined and restricted in the design phase. In the presence of role hierarchies, the SSD constrains are enforced in the same way for all the directly assigned and inherited roles. DSD relationships handles COI policies in the context of a session, where a user is activated with a set of assigned roles when logged into the system. As SSD, DSD constraints were specified in the design phase, however, they are enforced during the run-time of authoring process through activated sessions, thus, preventing the simultaneous activation of two or more conflicting roles.

RBAC administration is divided into user and administrator spaces. The former includes user roles and the latter administrative roles with permissions and operations, respectively. Once again, the principle of least privileged is maintained. Various RBAC administration models were proposed in [11], [12], [40], [46], with different approaches in role based administration.

RBAC is usually operated under a single domain. We adopt the multi-domain enabled RBAC model as domRBAC from [16] for modern collaborative systems. domRBAC is capable of enforcing security policies among application domains to assure a secure collaborative environment without permitting violations caused by new inter-domain role assignments. In addition, domRBAC is able to provide basic resource usage management applicable to cloud environments.

2.2 Secure Inter-operation

Secure inter-operation in collaborative systems is required for secure collaboration among participating parties such that the principles of autonomy and security can be guaranteed [15]. The principle of autonomy states that if an access is permitted by an individual system, it must also be permitted under secure inter-operation. The principle of security states that if an access is denied by an individual system, it must also be denied under secure inter-operation. In a RBAC collaborative system, violations of secure inter-operation can be caused by adding inter-domain role inheritance relations. As stated in [51] these types of violations can be detected by checking for cyclic inheritance, privilege escalation, and violation of SoD relations in RBAC policies. Thus, both security and autonomy can be characterized as safety properties of a system, which should be preserved during collaborations since their enforcement means that *"something bad never happens"* [6].

In the following subsection, we illustrate how to identify the aforementioned properties in a RBAC policy. Henceforth, to differentiate roles, users and permissions among domains, we use the *DomainRole* format in [16] whenever is needed to, where *Domain* denotes a domain name and *Role* denotes a role name. Thus, a role can be expressed as $d_{domain}r_{role}$, and if a role r_k belongs to a domain d_i, we write d_ir_k. The same applies for users and permissions. Further, an arrow $\rightarrow$ in Figures 2 - 4 denotes an immediate inheritance relation between two roles. For example $r_1 \rightarrow r_2$ denotes that role r_1 inherits the permissions of role r_2.

2.2.1 Cyclic Inheritance Property.

In multi-domain RBAC systems, the cyclic inheritance refers to the problem that a user d_iu_t assigned to the role d_ir_k in domain d_i, is authorized for the permissions of another local role d_ir_j such as $d_ir_j \gg d_ir_k$ (see Subsection 3.1 for definition of $\gg$), even though d_iu_t is not directly assigned to d_ir_j in the role hierarchy of domain d_i as shown in Figure 2.

2.2.2 Privilege Escalation Property.

Privilege escalation refers to the problem that a user d_iu_t assigned to a role d_ir_j in domain d_i, is authorized

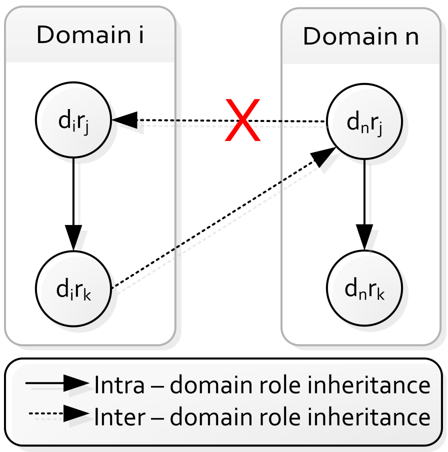
Domain i
Domain n
d_i r_j
d_i r_k
d_n r_j
d_n r_k
Intra – domain role inheritance
Inter – domain role inheritance

Fig. 2 Cyclic inheritance.

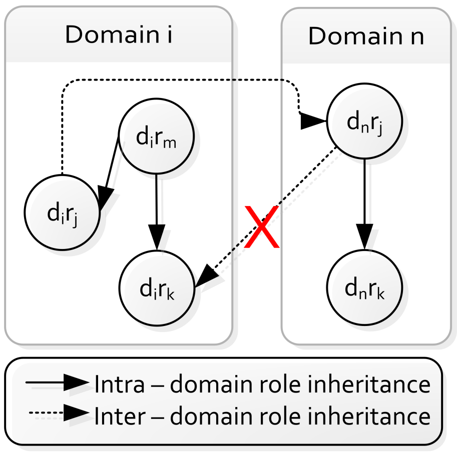
Domain i
Domain n
d_i r_m
d_i r_j
d_i r_k
d_n r_j
d_n r_k
Intra – domain role inheritance
Inter – domain role inheritance

Fig. 3 Privilege escalation.

for the permissions of another local role $d_i r_k$ such as $\neg(d_i r_j \geq d_i r_k)$ (see Subsection 3.1 for definition of $\geq$), even though $d_i u_t$ is not directly assigned to role $d_i r_k$ in the role hierarchy of domain d_i (Figure 3).

2.2.3 Separation of Duty Property.

SoD requires two or more division between users, so that no single user can compromise security. SoD methods can be further categorized into SSD and DSD. SSD are constraints that are placed on roles at the time roles are assigned to users. When implementing SSD in role hierarchy, both inherited and directly assigned roles need to be considered. In the same manner, DSD needs to check the role hierarchy when users activate already assigned roles [12].

Verification of the SSD property is based on the following properties [12]:

Property 1. Roles r_k and r_m are mutually exclusive if neither one inherit the other directly or indirectly.

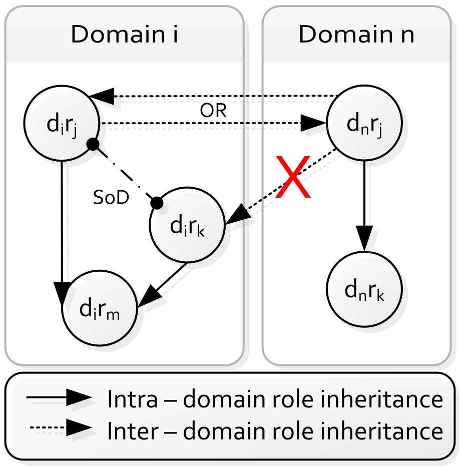
Domain i
Domain n
d_i r_j
OR
d_n r_j
SoD
d_i r_k
d_i r_m
d_n r_k
Intra – domain role inheritance
Inter – domain role inheritance

Fig. 4 Separation of duty.

Property 2. If roles r_k and r_m are mutually exclusive then there is no other role inherits both of them.

Similar to SSD, DSD has the property [12]:

Property 3. If SSD holds, then DSD is maintained.

Thus, properties 1 and 2 must be guaranteed.

