IET Information Security

Research Article

IET Journals
The Institution of Engineering and Technology

Security analysis of ABAC under an administrative model

ISSN 1751-8709
Received on 26th January 2018
Accepted on 27th September 2018
E-First on 22nd November 2018
doi: 10.1049/iet-ifs.2018.5010
www.ietdl.org

Sadhana Jha[1], Shamik Sural[2] ✉, Vijayalakshmi Atluri[3], Jaideep Vaidya[3]

[1]*Advanced Technology Development Center, Indian Institute of Technology, Kharagpur, India*
[2]*Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, India*
[3]*Management Science and Information Systems Department, Rutgers University, USA*
✉ *E-mail: shamik@cse.iitkgp.ernet.in*

Abstract: In the present-day computing environment, where access control decisions are often dependent on contextual information like the location of the requesting user and the time of access request, attribute-based access control (ABAC) has emerged as a suitable choice for expressing security policies. In an ABAC system, access decisions depend on the set of attribute values associated with the subjects, resources, and the environment in which an access request is made. In such systems, the task of managing the set of attributes associated with the entities as well as that of analysing and understanding the security implications of each attribute assignment is of paramount importance. Here, the authors first introduce a comprehensive attribute-based administrative model, named as AMABAC (Administrative Model for ABAC), for ABAC systems and then suggest a methodology for analysing the security properties of ABAC in the presence of the administrative model. For performing analysis, the authors use *μZ*, a satisfiability modulo theories-based model checking tool. The authors study the impact of the various components of ABAC and AMABAC on the time taken for security analysis.

1 Introduction

In recent years, attribute-based access control (ABAC) [1] is increasingly getting prominence due to its support for making fine grained access decisions as well as its ability to handle situations where the identity of the requesting users is not known a priori. An ABAC system includes a well-defined set of subject attributes (subjects represent the entities that can make an access request), object attributes (objects represent the resources to be protected against unauthorised accesses), and environmental attributes (environment represents the context in which an access request is made) along with a set of possible operations that can be performed in the system. To make authorisation decisions, a rule base consisting of a set of authorisation rules (hereinafter referred to simply as *rules*) is maintained. An authorisation rule essentially provides the required set of attributes a subject needs to possess in order to perform an operation on an object in a given environmental condition.

In any organisation, security policies are used to specify how access is managed and who may access what resources under which circumstances. An access control model is an abstraction that supports a certain class of policy specifications. Security policies of organisations can be enormous, often involving hundreds of subjects and thousands of resources. Moreover, these policies evolve over time and it is necessary to reflect every change they undergo, by making proper modifications in the component of the underlying access control model. However, due to the complex interactions among the components of an access control model, it becomes difficult to understand and analyse the security implications of each modification made. Therefore, prior to the deployment of an access control system, it is of utmost importance to regularise the process of modifications that can be made in the corresponding access control model, and also to have a formal means for understanding and analysing the security properties of the model. There is limited work [2, 3] on the development of administrative models for ABAC as well as ABAC security analysis in the presence of its administrative model [2, 3]. Owing to the absence of a complete administrative model, there is no reported work on comprehensive security analysis for ABAC.

In this paper, we first propose an attribute-based administrative model named as AMABAC for ABAC systems. We then introduce a methodology for performing security analysis of ABAC in the presence of this administrative model. Although various security properties can be defined for an ABAC system, in this work, we consider *safety* and *liveness* properties. A safety property can be stated as: '*Whether a subject s is able to perform an operation op on an object o in the environmental condition e*'. A liveness property, on the other hand, is of the form '*Whether there is at least one subject who is authorized to perform operation op in the system*'. For carrying out such security analysis, we use the *μZ* tool [4], which is developed as a part of the Z3 [5] satisfiability modulo theories (SMT) solver and is a freely available tool from *Microsoft*.

The rest of the paper is organised as follows. Preliminaries on the various components of ABAC are briefly described in Section 2. We introduce the proposed administrative model for ABAC in Section 3. Section 4 gives details on the formal representation of the various components of ABAC as well as AMABAC. Section 5 presents our methodology for security analysis of ABAC using *μZ*. Section 6 provides the results of experimental evaluation of the proposed approach. Related literature is reviewed in Section 7. Finally, we conclude the paper and suggest several possible directions for future research in Section 8.

2 Preliminaries

In this section, we introduce the notations used to represent the various components of ABAC [1]. The right-hand side rectangular box of Fig. 1 shows a conceptual diagram for ABAC. In the ABAC model, $\mathcal{S}$, $\mathcal{O}$, and $\mathcal{E}$ denote a set of authorised subjects, objects, and environmental conditions, respectively. $\mathcal{S}_\mathcal{A}$, $\mathcal{O}_\mathcal{A}$, and $\mathcal{E}_\mathcal{A}$ denote a set of subject, object, and environmental attributes, respectively. Each attribute $a \in \{\mathcal{S}_\mathcal{A} \cup \mathcal{O}_\mathcal{A} \cup \mathcal{E}_\mathcal{A}\}$ is associated with a set $\mathfrak{R}_a$ of possible values it can acquire. There are three sets of functions $\mathcal{F}_\mathcal{S}$, $\mathcal{F}_\mathcal{O}$, and $\mathcal{F}_\mathcal{E}$, where $\mathcal{F}_\mathcal{X} = \{f^i_\mathcal{X} \mid 1 \le i \le |\mathcal{X}_\mathcal{A}|^*$ [$^*\mathcal{X}$ can be either $\mathcal{S}$, $\mathcal{O}$, or $\mathcal{E}$. i represents the index of an attribute in an ordered list of attributes]$\}$, each of whose elements is a function of the form $f^i_\mathcal{X}: \mathcal{X} \rightarrow \mathfrak{R}^i_\mathcal{X}$, representing the set of subject, object, and environmental attribute assignments in the system.