2.2.4 Autonomy Property.

In addition to the security principle, autonomy should also be preserved for secure inter-operation. Maintaining the autonomy of all collaborative domains is a key requirement of the policy for inter-operation. However, access of inter-operation may be significantly reduced or even not authorized at all if the autonomy of individual domains is over addressed. Therefore, balancing autonomy and interoperability might be considered [51]. In almost any collaborative environment, it is not permissible to violate any domain's security policy. However, some domains may be willing to compromise their autonomy for the sake of establishing more interoperability, provided that autonomy loss remains within acceptable limits. Specifically, when using a RBAC policy integration framework, a violation in the autonomy of a domain may occur because of induced SoD constraints, as described in [51]. An induced SoD constraint is a SoD constraint between two intra-domain roles (e.g. $d_1 r_1$ and $d_1 r_2$) which do not conflict with each other in their original domain's RBAC policy. In a multi-domain system such a SoD constraint will deny concurrent access on roles $d_1 r_1$ and $d_1 r_2$, thus, reducing the autonomy in the original domain. Nevertheless, the autonomy principle, when applicable, can be verified by checking if all the assigned and authorized permissions of a role $d_i r_k$ in a domain d_i are preserved for inter-operation.

2.3 The Applied Model Checking Technique

For the verification of secure inter-operation properties, we apply the technique proposed in [19], which focuses on the verification of generic properties for AC models. The technique is able to cope with various types of AC properties including static, dynamic and historical. It also supports the generation of test cases to check the conformance between models and policy rules through combinatorial test array [37], and optionally generate the verified AC policies in eXtensible Access Control Markup Language (XACML) version 2.0 or 3.0, which is becoming the de facto language for the specification of policy rules in modern collaborative systems such as the cloud. We adopt the finite state machine to describe the transitions of the authorization states, and the usage of static constraints so to adequately cover the verification of secure inter-operation properties in RBAC. The technique is to verify specified AC properties against AC models using a black-box model checking method [20]. An implementation – Access Control Policy Tool (ACPT) [22] is developed by NIST Computer Security Division in corporation of North Carolina State University.

ACPT provides graphical user interface (GUI) templates for composing AC policies and properties. Checking for conformance of AC properties and models is through the SMV (Symbolic Model Verification) model checker. In addition, ACPT provides a complete test suite generated by NIST's combinatorial testing tool ACTS [37] and an XACML policy output for the verified model. Through these four major functions, ACPT performs syntactic and semantic verifications as well as the interfacing for composing and combining AC policies. ACPT assures the efficiency of specified AC policies, and detecting policy faults that leak or prohibit legitimate access privileges. Currently, ACPT provides model templates for three major AC models: static Attribute-Based AC, Multi-Leveled Security, and stated WorkFlow, and partially implements the methods described in [20]. Despite providing all the adequate functionality for the verification of AC policies, the function of RBAC reasoning regarding role hierarchies is absent. Nevertheless, we applied this model checking technique for its capabilities of defining and verifying basic RBAC rule statements and property propositions.

3 Model Checking Cloud Policies

We describe formal definitions for the core and hierarchical RBAC. Our formal definitions modified the transition system defined in [20], which is mainly for generic AC models. We also provide the definition of secure inter-operation properties in temporal logic that needs to be verified to ensure a secure policy for a consistent and conflict-free inter-operation. Specifically, we define the security properties of cyclic inheritance, privilege escalation, SoD, and autonomy (see Subsection 2.2).

3.1 Model Definitions

Each domain specifies its own policy in most collaborative systems today. Hence, we separate the specification of single domain AC policies (i.e. intra-domain administration) from multiple domains collaborative policies (i.e. inter-domain administration). Both specifications follow the ANSI INCITS 359-2004 definition of RBAC [3]. We also define review functions for intra-domain and inter-domain administration. The main components [3] are defined below.

- USERS, ROLES, OPS, OBS, stands for users, roles, operations, and objects, respectively.
- UA $\subseteq$ USERS $\times$ ROLES, a many-to-many set of user-to-role assignment relation mapping.
- PRMS = $2^{(OPS \times OBS)}$, the set of permissions.
- PA $\subseteq$ PRMS $\times$ ROLES, a many-to-many set of permission-to-role assignment relation mapping.
- Op(p: PRMS) $\rightarrow \{op \subseteq OPS\}$, the permission to operation mapping, which gives the set of operations associated with permission p.
- Ob(p: PRMS) $\rightarrow \{ob \subseteq OBS\}$, the permission to object mapping, which gives the set of objects associated with permission p.

For intra-domain, we redefine the hierarchical relations and administrative review functions below.

- assigned users: $SU_{d_i}(d_i r_k : ROLES) \rightarrow 2^{USERS}$, the mapping of role $d_i r_k$ onto a set of users enrolled in domain d_i. Formal definition: $SU_{d_i}(d_i r_k) = \{d_i u_t \in USERS | (d_i u_t, d_i r_k) \in UA\}$.
- assigned permissions: $SP_{d_i}(d_i r_k : ROLES) \rightarrow 2^{PRMS}$, the mapping of role $d_i r_k$ onto a set of permissions defined in domain d_i. Formal definition: $SP_{d_i}(d_i r_k) = \{d_i p_w \in PRMS | (d_i p_w, d_i r_k) \in PA\}$.
- $RH_{d_i} \subseteq ROLES \times ROLES$ is a partial order set on $ROLES$ of inheritance relation in domain d_i, denoted as $\geq$, where $d_i r_k \geq d_i r_m$ only if all permissions of $d_i r_m$ are also permissions of $d_i r_k$, and all users of $d_i r_k$ are also users of $d_i r_m$. Formal definition: $d_i r_k \geq d_i r_m \Rightarrow UP_{d_i}(d_i r_m) \subseteq UP_{d_i}(d_i r_k) \wedge UU_{d_i}(d_i r_k) \subseteq UU_{d_i}(d_i r_m)$.
- authorized users: $UU_{d_i}(d_i r_k : ROLES) \rightarrow 2^{USERS}$, the mapping of role $d_i r_k$ onto a set of users enrolled in domain d_i in the presence of a role hierarchy defined in domain d_i. Formal definition: $UU_{d_i}(d_i r_k) = \{d_i u_t \in USERS | d_i r_m \geq d_i r_k, (d_i u_t, d_i r_m) \in UA\}$.

- authorized permissions: $UP_{d_i}(d_ir_k : ROLES) \rightarrow 2^{PRMS}$, the mapping of role d_ir_k onto a set of permissions defined in domain d_i in the presence of a role hierarchy define in domain d_i. Formal definition: $UP_{d_i}(d_ir_k) = \{d_ip_w \in PRMS | d_ir_k \geq d_ir_m, (d_ip_w, d_ir_m) \in PA\}$.