The set of attribute name–value pairs associated with a subject s (respectively, object o and environmental condition e) is given by the expression *sattr*(s) [respectively, *oattr*(o) and *eattr*(e)]. Δ is a

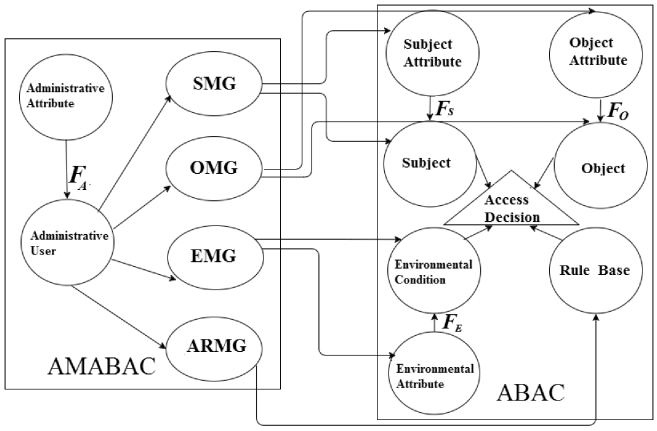

Fig. 1 *Conceptual diagram of user ABAC and AMABAC*

set of all possible operations allowed in the system. $\mathscr{P}$ denotes the set of all access control rules in the system. Each rule $r_i \in \mathscr{P}$ is of the form $\langle SA, OA, EA, op\rangle$, where SA, OA, and EA denote subject, object, and environmental attribute conditions, respectively. A subject attribute condition (respectively, object and environmental attribute condition) is made up of a set of subject attribute name–value pairs (respectively, object and environmental attribute name–value pairs). $op \in \Delta$ is an operation.

3 Administrative model for ABAC

In this section, we describe the proposed administrative model for ABAC, named as AMABAC (Administrative Model for ABAC). A conceptual diagram of AMABAC is shown in the left-hand side rectangular box of Fig. 1, which also depicts the interaction between AMABAC and user ABAC.

3.1 Overview of AMABAC

The core components of AMABAC include a set $\hat{\mathscr{U}}$ of authorised administrative users, a set $\mathscr{A}$ of administrative attributes, where each attribute $a \in \mathscr{A}$ is associated with a set $\mathfrak{R}_a$ of possible values it can acquire, a set of administrative attribute assignments given by the set of functions $\mathscr{F}_{\mathscr{A}} = \{f^i_{\hat{u}} \mid \hat{u} \in \hat{\mathscr{U}}, 1 \le i \le |\hat{\mathscr{A}}|\}$, each of whose elements is a function of the form $f^i_{\hat{u}}: \hat{\mathscr{U}} \to \mathfrak{R}^i_{\hat{a}}$, and an administrative rule base $\mathscr{AP}$ comprised of a set of administrative relations. The set of attribute name–value pairs associated with an administrative user u is given by the expression $attr(a)$.

Each administrative relation $\mathrm{Re}_i \in \mathscr{AP}$ is of the form $\langle ac, Par\rangle$, where ac denotes an administrative attribute condition. It represents a set of administrative attribute name–value pairs. Par represents an optional set of parameters that is passed as argument to the relation Re_i. Each relation Re_i defined in AMABAC is associated with an administrative command C_i. While an administrative relation defines the set of attribute assignments that an administrative user is required to be associated with, in order to modify certain component of a user ABAC system, an administrative command is required to be executed to perform the actual modification.

Each administrative command C_i is of the form $C_i = \langle pre, post, P\rangle$, where pre, $post$, and P, respectively, represent a set of one or more preconditions, a set of one or more postconditions, and an optional set of parameters that is passed as argument to the command C_i. For successful execution of a command $C_i = \langle pre, post, P\rangle$, the set of conditions in the set pre need to be satisfied prior to the execution of C_i and the set of conditions defined in the set $post$ should hold after the execution of C_i.

Consider an administrative user a who attempts to execute an administrative command c of the form $\langle pre, post, P\rangle$. Whether a is allowed to execute c or not is checked using Algorithm 1. It takes the identity of the administrative user a and a command c as input and checks whether u can execute c or not.

Algorithm 1: CHECK_COMMAND(a, $c = \langle pre, post, P\rangle$)

1: Search a rule of the form $\mathrm{Re}_i(ac, Par) \in \mathscr{AP}$, such that c is defined for the relation Re_i
2: **if** ($ac \subseteq attr(a)$) **then**
3: **for** ($precondition \in pre$) **do**
4: **if** precondition is *false* **then**
5: '*Access denied, precondition does not hold*'
6: EXIT
7: **end if**
8: **end for**
9: print '*Command can be executed successfully*'
10: EXIT
11: **end if**
12: print '*Administrative attribute condition not satisfied*'

In the following subsections, we give a brief description of the administrative relations and commands included in AMABAC.

3.2 Administrative relations and commands in AMABAC

There are 20 administrative relations and commands included in AMABAC. Based on the set of components, they are meant to modify, the set of relations in AMABAC is partitioned into four different groups, namely subject modification group (SMG), object modification group (OMG), environment modification group (EMG), and authorisation rule modification group (ARMG). Administrative relations included in SMG are meant for modifying the set of subjects and subject attribute-related components of a user ABAC system. Similarly, relations in OMG, EMG, and ARMG, respectively, are meant for modifying object, environmental condition, and authorisation rules-related components of a user ABAC system. As mentioned in the previous section, all the relations are associated with corresponding administrative commands.

Figs. 2–5 provide a summary of all the administrative relations included in AMABAC along with their associated commands as well as pre and $post$ conditions. Each relation $\mathrm{Re}_i = \langle ac, Par\rangle$ shown in these figures authorises the administrators to perform modification only into that component of the system for which Re_i is meant for. For instance, consider the administrative relation $can\_insert\_sub(ac)$ (refer to Fig. 2) meant for adding new subjects into the system. If a system administrator satisfies the attribute condition given by ac, then only she is authorised to add new subjects into the system.

4 Formal representation of ABAC and AMABAC

In this section, we show how ABAC and AMABAC systems can be formally represented. This representation can be used as the

Re_1	$can\_insert\_sub(ac)$
C_1	$insert\_sub(a, s)$
Preconditions	$ac \subseteq attr(a), s \notin \mathcal{S}$
Postconditions	$\mathcal{S} = \mathcal{S} \cup \{s\}$
Re_2	$can\_remove\_sub(ac)$
C_2	$remove\_sub(a, s)$
Preconditions	$ac \subseteq attr(a), s \in \mathcal{S}$
Postconditions	$\mathcal{S} = \mathcal{S} \setminus \{s\}$
Re_3	$can\_insert\_sub\_attr(ac)$
C_3	$insert\_subject\_attr(a, at)$
Preconditions	$ac \subseteq attr(a), at \notin \mathcal{S}_\mathcal{A}$
Postconditions	$\mathcal{S}_\mathcal{A} = \mathcal{S}_\mathcal{A} \cup \{at\}$
Re_4	$can\_modify\_sub\_attr\_range\_\mathcal{S}_\mathcal{A}(ac)$
C_4	$modify\_sub\_attr\_range(a, i, v)$
Preconditions	$ac \subseteq attr(a), v \notin \mathfrak{R}^i_\mathcal{S}$
Postconditions	$\mathfrak{R}^i_\mathcal{S} = \mathfrak{R}^i_\mathcal{S} \cup \{v\}$
Re_5	$can\_assign\_value\_sub\_attr(ac, sc)$
C_5	$assign\_value\_sub\_attr(a, s, i, v)$
Preconditions	$ac \subseteq attr(a), sc \subseteq sattr(s), v \in \mathfrak{R}^i_\mathcal{S}, s \in \mathcal{S}$
Postconditions	$f^i_\mathcal{S}(s) = v$
Re_6	$revoke\_value\_sub\_attr(ac, sc)$
C_6	$revoke\_value\_sub\_attr(a, s, i)$
Preconditions	$ac \subseteq attr(a), sc \subseteq sattr(s), s \in \mathcal{S}, f^i_\mathcal{S}(s) = v$
Postconditions	$f^i_\mathcal{S}(s) \rightarrow \phi$

Fig. 2 *Administrative relations and commands in SMG*

Re_7	$can\_insert\_object(ac)$
C_7	$insert\_object(a, o)$
Preconditions	$ac \subseteq attr(a), o \notin \mathcal{O}$
Postconditions	$\mathcal{O} = \mathcal{O} \cup \{o\}$
Re_8	$can\_remove\_object(ac)$
C_8	$remove\_object(a, o)$
Preconditions	$ac \subseteq attr(a), o \in \mathcal{O}$
Postconditions	$\mathcal{O} = \mathcal{O} \setminus \{o\}$
$Relation_9$	$can\_insert\_obj\_attr(ac)$
C_9	$insert\_object\_attr(a, at)$
Preconditions	$ac \subseteq attr(a), at \notin \mathcal{O}_\mathcal{A}$
Postconditions	$\mathcal{O}_\mathcal{A} = \mathcal{O}_\mathcal{A} \cup \{at\}$
Re_{10}	$can\_modify\_range\_obj\_attr(ac)$
C_{10}	$modify\_range\_obj\_attr(a, i, v)$
Preconditions	$ac \subseteq attr(a), v \notin \mathfrak{R}^i_\mathcal{O}$
Postconditions	$\mathfrak{R}^i_\mathcal{O} = \mathfrak{R}^i_\mathcal{O} \cup \{v\}$
Re_{11}	$can\_assign\_value\_obj\_attr(ac, oc, oc_{pre})$
C_{11}	$assign\_value\_obj\_attr(a, s, V)$
Preconditions	$ac \subseteq attr(a), oc_{pre}\ oattr(o)$
Postconditions	for each attribute $a_i \in oc, f^i_O(o) = v^i$
Re_{12}	$revoke\_value\_\mathcal{O}_\mathcal{A}(ac)$
C_{12}	$revoke\_value\_obj\_attr(a, o, i)$
Preconditions	$ac \subseteq attr(a), o \in \mathcal{O}$
Postconditions	$f^i_O(o) \rightarrow \phi$

Fig. 3 *Administrative relations and commands in OMG*

basis for modelling both ABAC and AMABAC using any first-order logic language. We also present the logical expressions that define the impact of the different relations of AMABAC on the various components of an ABAC system.

4.1 Representation of ABAC components

The set of all subjects (respectively, objects and environmental conditions) is denoted as *subjs* (respectively, *objs* and *envs*). To represent the *i*th member of these sets, notations of the form s_i, o_i, and e_i are used. The set of subject (respectively, object and environmental) attributes is denoted as *subj_attr* (respectively, *obj_attr* and *env_attr*). To represent the *i*th member of these sets, notations of the form sa_i, oa_i, and ea_i are used. The range set of an attribute a is denoted as $Range(a)$. Subject–subject attribute assignments, object–object attribute assignments, and environment–environmental attribute assignments are represented using relations of the form, *subj_attr_assignment* (s_i, sa_j), *obj_attr_assignment* (o_i, oa_j), and *env_attr_assignment*(e_i, ea_j), respectively.