For inter-domain, we extend the aforementioned hierarchy relations and administrative review functions below:

- $RH \subseteq ROLES \times ROLES$ is a partial order set on $ROLES$ of inheritance relation, denoted as $\geq$, where $d_ir_k \geq d_jr_m$ only if all permissions of d_jr_m are also permissions of d_ir_k, and all users of d_ir_k are also users of d_jr_m. Formal definition: $d_ir_k \geq d_jr_m \Rightarrow UP(d_jr_m) \subseteq UP(d_ir_k) \wedge UU(d_ir_k) \subseteq UU(d_jr_m)$.
- authorized users: $UU(d_ir_k : ROLES) \rightarrow 2^{USERS}$, the mapping of role d_ir_k onto a set of users enrolled in any domain in the presence of a inter-domain role hierarchy. Formal definition: $UU(d_ir_k) = UU_{d_i}(d_ir_k) \cup \{d_ju_t \in USERS | d_jr_m \geq d_ir_k, (d_ju_t, d_jr_m) \in UA\}$.
- authorized permissions: $UP(d_ir_k : ROLES) \rightarrow 2^{PRMS}$, the mapping of role d_ir_k onto a set of permissions defined in any domain in the presence of a inter-domain role hierarchy. Formal definition: $UP(d_ir_k) = UP_{d_i}(d_ir_k) \cup \{d_jp_w \in PRMS | d_ir_k \geq d_jr_m, (d_jp_w, d_jr_m) \in PA\}$.

The absence of relational operators in temporal logic, i.e. $\gg$ and $\geq$, led us to the definition of a series of appropriate predicates below.

- $IR(r_k, r_m)$ denotes the existence of an immediate (either inter or intra domain) inheritance relationship between the two roles. Formal definition: $IR(r_k, r_m) = true \Leftrightarrow r_k \gg r_m$. The operator $\gg$ means immediate inheritance relation as defined in [3].
- $MR_{d_i}(d_ir_k, d_ir_m)$ denotes that there is an (immediate or not) inheritance relationship between the two roles in the role hierarchy defined in domain d_i. Formal definition: $MR_{d_i}(d_ir_k, d_ir_m) = true \Leftrightarrow d_ir_k \geq d_ir_m$.
- $RP(r_k, r_m)$ denotes that for two roles with an immediate inheritance relation $(r_k, r_m : r_k \gg r_m)$ the set of role's r_k assigned permissions is a subset of role's r_m authorized permissions. Formal definition: $RP(r_k, r_m) = true \Leftrightarrow IR(r_k, r_m) \wedge SP_{d_i}(r_k) \subseteq UP(r_m)$.
- $IB_{d_i}(d_ir_k, d_ir_m, r_n)$ denotes that for two roles d_ir_k and d_ir_m in domain d_i the set of role's r_n authorized permissions, regardless of the domain to which it belongs, includes the assigned permissions of both roles d_ir_k and d_ir_m, where r_n is a role senior to roles d_ir_k and d_ir_m. Formal definition: $IB_{d_i}(d_ir_k, d_ir_m, r_n) = true \Leftrightarrow SP_{d_i}(d_ir_k) \cup SP_{d_i}(d_ir_m) \subseteq UP(r_n) \wedge r_n \geq d_ir_k \wedge r_n \geq d_ir_m$.
- $BA(d_ir_k)$ denotes that the mapping of role d_ir_k onto the set of all its assigned and authorized permissions in domain d_i is a subset of all its permissions under the presence of an inter-domain hierarchy. Formal definition: $BA(d_ir_k) = true \Leftrightarrow UP_{d_i}(d_ir_k) \subseteq UP(d_ir_k)$.

3.2 Transition System

In this subsection, we define AC rule, property, and transition system for RBAC models. The definitions are modified from [20]. We use Computation Tree Logic (CTL) for the specification of policy properties.

In CTL, prefixed path quantifiers assert arbitrary combinations of linear-time operators. For our purpose, we use universal path quantifier $\forall$ means *"for all paths"* and the linear temporal operators $\Box$ and $\Diamond$ means *"always"* and *"eventually"*, respectively. Furthermore, we use the temporal modalities $\forall\Box\Phi$ representing *invariantly Φ*, and $\forall\Diamond\Phi$ representing inevitably Φ, where Φ is a state formula.

Definition 1. An RBAC rule is a proposition of type "if c then d", where constraint c is a predicate expression on $(r, UP(r))$ for the permission decision d. Thus, RBAC policies consist a sequence of rules, each has form $(r, UP(r))$ in the logic expression c.

Definition 2. An RBAC AC property p is a formula of type "$b \rightarrow d$", where the result of the access permission d depends on quantified predicate b on $(r, UP(r))$ mapping. The $\rightarrow$ means "imply".

Definition 3. A transition system TS is a tuple (S, Act, δ, i_0) where

- S is a set of states, $S = \{Permit, Deny\}$,
- Act is a set of actions, where $Act = \{(r_1, UP(r_1)), \ldots, (r_n, UP(r_n))\}$,
- δ is a transition relation where $\delta : S \times Act \rightarrow S$, and
- $i_0 \in S$ is the initial state.

The p in Definition 2 is expressed by the proposition $p : S \times Act^2 \rightarrow S$ of TS, which can be collectively translated in terms of logical formula such that $p = (s_i * (r_1, UP(r_1)) * \ldots * (r_n, UP(r_n))) \rightarrow d$, where $p \in P$ is a set of properties, and $*$ is a Boolean operator in CTL [6].

The RBAC rule function as the transition relation δ in the TS. Thus, by representing RBAC AC property in temporal logic formula p, we can assert that model TS satisfies p by $TS \vDash \forall\Box(b \rightarrow \forall\Diamond d)$. Property $\forall\Box(b \rightarrow \forall\Diamond d)$ is a response pattern such that d responds to b globally (b is the cause and d is the effect) [49].

3.3 Specification of Properties

In a collaborative RBAC system, violations of secure inter-operation may be caused by add-hoc inter-domain role inheritance. As stated in [51] and [16], such violations can be checked by detecting cyclic inheritance, privilege escalation, and violation of SoD relations in the system. We provide formal representations of the aforementioned safety properties using temporal logic below.

3.3.1 Cyclic Inheritance Property.

To detect a cyclic inheritance for a role d_ir_k, we check if the proposition $RP(d_ir_j, d_ir_k) \rightarrow \forall\Diamond Deny$ is satisfied invariantly in the TS, formally:

$$TS_{RBAC} \vDash \forall\Box(RP(d_ir_j, d_ir_k) \rightarrow \forall\Diamond Deny). \tag{1}$$

3.3.2 Privilege Escalation Property.

To detect privilege escalation for a role d_ir_k against a role d_ir_j, we check if the proposition $(\neg MR_{d_i}(d_ir_j, d_ir_k) \wedge RP(d_ir_j, d_ir_k)) \rightarrow \forall\Diamond Deny$ is satisfied invariantly by the TS_{RBAC}, formally:

$$\begin{aligned} TS_{RBAC} \vDash \forall\Box((\neg MR_{d_i}(d_ir_j, d_ir_k)\wedge \\ RP(d_ir_j, d_ir_k)) \rightarrow \forall\Diamond Deny). \end{aligned} \tag{2}$$

3.3.3 Separation of Duty Property.

In general, we enforce SoD by role pairs [12]. The minimum number of mutual exclusion role pairs needs to be checked for the $(d_irs, n) \in SSD$ constraint, where each d_irs is a role set and n is a number $\geq$2 such that no user is assigned to or authorized for n or more roles from the set d_irs in each $(d_irs, n) \in SSD$. This is equal to the binomial coefficient $_{|d_irs|}C_2 \equiv \begin{pmatrix} |d_irs| \\ 2 \end{pmatrix} \equiv \frac{|d_irs|!}{2!(|d_irs|-2)!}$.