The set of all attribute name–value pairs associated with a subject s_i, an object o_i, and an environmental condition e_i is given by the expressions $sattr(s_i)$, $oattr(o_i)$, and $eattr(e_i)$, respectively. The set of all permissible operations is denoted as *opr*. To represent the *i*th operation, notations of the form op_i are used. *Rules* denotes the set of all authorisation rules in the system. A notation of the form $rule_i$ represents the *i*th tuple in the *Rules* relation. Expressions of the form $rule_i$ (*sc*), $rule_i$ (*oc*), $rule_i$ (*ec*), and $rule_i$ (*op*), respectively, return the set of subject attribute conditions, object attribute conditions, environmental attribute conditions, and the operation name associated with the *i*th rule.

4.2 Representation of AMABAC components

The set of all administrative users and administrative attributes are denoted as *AdminUser* and *AdminAttr*, respectively. To represent the *i*th member of these sets, notations of the form au_i and aa_i are used. An administrator–administrative attribute assignment relation is represented using the relation of the form *AdminAttrAssignment*(au_i, aa_j). It returns the value of the *j*th attribute for the *i*th administrative user. The set of all administrative attribute name–value pairs associated with an administrative user au_i is given by $attr(au_i)$.

4.2.1 Representation of administrative relations in SMG:

can_insert_subj(ac): New subjects can be added to the set *subjs*, if the following condition holds:

$$\exists_{ac \in can\_insert\_subj} \left| \exists_{au_i \in AdminUser} \right| ac \subseteq attr(au_i) \tag{1}$$

. Successful execution of a command associated with the relation *can_insert_subj* causes addition of a new tuple s_i into the relation *subjs*. The modified content of the relation *subjs* is

$$subjs \rightarrow subjs \cup \{s\} \tag{2}$$

.

can_ remove _subj(*ac*): Existing subjects can be removed from the set *subjs*, if the following condition holds:

$$\exists_{ac \in can\_remove\_subj} \left| \exists_{au_i \in AdminUser} \right| ac \subseteq attr(au_i) \tag{3}$$

. Successful execution of a command associated with a *can_remove_subj* relation causes removal of a tuple from the relation *subjs*. The modified content of *subjs* is *subjs*→ *subjs* \ {*s*}.

can_insert_subj_attr(*ac*): New subject attributes can be added to the set *subj_attr* if the following condition holds:

$$\exists_{ac \in can\_insert\_subj\_attr} \left| \exists_{au_i \in AdminUser} \right| ac \subseteq attr(au_i) \tag{4}$$

. Successful execution of a command associated with the relation *can_insert_subj_attr* causes addition of a new tuple *sa* into the relation *subj_attr*. The modified content of the relation *subj_attr* is *subj_attr* → *subj_attr*∪ {*sa*}.

can_modify_subj_attr_range(*ac*): Range sets associated with the subject attributes in the system can be modified if the following condition holds:

$$\exists_{(ac) \in can\_modify\_subj\_attr\_range} \left| \exists_{au_i \in AdminUser} \right| ac \subseteq attr(au_i) \tag{5}$$

. Successful execution of a command associated with the relation *can_modify_subj_attr_range*, meant for modifying the range set of a subject attribute *sa*, causes addition of a new tuple *v* to the relation *Range*(*sa*). The modified range set for the subject attribute *sa* is $Range(sa) \rightarrow Range(sa) \rightarrow v$.

Re_{13}	$can\_insert\_env(ac)$
C_{13}	$insert\_env(a, e)$
Preconditions	$ac \subseteq attr(a), e \notin \mathcal{E}$
Postconditions	$\mathcal{E} = \mathcal{E} \cup \{e\}$
Re_{14}	$can\_remove\_env(ac)$
C_{14}	$remove\_env(a, e)$
Preconditions	$ac \subseteq attr(a), e \in \mathcal{E}$
Postconditions	$\mathcal{E} = \mathcal{E} \setminus \{e\}$
Re_{15}	$can\_insert\_env\_attr(ac)$
C_{15}	$insert\_env\_attr(a, at)$
Preconditions	$ac \subseteq attr(a), at \notin \mathcal{E}_{\mathcal{A}}$
Postconditions	$\mathcal{E}_{\mathcal{A}} = \mathcal{E}_{\mathcal{A}} \cup \{at\}$
Re_{16}	$can\_modify\_range\_env\_attr(ac)$
C_{16}	$modify\_range\_env\_attr(a, i, v)$
Preconditions	$ac \subseteq attr(a), v \notin \mathfrak{R}^i_{\mathcal{E}}$
Postconditions	$\mathfrak{R}^i_{\mathcal{E}} = \mathfrak{R}^i_{\mathcal{E}} \cup \{v\}$
Re_{17}	$can\_assign\_value\_env\_attr(ac, ec, ec_{pre})$
C_{17}	$assign\_value\_env\_attr(a, s, V)$
Preconditions	$ac \subseteq attr(a), ec_{pre}\ eattr(e)$
Postconditions	for each attribute $a_i \in ec, f^i_E(e) = v^i$
Re_{18}	$revoke\_value\_\mathcal{E}_{\mathcal{A}}(ac)$
C_{18}	$revoke\_value\_env\_attr(a, e, i)$
Preconditions	$ac \subseteq attr(a), e \in \mathcal{E}$
Postconditions	$f^i_E(e) \rightarrow \phi$

Fig. 4 *Administrative relations and commands in EMG*

Re_{19}	$can\_add\_rule(ac)$
C_{19}	$add\_rule(a, r)$
Preconditions	$ac \subseteq attr(a), r \notin \mathcal{P}$
Postconditions	$\mathcal{P} = \mathcal{P} \cup \{r\}$
Re_{20}	$can\_remove\_rule(ac)$
C_{20}	$remove\_rule(a, r)$
Preconditions	$ac \subseteq attr(a), r \in \mathcal{P}$
Postconditions	$\mathcal{P} = \mathcal{P} \setminus \{r\}$

Fig. 5 *Administrative relations and commands in ARMG*

can_assign_value_subj_attr(*ac*, *sc*): New values can be assigned to the attributes associated with a subject *s* of a system, if the following condition holds:

$$\exists_{(ac,sc) \in can\_assign_value\_subj\_attr} \exists_{au_i \in AdminUser} ac \subseteq attr(au_i) \wedge (sc \subseteq sattr(s)) \quad (6)$$

. Successful execution of a command associated with the relation *can_assign_value_subj _attr*(*ac*, *sa*), meant for assigning value *v* to the *i*th attribute of subject *s*, causes addition of a new tuple in the relation *sattr*(s) associated with the subject *s*. The modified *sattr*(*s*) for the subject *s* is $sattr(s) \rightarrow sattr(s) \cup \{(i, v)\}$.

can_remove_value_subj_attr(*ac*): Attribute values assigned to the attributes associated with a subject *s* of a system can be revoked, if the following condition holds:

$\exists_{(ac) \in can\_revoke_value\_subj\_attr} \exists_{au_i \in AdminUser} ac \subseteq attr(au_i)$.

Successful execution of a command associated with the relation *can_remove_value_subj_attr*(*ac*), meant for revoking value *v* from the *i*th attribute of subject *s*, causes removal of an existing tuple from the relation *sattr*(*s*) associated with the subject *s*. The modified *sattr*(*s*) for the subject *s* is $sattr(s) \rightarrow sattr(s) \setminus \{(i, v)\}$.

Formal notations used for the set of administrative relations included in OMG and EMG are similar to that used for SMG and, hence, are not repeated. In the next subsection, we therefore provide formal representations for the set of administrative relations included in ARMG.

4.2.2 Representation of administrative relations in ARMG: can_add_rule(ac): A new rule *r* can be added to the set *Rules*, if the following condition holds:

$$\exists_{(ac) \in can\_insert\_subj} \exists_{au_i \in AdminUser} ac \subseteq attr(au_i) \quad (7)$$

. Successful execution of a command associated with the relation *can_add_rule* causes addition of a new tuple into the relation *Rules*. The modified content of the relation is $Rules \rightarrow Rules \cup \{r\}$.

can_remove_rule(*ac*): An existing rule *r* can be removed from the set *Rules*, if the following condition holds:

$$\exists_{(ac) \in can\_remove\_subj} \exists_{au_i \in AdminUser} ac \subseteq attr(au_i) \quad (8)$$

. Successful execution of a command associated with the relation *can_remove_rule* causes removal of a tuple from the relation *Rules*. The modified content of the relation *Rules* is $Rules \rightarrow Rules \setminus \{r\}$.

4.3 Representation of security properties

Representation of safety property: To represent a safety property, a relation of the form *UserCanPerformOperation*(s_i, op_j, o_k, e_m) is used. This relation returns *true* if the following condition holds:

$$\exists_{rule \in Rules} rule(sc) \subseteq sattr(s_i) \wedge rule(oc) \subseteq oattr(o_k) \wedge rule(ec) \subseteq eattr(e_m) \wedge rule(op) = op_j$$

Representation of liveness property: To represent a liveness condition, a relation named as *liveness* (op_i) is used. This relation returns *true* if the following condition holds:

$$\exists_{rule \in Rules, s \in subjs} rule(sc) \subseteq sattr(s) \wedge rule(op) = op_i$$

Thus, the problem of security analysis is to determine if the above-mentioned safety and liveness properties hold for an ABAC system with its user ABAC components defined in Section 4.1 in the presence of the administrative components and relations defined in Section 4.2.

5 Analysis of security properties

In this section, we demonstrate how security analysis of ABAC systems can be performed using μZ, an SMT-based tool that accepts a model specified in Datalog as input and verifies the properties of the input model. We also show the effect of the presence of the administrative relations in AMABAC on the security properties of a user ABAC system.

5.1 Overview and running example

For performing security analysis, we consider ABAC in a hospital set up as a running example. Let there be three subjects ($\mathcal{S}$ = {*John, Mary, Charles*}), three subject attributes ($\mathcal{S}_{\mathcal{A}}$ = {*qualification, designation, specialisation*}), three objects ($\mathcal{O} = \{O_1, O_2, O_3\}$), two object attributes ($\mathcal{O}_{\mathcal{A}}$ = {*purpose, department*}), two environmental conditions ($\mathcal{E} = \{E_1, E_2\}$), and two environmental attributes ($\mathcal{E}_{\mathcal{A}}$ = {*access time, access IP*}) in the initial state of a user ABAC system. The range of values associated with each attribute is shown in Table 1. Attribute assignments are shown in Table 2. A representative set of authorisation rules for the example user ABAC system is shown in Table 3. The administrative ABAC consists of two administrative users {*Alice*, *Stephen*} and two administrative attributes {*certified*, *designation*}. The administrative attribute assignments are shown in Table 2. The set of administrative relations and commands are defined in Tables 4 and 5.

It may be noted that we define the example administrative relations and commands in two different categories. While the first category (denoted as $category_1$) consists of only those administrative relations which add new elements to the system, the second one (denoted as $category_2$) consists of those that remove existing elements from the system. The motivation behind such partitioning of the administrative relations is explained later in this section.