SSD property verification rely on mutual exclusion of roles specified by role pairs [12]. This implies that roles can not have common assigned users for any role pair in the role set d_irs, formally:

$$\begin{aligned} TS_{RBAC} \vDash \forall\Box((d_ir_j \in d_irs_w \wedge d_ir_k \in d_irs_w \wedge \\ (RP(d_ir_j, d_ir_k) \vee RP(d_ir_k, d_ir_j)\vee \\ IB_{d_i}(d_ir_j, d_ir_k, r_m))) \rightarrow \forall\Diamond Deny). \end{aligned} \tag{3}$$

3.3.4 Autonomy Property.

The autonomy principle, when applicable, can be verified by checking if all the assigned and authorized permissions of a role d_ir_k in a domain d_i are preserved during inter-operation, formally:

$$TS_{RBAC} \vDash \forall\Box(BA(d_ir_k) \rightarrow \forall\Diamond Permit). \tag{4}$$

4 Implementation Aspects

This section discusses aspects of the implemented technique. The technique described in [19] is unable to specify role hierarchies for RBAC policies because it is not geared for RBAC models. To specify role hierarchies, we propose a role-to-role mapping algorithm derived from the graph theory in terms of Definition 1. When defining a role hierarchy $r_k \geq r_m$, r_k and r_m are assigned to permissions $PRMS_k$ and $PRMS_m$, respectively. In turn, we generate additional rules according to Definition 1, apart from the initial rules that map roles and their assigned permissions. For example, regarding the previously mentioned roles r_k and r_m, an additional rule is generated automatically to record the r_k's inheritance of permission $PRMS_m$. In this way, a reasoning for RBAC hierarchies is introduced, which depicts the role hierarchy $r_k \geq r_m$ (i.e. role r_k is a senior of role r_m). As role hierarchies are represented by sparse graphs, we use linked lists instead of matrices for implementing role hierarchies since, in the examined case, the former are more efficient regarding performance. Formal implementation aspects are listed below [16].

1. $G = (V, E)$ is a directed graph of inter-domain role hierarchy. The graph consists of a finite, non-empty set of role vertices $V \subseteq ROLES$ and a set of edges E. Each edge is an ordered pair (d_ir_m, d_jr_n), of role vertices that indicates the relation: $d_ir_m \geq d_jr_n$.
2. A path in a G graph is a sequence of edges (d_ir_1, d_ir_2), (d_ir_2, d_ir_3), ..., (d_ir_{n-1}, d_ir_n) with length n-1 connecting from role vertex d_ir_1 to role vertex d_ir_n. A path represents indirect inheritance relations between role vertex d_ir_1 and d_ir_n.
3. An adjacency list representation for graph $G = (V, E)$ is an array L of $|V|$ lists, one for each role vertex in V. For each role vertex d_ir_m there is a pointer $L_{d_ir_m}$ to a linked list containing all the role vertices that are adjacent to d_ir_m. A linked list is terminated by a *nil* pointer. We refer to the adjacency list of a graph G as A_G.
4. $G^* = (V, E^*)$ is the transitive closure of the graph $G = (V, E)$ where E^* contains an edge (u, v) if and

Algorithm Rule and property creation algorithm

```
1: procedure ITERATOR_SKELETON ($T_G$)
2:     for all vertex $dr_i \in T_G$
3:         for all adjacent vertex $dr_j$
4:             //Generate the required rule or property
5: end procedure
```

Fig. 5 Iterator skeleton procedure.

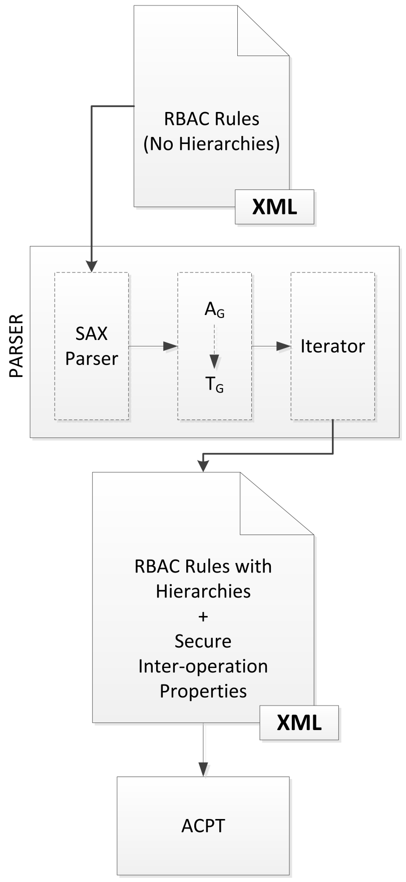

Fig. 6 The proposed tool chain.

only if G contains a path from u to v. The applied transitive closure computation algorithm is based on [39] and [45] (i.e. strong components) with a time complexity of $O(|V||E|)$. We refer T_G as the transitive closure list of a directed graph $G = (V, E)$ with adjacency list A_G.

To specify role hierarchies according to Definition 1, we compute the transitive closure list T_G from the adjacency list A_G of graph $G = (V, E)$ built from RBAC rules. The result is generated from iterating the vertices of the transitive closure list T_G. And using iteration for the vertices to generate specifications to be verified as defined in Definition 2. The pseudo code in Figure 5 procedure illustrates how the iteration is generated.

Specifically, SAX stream API is employed as the parser to load a RBAC policy in XML (created by ACPT [22]) and produce the adjacency list A_G. In turn, the T_G is computed from the A_G. Next, a new XML file is created by iterating the vertices of T_G using the iterator in Figure 5. The XML includes both the RBAC rules with hierarchies, and specifications to be verified. Figure 6 illustrates the described tool chain for the process.

We further elaborate on the creation and computation of the A_G and T_G, respectively, through an example. Let's assume the multi-domain AC policy depicted in Figure 7 that allows collaboration between domain d_1 and domain d_2. Domain d_1 has the following roles: d_1r_a, d_1r_b, d_1r_c, d_1r_d and d_1r_e. Role d_1r_a inherits all permissions of d_1r_b which further inherits d_1r_e. Role d_1r_c inherits all permissions of d_1r_d which further inherits d_1r_e. A SSD relation is specified for d_1r_b and d_1r_c meaning that these roles cannot be assigned to the same user simultaneously. Domain d_2 has the following roles: d_2r_f and d_2r_g. Role d_2r_f inherits all permissions of d_2r_g. The multi-domain AC policy defines the following inter-domain inheritance relationships between domains d_1 and d_2:

(a) Role d_1r_b inherits role d_2r_g.
(b) Role d_2r_g inherits role d_1r_c.

The A_G and T_G list representations are created, as listed below, and can used as an input to our specified tool chain for further manipulation from the iterator procedure. In our implementation, we utilise functionalities provided by the BOOST C++ libraries [7]. Specifically, we use the *adjacency_list* class, which implements a generalized adjacency list graph structure for A_G and the *transitive_closure()* function, which transforms the input graph G into the transitive closure graph. In the following examples, A_G and T_G represents the adjacency and transitive closure lists of the multi-domain AC policy depicted in Figure 7. $A_{G_{d_1}}$ and $T_{G_{d_1}}$ represents the adjacency and transitive closure lists of the AC policy in domain 1. The latter are utilised as a requirement for verifying the principle of autonomy between the original domain and collaborative environment.

$$A_G = \begin{cases} Vertex & Linked\,list \\ d_1r_a : & d_1r_b \rightarrow nil \\ d_1r_b : & d_1r_e \rightarrow d_2r_g \rightarrow nil \\ d_1r_c : & d_1r_d \rightarrow nil \\ d_1r_d : & d_1r_e \rightarrow nil \\ d_1r_e : & nil \\ d_2r_f : & d_2r_g \rightarrow nil \\ d_2r_g : & d_1r_c \rightarrow nil \end{cases}$$

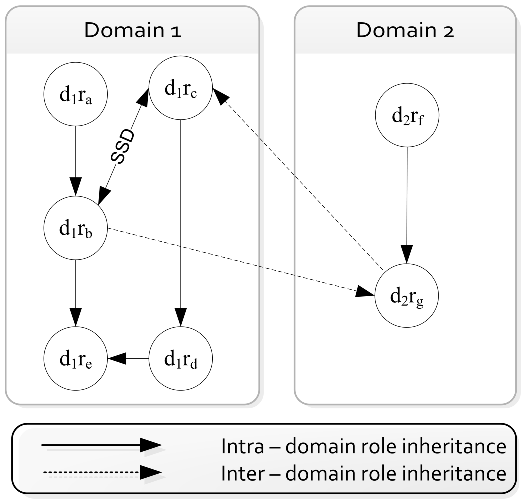

Fig. 7 A multi-domain AC policy defining inter-operation between domains d_1 and d_2.

$$A_{G_{d_1}} = \begin{cases} Vertex & Linked\,list \\ d_1r_a: & d_1r_b \to nil \\ d_1r_b: & d_1r_e \to nil \\ d_1r_c: & d_1r_d \to nil \\ d_1r_d: & d_1r_e \to nil \\ d_1r_e: & nil \end{cases}$$

$$T_G = \begin{cases} Vertex & Linked\,list \\ d_1r_a: & d_1r_b \to d_1r_c \to d_1r_d \to d_1r_e \to d_2r_g \to nil \\ d_1r_b: & d_1r_c \to d_1r_d \to d_1r_e \to d_2r_g \to nil \\ d_1r_c: & d_1r_d \to d_1r_e \to nil \\ d_1r_d: & d_1r_e \to nil \\ d_1r_e: & nil \\ d_2r_f: & d_2r_g \to d_1r_c \to d_1r_d \to d_1r_e \to nil \\ d_2r_g: & d_1r_c \to d_1r_d \to d_1r_e \to nil \end{cases}$$

$$T_{G_{d_1}} = \begin{cases} Vertex & Linked\,list \\ d_1r_a: & d_1r_b \to d_1r_e \to nil \\ d_1r_b: & d_1r_e \to nil \\ d_1r_c: & d_1r_d \to d_1r_e \to nil \\ d_1r_d: & d_1r_e \to nil \\ d_1r_e: & nil \end{cases}$$

5 Performance Evaluation

In this section, we present a series of performance metrics regarding the evaluation of the proposed technique in small, medium and large-scale cloud systems. Furthermore, we recall performance metrics regarding the enforcement of secure inter-operation in domRBAC [16] to compare the performance overhead between the proposed technique and embedded secure inter-operation approaches. Specifically, Subsection 5.1 provides information regarding the simulation of AC policies and requests and system configuration. Subsection 5.2 presents the evaluation of the proposed technique as a management service/tool. Subsection 5.3 demonstrates performance metrics regarding domRBAC. Subsection 5.4 presents a discussion on the results.

5.1 Specifications

To evaluate performance, we first generated AC requests from cloud users using the NetworkX python package [35], which operated as input data for the proposed parser. Cloud users' AC requests and corresponding access privileges according to AC polices were created by the *gnc_graph* function, which returns a growing network with copying (GNC) directed graph built by adding nodes linked to previous ones (one at a time) [26], [35].

Next step, we simulated role assignments for a number of cloud hosted domains using the domRBAC simulator [16]. We assumed the existence of 5, 10, 15 and 20 cloud hosted domains containing 50 roles each. In turn, we simulated collaborations among cloud hosted domains, which resulted in an aggregation of 250, 500, 750 and 1000 roles per collaborative environment, respectively. Therefore, we have managed to perform simulation of various size systems (i.e. small to large-scale systems) [50].

Our evaluation of the simulated cloud system is running on the Microsoft Windows 2003 Server Enterprise Edition operating system with service pack 2 running on 3.0 GHz Intel Pentium processor, with 2 GB of RAM.

5.2 Secure Inter-operation via Verification

In this section, we provide a number of quantitative results from the technique used for the verification of the proposed secure inter-operation properties. Table 1 summarizes the information generated from both the implemented parser and NuSMV. Specifically, RBAC policies are indicated by the number of edges of T_G. The verification time for specifications is significantly bigger for large scale systems, as the time was increased by both the number of reachable states from binary decision diagram's (BDD) and specifications. The number of BDD's reachable states increase when the number of RBAC policies increases. However, the specifications can be parallel verified. Plus, the examined simulated data were evaluated by the NuSMV symbolic model checker in both normal and optimized mode. Normal mode does not include any additional command line parameters, while optimised mode includes three parameters to improve the performance [38].