We next describe how analysis of the given ABAC system can be done both in the absence and the presence of the administrative

Table 1 Attribute names along with their range set for the example user ABAC system

Subject attributes
qualification: {*MD, MBBS, graduate*}
designation: {*doctor*, *receptionist*}
specialisation: {*cardiology, orthopaedics*}
object attributes
purpose: {*medical_report, patient_list*}
department: {*cardiology, orthopaedics*}
environmental attributes
access time: {10.00 AM–06.00 PM, 06.00 PM–02.00 AM}
access IP: {*private*, *public*}
administrative attributes
certified: {CISSP, CISM}
designation: {CSO, DSO}

Table 2 Attribute assignments for the example user and administrative ABAC system

subject attribute assignments			
Subject	Qualification	Designation	Specialisation
John	MD	doctor	cardiology
Mary	MBBS	doctor	orthopaedics
Charles	graduate	receptionist	

Object attribute assignments		
Object	Purpose	Department
O_1	medical_report	cardiology
O_2	patient_list	
O_3	medical_report	orthopaedics

Environmental attribute assignments		
Environmental condition	Access_time	Access_IP
E_1	10.00 AM–06.00 PM	private
E_2	06.00 PM–02.00 AM	private

Administrative attribute assignments		
Administrator	Certified	Designation
Alice	CISSP	CSO
Stephen	CISM	DSO

ABAC. For performing security analysis, the components of ABAC and AMABAC along with the set of administrative relations are modelled using a formal language. The model thus generated is fed as input to some formal verification tool, and serves as the knowledge base (KB) for the analysis. To check whether a security property holds in the input model, a security query is presented to the tool, which then checks for the validity of the query in the presence of the KB. As mentioned in Section 1, in this work, we use the μZ verification tool, which is an SMT-based solver. It accepts a model specified in Datalog notation. When presented with a security query, it returns *sat* as output if the query holds true, otherwise returns *unsat*.

5.2 Safety analysis

For safety analysis, initially, we define a set of conditions which are considered to be *safe*, and then check whether the defined conditions hold, both in the absence and presence of administrative ABAC. While performing safety analysis, we consider only the set

Table 3 Authorisation rules for the example user ABAC system

r_1: 〈(qualification = *MD*, designation = *any*, specialisation = *cardiology*), (purpose = *medical_report*, department = *cardiology*), (*access time* = *10.00AM–06.00PM*, access IP = *private*), *delete*〉
r_2: 〈(qualification = *MBBS*, designation = *doctor*, specialisation = *orthopaedics*), (purpose = *medical_report*, department = *orthopaedics*), (access time = *10.00AM −06.00PM*, access IP = *private*), *update*〉
r_3: 〈(qualification = *any*, designation = *receptionist*, specialisation = *any*), (purpose = *patient_list*, department = *any*), (access time = *any*, access IP = *any*), *update*〉

Table 4 Illustrative set of category$_1$ relations and commands

<*can_insert_subject*(certified = *CISSP*, designation = *DSO*)>
<*can_assign_value_sub_attr*((certified = *CISSP*, designation = *CSO*),*3*)>
<*can_assign_value_env_attr*((certified = *CISSP*, designation = *CSO*), *2*)>
<*can_add_rule*((certified = *CISM*, designation = *DSO*)>
<*insert_subject*(*Alice, harry*)>
<*assign_value_sub_attr*(Alice, John, orthopaedics)>
<*assign_value_env_attr*(*Alice, E$_1$, public*)>
<*add_rule*(Stephen 〈(qualification = *any*, designation = *receptionist*, specialisation = *any*), (purpose = *patient_list*, department = *any*), (access time = *any*, access IP = *any*), *update*〉)>

Table 5 Illustrative set of category$_2$ relations and commands

<*can_remove_object*((certified = *CISM*, designation = *DSO*)>
<*can_remove_rule*((certified = *CISM*, designation = *DSO*)>
<*remove_object*(*Stephen* O_1)>
<*remove_rule*(*Stephen*, r_2)>

of administrative relations and commands included in *category*$_1$. The example safety condition that we use for analysis can be stated as '*Only doctors with the qualification MD should be able to delete the medical reports of the patients*'. To understand the impact of the administrative ABAC on the safety of a system, we ask the following safety queries to the solver, both in the absence and presence of administrative ABAC:

- *safety*(*Mary*, O_1, *any*, *delete*). This safety query asks the solver to check whether *Mary* is allowed to perform *delete* operation on the object O_1 in any environmental condition.
- *safety*(*Mary*, O_3, *any*, *delete*). This safety query asks the solver to check whether *Mary* is allowed to perform *delete* operation on the object O_3 in any environmental condition.

The system is considered to be in *safe* state only if the solver returns *unsat* for both of the above safety queries. In the absence of administrative ABAC, the solver returns *unsat* for both the above queries. On the other hand, in the presence of the administrative relations for the first query, i.e. *safety*(*Mary*, O_1, *any*, *delete*), the solver returns *unsat*, for the second query, i.e. *safety*(*Mary*, O_3, *any*, *delete*), it returns *sat*. It may be noted that, in the absence of the administrative ABAC, there is only one authorisation rule r_1 defined for *delete* operation and the subject attribute condition specified in r_1 is not satisfied by *Mary* and hence, she is not allowed to perform the delete operation. However, due to the presence of the administrative relations *can_add_rule*(*CISM, DSO*) and the administrative command <*add_rule*(Stephen 〈 (qualification = *any*, designation = *doctor*, specialisation =

Table 6 Experimental data showing comparison between time taken by the μZ and Alloy analyzer for safety analysis

$\lvert\mathcal{S}_{\mathcal{A}}\rvert$	$\lvert\mathcal{P}\rvert$	$\mathcal{A}_{\mathcal{R}}$	Alloy, ms	SMT, ms
4	5	5	17	<0.01
4	5	10	17.8	<0.01
4	10	5	18	<0.01
4	10	10	17	<0.01
8	5	5	23.8	<0.01
8	5	10	23.5	<0.01
8	10	5	24	<0.01
8	10	10	25	<0.01
12	15	15	51.6	0.01
12	15	20	52.3	0.01
12	20	15	53	0.01
12	20	20	53.8	0.02
16	15	15	79.2	0.02
16	15	20	79	0.03
16	20	15	79.1	0.03
16	20	20	79.4	0.03

orthopaedics), (purpose = *medical_report*, department = *orthopaedics*), (access time = *any*, access IP = *any*), *delete*〉)> (shown in Table 4) issued by Stephen, a new rule gets added to the system. The newly added rule specifies that all the *doctors*, regardless of their qualification, can perform *delete* operation, thus allowing Mary to perform the *delete* operation. It can thus be observed that execution of administrative commands can potentially take a system initially in a *safe* state into an *unsafe* state. Therefore, it is important that security analysis of the system is done prior to performing any actual modification in the system by the administrator.

Impact of excluding category$_2$ *relations:* The effectiveness of a test can be measured by measuring the *sensitivity* and *specificity* of the test. While sensitivity measures the number of *true-positive* results given by the test, specificity is used for determining the number of *true negatives* given by the test. In safety analysis, the main objective is to correctly identify whether a subject is getting a privilege to perform an operation or not, i.e. identifying true-positive cases. For instance, consider a safety query of the form (*John, O*$_3$*, any, delete*). This query intends to ask the solver whether the subject *John* can perform *delete* operation on object O_3. From the attribute assignments of John and the authorisation rules in the system, it can be observed that, *John* is associated with the appropriate set of attribute values so that he can perform the *delete* operation. When the solver is posed with the safety query safety(*John, O*$_3$*, any, delete*), it correctly returns *sat*.

The ability of John to perform the *delete* operation can be revoked from *John* only if some administrative commands either having the potential to remove *John* from the system (through *remove_subj* command), de-associate *John* from his existing attribute value assignments (through *revoke_sub_attr_value* command), or remove the access rule due to which *John* gets the permission to perform the delete operation (through *remove_rule* command) is considered while doing analysis. Since we do not consider any *category*$_2$ relation, the solver would always continue to return *sat* for the query *safety*(*John, O*$_3$*, any, delete*). While this would increase the number of *false-positive* results, it would not affect true-positive results, thus preserving the *specificity* of the used approach.

5.3 Liveness analysis

In liveness analysis, we pose a query of the form liveness(*op*), which asks the solver to check for the presence of at least one subject who is authorised to perform the operation *op* in the system. If any such rule exists, then the solver returns *sat*, otherwise, it returns *unsat*. A *sat* response from the solver indicates that the system is *live*. An *unsat* response indicates that the operation *op* is *unreachable* and cannot be performed by any subject in the system. In the above example hospital system, we may ask a liveness query *liveness*(*prepare*). Initially, in the presence of only the basic user ABAC components, the solver would return *sat*. Next, we add a set of administrative relations given in Table 5 as input to the μZ. In the presence of these administrative relations, the solver would return *unsat*, indicating that none of the subjects in the system is allowed to perform the *delete* operation. It can be observed that, although in the initial state of the system, *John* is associated with the set of attributes required for performing a delete operation in the system, still the solver returns *unsat*. This is due to the presence of the administrative relation *can_remove_rule* in the administrative rule base, which authorises *Stephen* to remove rule r_2, thus making the *delete* operation unreachable.

Impact of excluding category$_1$ *relations:* When a liveness query of the form liveness(*op*) is posed to the solver, it searches for the presence of a tuple that establishes a relationship between a set of subject attributes and an operation *op* in the *access_rule* relation defined in the Datalog specification for an ABAC system. An operation *op* is considered to be *unreachable* only if the solver fails to find any such tuple. Now, consider a case where for an operation name *op*, there exists a tuple that associates *op* with a set of entity attributes. The solver would return *sat* indicating that the system is in *secured* state. The system currently in a secure state can transit to an unsecure state (in which the operation *op* is unreachable) only if administrative relations that either allow removal of the tuples from the attribute assignment relation or removal of the tuples from the access rule relation are considered while doing the analysis. Since the set of administrative relations included in *category*$_1$ are not meant for removal of any existing tuple from any of the relations, they do not have the potential to make an operation which was reachable earlier, an unreachable one. Therefore, they have no impact on the answer to the liveness query.

In this section, we have demonstrated how μZ can be effectively used for performing security analysis of ABAC in the presence of AMABAC. In the next section, we provide details of our experiments conducted to analyse the impact of various components of ABAC and AMABAC on the analysis time.

6 Experimental results

To study the performance of the proposed methodology, we have built a data generator that takes the cardinality of each component of ABAC and an AMABAC as input and generates an ABAC and AMABAC system as output. All implementation has been done using Java (7.0.1–17) on an Intel Xeon E5-2697 v2 processor @ 2.70 GHz with 256 GB RAM. The μZ tool, which is integrated as a part of Z3 4.0, is used for analysis.

A comparative analysis of the proposed technique with the methodology presented in [3] using Alloy analyzer is provided in Table 6. It is observed that while Alloy could analyse specification having at most 16 attribute values, there is no such restriction for μZ. Also, it can be observed from the table that even for attribute values <16, μZ outperforms Alloy.

The effect of the number of subjects and subject attribute values on liveness analysis is shown in Fig. 6. Since liveness analysis checks for the presence of access rules using which an authorised subject can perform an operation, we show only the effect of the number of access rules and subjects on analysis time. From the figure, it can be observed that although the analysis time increases with increase in number of subjects and subject attribute values, the overall analysis time is <0.05 s even for 400 users.