Table 2 summarizes the aforementioned performance measurements for test cases number 2, 3 and 4 in both normal and optimized mode. Test case number 1 is omitted because its single process execution time takes less than a minute. Figures 8, 9 and 10 illustrates the time required for each of the nine processes in both normal and optimized mode, for test cases 2, 3 and 4, respectively. To calculate the speed improvement for property verification from running nine processes in parallel versus one single process, we used the following formula:

$$Reduction\_time(\%) = (1 - \frac{maxT}{Single\_process\_time}) * 100 \quad (5)$$

where $maxT$ is the maximum time value of a set of elements $T = (t_{p_i})_{i=1}^{N}$, where N is the number of parallel processes and t_{p_i} is the execution time of a process p_i. Table 2 shows the time required for one process to verify all the specifications, where $minT$ is time value of a set of elements $T = (t_{p_i})_{i=1}^{N}$, and $\sum_{i=1}^{N} t_{p_i}$ is the total time required for all parallel processes to finish when executed sequentially.

The results conclude that parallel processing significantly improves property verification performance as in the example cases, which evenly distribute specifications to nine processes and reduced time by 86% in normal and 77% in optimized mode on the average when compared to the time from single process. Further, from Figures 8, 9 and 10, we conclude that time for NuSMV in normal mode fluctuated greater than in optimized mode. Thus, performance in optimized mode is more steady and predictable than in normal mode. The results also show that the following applies to small and medium systems: $maxT_{normal} < maxT_{optimized}$ and $minT_{normal} < minT_{optimized}$. However, this will significantly change in large scale systems where $maxT_{normal} \gg maxT_{optimized}$ and $minT_{normal} \gg minT_{optimized}$. Lastly, in all cases, we observed that $\sum_{i=1}^{N} t_{p_i} \neq Single\_process\_time$, which means that the sum of all times required for verifying a number of specifications using sequentially processes differs from the time required to verify the same number of specifications using a single process.

In summary, the parallel property verification seems to significantly improve the efficiency for the proposed technique in term of the time required for a complete property verification for large scale systems. Therefore, the proposed technique can be used to verify the correctness of AC systems as a management service/tool. And, despite the limitations of model checking (i.e. state-space explosion issue), it is an effective technique to expose potential design and implementation errors [6].

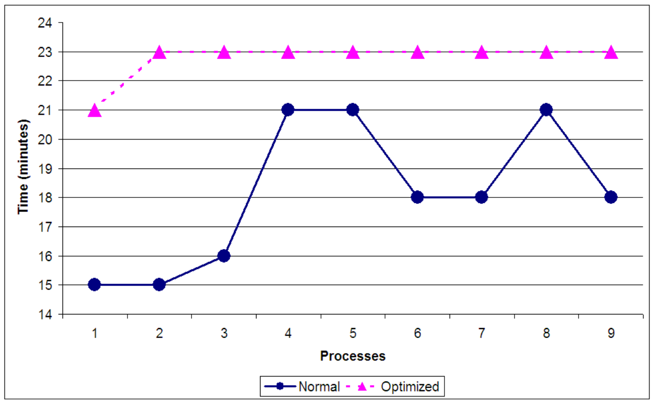

Fig. 8 Parallel verification of specifications for test case #2 (approximately 3029 specifications per process).

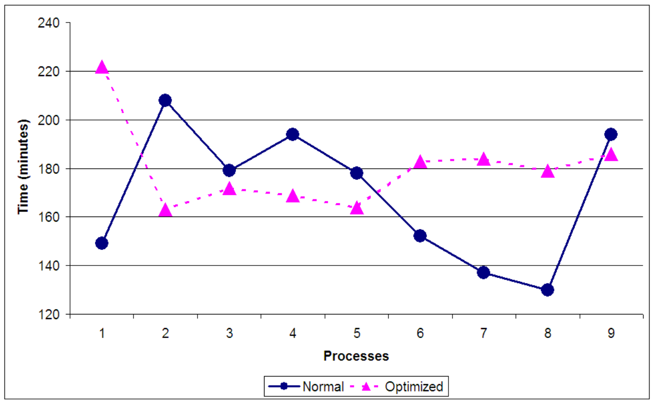

Fig. 9 Parallel verification of specifications for test case #3 (approximately 5793 specifications per process).

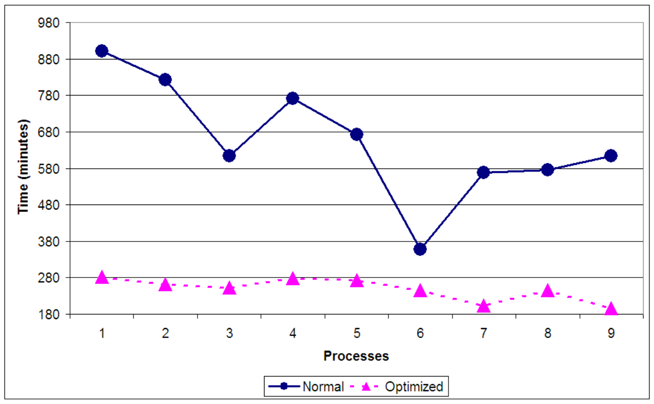

Fig. 10 Parallel verification of specifications for test case #4 (approximately 8551 specifications per process).

Table 1 Summary of the evaluated data.

#	Roles	T_G Edges	# of Specifications	Reachable States	Execution Time (min.)
1.	5*50	1031	12405	$2^{29.2366}$	< 1
2.	10*50	1784	27267	$2^{32.2279}$	176
3.	15*50	3179	52143	$2^{33.9799}$	1472
4.	20*50	3421	76964	$2^{35.2236}$	5820

Table 2 Summary of the performance measurements using 9 processes (Normal versus Optimized mode).

#	Single process time (min.)	$\sum_{i=1}^{N} t_{p_i}$ (min.)	maxT (min.)	minT (min.)	Reduction time
1 N	<1	N/A	N/A	N/A	N/A
1 O	<1	N/A	N/A	N/A	N/A
2 N	176	163	21	15	88%
2 O	134	205	23	21	82%
3 N	1472	1521	208	130	86%
3 O	744	1622	222	163	70%
4 N	5820	5865	901	357	84%
4 O	1506	2241	283	196	81%

N: Normal mode
O: Optimized mode

Table 3 Summary of the evaluated data in the domRBAC simulator.

#	Roles	T_G Edges	Maximum Execution Time (msec)
1.	5*50	1031	< 1
2.	10*50	1784	< 1
3.	15*50	3179	≤ 1
4.	20*50	3421	≤ 1

5.3 Secure Inter-operation in domRBAC

Using the same example data sets for the domRBAC simulator, we performed a series of tests as summarized in Table 3. The maximum execution time is the sum of six metrics, which are required for creating/updating the A_G and T_G and detecting cycle inheritance, privilege escalation or SSD/DSD violations. As shown in Table 3, domRBAC take less than one msec for test cases 1 and 2, and at maximum one msec for test cases 3 and 4.

5.4 Discussion

From Tables 1, 2 and 3, it is shown that secure inter-operation aware AC models (e.g. domRBAC) are able to perform faster than verification techniques. However, an AC model cannot assure or verify its correctness. The proposed verification technique fulfils the latter requirement using formal methods, thus, it is able to assure the correctness of both the AC model and security policies. Moreover, it can be used as a valuable administrative service/tool for both the a priori and a posteriori enforcement of RBAC policies, thus, ensuing a highly secured collaborative environment.

In addition, the proposed technique can benefit of automated testing with ACTS. The latter is integrated in ACPT as presented in Subsection 2.3. ACTS provides pairwise testing, which has become a popular approach to software quality assurance since it often provides effective error detection at low cost. Automating test generation via ACTS can provide much more thorough testing than is possible with most conventional methods. Moreover, testing has a sound empirical basis in the observation that software failures have been shown to be caused by the interaction of relatively few variables and stronger assurance for critical software can be provided by testing all variable interactions to an appropriate strength [27]. As a proof of concept, we provide a list of indicative results in Table 4. Specifically, we illustrate for each of our examined case the number of test cases generated by ACTS and their execution time.

6 Related Work

The need for applying RBAC solutions in cloud systems has been noticed by various researchers. In [30] a refined RBAC model for cloud computing is discussed in order to provide a protection mechanism based on the RBAC model. A number of cloud entities were identified at a refined level to examine the instantiation of RBAC in cloud systems. In [1] RBAC was applied to the implementation of an authorization system for cloud services. The authorization system is able to support multi-tenancy, role-base access control, hierarchical RBAC, path-based object hierarchies and federation. The CARBAC is proposed in [54] as a new access control model for cloud computing, and is mostly fo-

Table 4 Automated Combinatorial with ACTS.

#	Number of Test cases	Execution Time (sec)
1.	62500	0.297
2.	250000	1.11
3.	562500	1.64
4.	1000000	2.843

Test Generation Profile:

Degree of interaction: 2
Mode: scratch
Algorithm: ipog
Progress Info: off
Debug mode: off
Verify Coverage: off

System Under Test:

Name: Fireeye Input
Number of Params: 3

cused on the SaaS delivery model. CARBAC incorporates support for environmental state and object state in access control policies to overcome the subject-centric nature of RBAC, and differentiates role hierarchies to user and data owner hierarchies. Furthermore, RBAC's significance is also indicated by the fact that it is employed by most of the cloud computing platforms, as stated in Section 1. Nevertheless, none of the aforementioned approaches can handle efficiently the integration of AC policies from different cloud hosted domains to define a global access control policy where secure inter-operation among participants is maintained.

A few papers examined the automated verification of generic policies, and a number of techniques have been proposed for the purpose in [17], [23], [19], [13], [18], [21], [25]. The objective for the research is to answer the need for expressive power or better performance of AC polices. Some use verification tools as back end. For instance, the declarative language Alloy [2] supports first order logic and relational calculus, [38], the symbolic model checker NuSMV verifies temporal logic properties with finite state models as well as the SPIN model checker [52]. There are cases where AC policies are defined as ordering relations, which can be further translated into Boolean satisfiability problems, and applied to SAT solvers [21]. The SAT solver is a program that takes formulas in conjunctive normal form (CNF) and returns assignments, or says none exists. Also as part of the AVANTSSAR EU research project, an automated symbolic analysis of ARBAC-Policies has been proposed in [4]. The latter can enable the design of parametric security analysis techniques since it is based on model checking techniques that are targeted to infinite state systems, through the use of first-order logic and theorem proving techniques. Nevertheless, despite of its virtues, ARBAC policies are examined on a domain basis, and therefore collaboration among different domains and secure inter-operation are not examined. The aforementioned techniques can serve as foundations for the verification of system specifications. A specification of a system can be defined as *"what the system is supposed to do"* [28], so, we decided to apply the technique in [18] for the reasons stated in Section 2.

To the best of our knowledge, there is no similar work in the area of policy verification for multi-domain cloud systems, let alone a tool to handle inter-operation security among domains in cloud systems. A project with relevant research work can be found in [24]; the main objective of that research is to facilitate the management of distributed resources in Virtual Organizations of Grid systems. Their focus is on the detection of policy conflicts (i.e. different policy behaviours) and redundancies (i.e. same policy behaviours), which has little concern for secure inter-operation. Moreover, the performance of their approach was limited for analysing small-scale systems (e.g. 557 policy rules) and the verification of a relative small number of specifications (e.g. 125 specifications).

7 Conclusions

In this paper, we demonstrated the security policy verification in multi-domain cloud systems. We applied a partially redefined ANSI INCITS 359-2004 to express both intra-domain and inter-domain administrations. We defined cyclic inheritance, privilege escalation, and SoD constraints properties in temporal logic. The verification of these security properties is vital, because they assure the correctness of the enforced policy for secure inter-operation of cloud systems. We further verified the properties in a RBAC implementation. We modified the model checking technique in [20] in the context of RBAC models. We proposed a parser to tackle RBAC reasoning and handle role hierarchies. To the best of our knowledge, there is no equivalent secure inter-operation verification technique for RBAC that is applied to cloud systems. The efficiency of our technique was evaluated via a number of simulations. The results showed that our approach is feasible for large scale systems by independent and parallel processing. We conclude that our proposed technique as a management service/tool for system administration allows verifying either the a priori or a posteriori enforcement of RBAC policies correctness. Also the integration of automating test generation via ACTS strengthens our technique with the capability to perform software assurance. To sum, the aforementioned renders the proposed technique an efficient approach to maintain secure inter-operation in multi-domain cloud systems.

Acknowledgements This work has been (partially) funded by the Research Committee of the University of Macedonia, Greece.

References

1. J. Alcaraz Calero, N. Edwards, J. Kirschnick, L. Wilcock, and M. Wray. Toward a multi-tenancy authorization system for cloud services. *Security Privacy, IEEE*, 8(6):48–55, 2010.
2. Alloy. A language and tool for relational models, http://alloy.mit.edu/alloy/.
3. ANSI. ANSI INCITS 359-2004, role based access control, 2004.
4. A. Armando and S. Ranise. Automated symbolic analysis of arbac-policies (extended version). *arXiv preprint arXiv:1012.5590*, 2010.
5. J. Bacon, D. Evans, D. M. Eyers, M. Migliavacca, P. Pietzuch, and B. Shand. Enforcing end-to-end application security in the cloud (big ideas paper). In *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware*, pages 293–312. Springer-Verlag, 2010.
6. C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.
7. Boost. Boost c++ libraries, http://www.boost.org/, 2011.
8. J. W. Bryans and J. S. Fitzgerald. *Formal engineering of XACML access control policies in VDM++*. Springer, 2007.
9. S. Capitani di Vimercati, S. Foresti, and P. Samarati. Authorization and access control. In M. Petkovic and W. Jonker, editors, *Security, Privacy, and Trust in Modern Data Management*, Data-Centric Systems and Applications, pages 39–53. Springer Berlin Heidelberg, 2007.
10. CITRIX. Available role based access control permissions for xenserver, http://support.citrix.com/article/ctx126441, 2013.
11. J. Crampton and G. Loizou. Administrative scope and role hierarchy operations. In *In Proceedings of Seventh ACM Symposium on Access Control Models and Technologies (SACMAT 2002)*, pages 145–154, 2002.
12. D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli. *Role-Based Access Control*. Artech House, Inc., 2003.
13. K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th international conference on Software engineering*, ICSE '05, pages 196–205, New York, NY, USA, 2005. ACM.
14. I. Foster, Z. Yong, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10, 2008.
15. L. Gong and X. Qian. Computational issues in secure interoperation, 1996.
16. A. Gouglidis and I. Mavridis. domRBAC: An access control model for modern collaborative systems. *Computers & Security*, 31(4):540 – 556, 2012.
17. F. Hansen and V. Oleshchuk. Conformance checking of RBAC policy and its implementation. In R. Deng, F. Bao, H. Pang, and J. Zhou, editors, *Information Security Practice and Experience*, volume 3439 of *Lecture Notes in Computer Science*, pages 144–155. Springer Berlin / Heidelberg, 2005.
18. H. Hu and G. Ahn. Enabling verification and conformance testing for access control model. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, SACMAT '08, pages 195–204, New York, NY, USA, 2008. ACM.
19. V. C. Hu, D. R. Kuhn, and T. Xie. Property verification for generic access control models. In *Proceedings of the 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing - Volume 02*, EUC '08, pages 243–250, Washington, DC, USA, 2008. IEEE Computer Society.
20. V. C. Hu, D. R. Kuhn, T. Xie, and J. Hwang. Model checking for verification of mandatory access control models and properties. *International Journal of Software Engineering and Knowledge Engineering*, 21(1):103–127, 2011.
21. G. Hughes and T. Bultan. Automated verification of access control policies using a SAT solver. *Int. J. Softw. Tools Technol. Transf.*, 10(6):503–520, Oct. 2008.
22. J. Hwang, T. Xie, V. Hu, and M. Altunay. ACPT: A tool for modeling and verifying access control policies. In *Proceedings of the 2010 IEEE International Symposium on Policies for Distributed Systems and Networks*, POLICY '10, pages 40–43, Washington, DC, USA, 2010. IEEE Computer Society.
23. K. Jayaraman, V. Ganesh, M. Tripunitara, M. Rinard, and S. Chapin. Automatic error finding in access-control policies. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 163–174, New York, NY, USA, 2011. ACM.
24. H. JeeHyun, A. Mine, X. Tao, and H. Vincent. Model checking grid policies, https://sites.google.com/site/gridpolicyproject/home.
25. S. Jha, N. Li, M. Tripunitara, Q. Wang, and W. Winsborough. Towards formal verification of role-based access control policies. *IEEE Transactions on Dependable and Secure Computing*, 5:242–255, 2008.
26. P. Krapivsky and S. Redner. Network growth by copying. *Physical Review E*, 71(3):036118, 2005.
27. D. R. Kuhn and D. R. Kacker. Automated combinatorial test methods - beyond pairwise testing. 2010.
28. L. Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley Professional, 1st edition, 2002.
29. N. Li, J.-W. Byun, and E. Bertino. A critique of the ANSI standard on role-based access control. *IEEE Security and Privacy*, 5(6):41–49, 2007.
30. W. Li, H. Wan, X. Ren, and S. Li. A refined rbac model for cloud computing. In *Computer and Information Science (ICIS), 2012 IEEE/ACIS 11th International Conference on*, pages 43–48, 2012.
31. T. Mather, S. Kumaraswamy, and S. Latif. *Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance*. Oreilly & Associates Inc, 2009.
32. Microsoft. Windows azure security guidance, http://www.windowsazure.com/en-us/develop/net/best-practices/security/, 2013.
33. M. Migliavacca, I. Papagiannis, D. M. Eyers, B. Shand, J. Bacon, and P. Pietzuch. Distributed middleware enforcement of event flow security policy. In *Middleware 2010*, pages 334–354. Springer, 2010.
34. NASA. Nebula's implementation of role based access control (RBAC),

http://nebula.nasa.gov/blog/2010/06/03/nebulas-implementation-role-based-access-control-rbac/, 2010.
35. NetworkX. Networkx, http://networkx.lanl.gov/, 2012.
36. NIST. Role based access control (RBAC) and role based security, http://csrc.nist.gov/groups/sns/rbac/index.html.
37. NIST. Combinatorial and pairwise testing, http://csrc.nist.gov/groups/sns/acts/, 2012.
38. NuSMV. A new symbolic model checker, http://nusmv.fbk.eu/.
39. E. Nuutila. *Efficient transitive closure computation in large digraphs*. PhD thesis, Acta Polytechnica Scandinavica, Helsinki University of Technology, 1995.
40. S. Oh and R. Sandhu. A model for role administration using organization structure, 2002.
41. OpenStack. Managing compute users, http://docs.openstack.org/diablo/openstack-compute/admin/content/managing-compute-users.html, 2013.
42. OpenStack. Users and projects, http://docs.openstack.org/diablo/openstack-compute/admin/content/users-and-projects.html, 2013.
43. M. Peter and G. Timothy. The NIST definition of cloud computing, September 2011.
44. D. Power, M. Slaymaker, and A. Simpson. Conformance checking of dynamic access control policies. In *Formal Methods and Software Engineering*, pages 227–242. Springer, 2011.
45. P. Purdom. A transitive closure algorithm. *BIT Numerical Mathematics*, 10:76–94, 1970. 10.1007/BF01940892.
46. R. Sandhu, V. Bhamidipati, and Q. Munawer. The arbac97 model for role-based administration of roles. *ACM Trans. Inf. Syst. Secur.*, 2(1):105–135, 1999.
47. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.
48. R. S. Sandhu and P. Samarati. Access control: Principles and practice. *IEEE Communications Magazine*, 32:40–48, 1994.
49. SAnToS Laboraroty. Spec patterns, responce property pattern, http://patterns.projects.cis.ksu.edu/, 2012.
50. 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 the sixth ACM symposium on Access control models and technologies*, pages 3–9. ACM, 2001.
51. B. Shafiq, J. B. D. Joshi, E. Bertino, and A. Ghafoor. Secure interoperation in a multidomain environment employing RBAC policies. *IEEE Trans. on Knowl. and Data Eng.*, 17(11):1557–1577, 2005.
52. SPIN. The SPIN model checker, http://spinroot.com/spin/.
53. H. Takabi, J. B. Joshi, and G.-J. Ahn. Security and privacy challenges in cloud computing environments. *Security & Privacy, IEEE*, 8(6):24–31, 2010.
54. Z. Tang, J. Wei, A. Sallam, K. Li, and R. Li. A new rbac based access control model for cloud computing. In R. Li, J. Cao, and J. Bourgeois, editors, *Advances in Grid and Pervasive Computing*, volume 7296 of *Lecture Notes in Computer Science*, pages 279–288. Springer Berlin Heidelberg, 2012.