To study the scalability of μZ, we performed experiments with larger data sets. Table 7 shows the variation in time taken to generate the Datalog model corresponding to an ABAC specification and the time taken to perform analysis of the generated Datalog model with increase in the number of access rules and subject attribute values in a system. In the Datalog generation step, Datalog specification of only those subjects, objects, and environmental factors is generated which are mentioned in a given safety query, and then analysis is done for the generated reduced data specification. It can be observed that, even for larger data sets containing as many as 250 rules and 250 subject

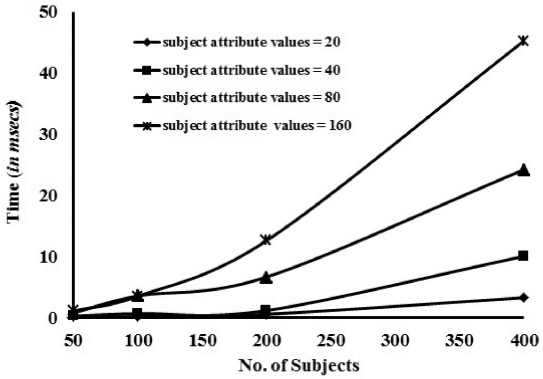

Fig. 6 *Variation in liveness analysis time with increase in the number of subjects and subject attribute values*

Table 7 Variation in Datalog generation time (DGT) (ms) and analysis time (AT) (ms) with increase in the number of access rules and subject attribute values

$\lvert\mathscr{P}\rvert$	$\lvert\mathscr{S}_{\mathscr{A}}\rvert = 100$		$\lvert\mathscr{S}_{\mathscr{A}}\rvert = 150$		$\lvert\mathscr{S}_{\mathscr{A}}\rvert = 200$		$\lvert\mathscr{S}_{\mathscr{A}}\rvert = 250$	
	DGT	AT	DGT	AT	DGT	AT	DGT	AT
100	49	0.48	51	0.48	54	0.51	57	0.5
150	49.7	0.51	53	0.54	56.2	0.53	58.4	0.55
200	50	0.55	57	0.53	57.6	0.58	61	0.59
250	55	0.56	61	0.57	62	0.63	65	0.62

attribute values, the total time required is low, indicating that our modelling and analysis methodology is quite efficient. Furthermore, while the results are reported with respect to variation in the number of subjects and subject attributes, when analysis from the perspective of objects is desired, the same scalability results hold with respect to the number of objects and object attributes.

7 Related work

A significant amount of work has been done towards the development of administrative models for role-based access control (RBAC) [6] and its variants [7–9]. ARBAC97 [10], one of the first such models, includes URA97, PRA97, and RRA97 to administer the user-role assignment, role-permission assignment, and role hierarchy of an RBAC system. AMTRAC, a complete administrative model for a temporal extension of RBAC (TRBAC) [8], is discussed in [11]. Other administrative models for the variants of RBAC include [12–14]. Along with the development of administrative models, significant effort can also be seen towards security analysis of RBAC and its variants [15–19].

An initial effort to develop an administrative model for ABAC and performing its analysis can be found in [2]. It presents an administrative model named as generalised user-role assignment model (GURA), comprised of a set of administrative requests and a set of administrative rules. Although relations defined in GURA can be used for administering user attribute assignments, a shortcoming is that it relies on the set of relations defined in the URA97 of ARBAC97 [10]. In our prior work [3], we have introduced an attribute-based administrative model named as ADABAC as well as a methodology for performing security analysis using the Alloy analyzer. However, ADABAC can only be used to manage the subjects and subject-related components of an ABAC system and it does not include any component for managing object and environmental factors or the policies. Furthermore, while Alloy can be used for analysing small ABAC systems, it fails to cater to the need of analysing large ABAC systems. As such, the current work presents a comprehensive administrative model which can be used for managing all aspects of a complete ABAC system. Furthermore, the use of SMT solvers makes security analysis much more scalable when compared with SAT-based solvers. Apart from the development of administrative models, some of the notable work can also be seen in the direction of specifying constraints and mining policies in ABAC [20–26]. There is also a recent work on analysing how separation duty can be enforced in ABAC systems [27].

8 Conclusions

In this paper, initially, we have proposed an administrative model for ABAC named as AMABAC and then presented a methodology for performing a comprehensive security analysis of ABAC using the same administrative model. We have studied the impact of various components of ABAC and AMABAC on the time taken to answer safety and liveness queries. Also, a detailed comparative analysis of the proposed methodology with the work reported in [3] has been done.

We have made certain assumptions such as direct assignment of attribute values to the entities, fixed size authorisation rules etc. Performing analysis in the absence of such constraints would provide an even more practical insight into the security properties of an ABAC system. The approach defined in this paper itself can be extended by considering security constraints such as mutual exclusion and separation of duty. It is worthwhile noting that even though μZ outperforms Alloy in terms of scalability and flexibility, there is a limitation that it cannot highlight the specific point of security breach. Exploration of other verification tools which are capable of providing such information is also a potential direction for further research.

9 References

[1] Hu, V.C., Ferraiolo, D., Kuhn, R., *et al.*: 'Guide to attribute based access control (ABAC) definition and considerations'. NIST Special Publication, 2014

[2] Jin, X., Krishnan, R., Sandhu, R.: 'Reachability analysis for role-based administration of attributes'. Proc. of the 2013 ACM Workshop on Digital Identity Management, Berlin, Germany, 2013, pp. 73–84

[3] Jha, S., Sural, S., Atluri, V., *et al.*: 'An administrative model for collaborative management of ABAC systems and its security analysis'. Proc. of the IEEE 2nd Int. Conf. Collaboration and Internet Computing, Pittsburgh, USA, 2016, pp. 64–73

[4] Hoder, K., Bjørner, N., Moura, L.M.D.: 'μZ – an efficient engine for fixed points with constraints'. Proc. of the Conf. Tools and Algorithms for the Construction and Analysis of Systems, Saarbrücken, Germany, 2011, pp. 457–462

[5] Moura, L.M.D., Bjørner, N.: 'Z3: an efficient SMT solver'. Proc. of the 14th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems, Budapest, Hungary, 2008, pp. 337–340

[6] Sandhu, R., Coyne, J.E., Feinstein, H.L., *et al.*: 'Role-based access control models', *IEEE Comput.*, 1996, **29**, (2), pp. 38–47

[7] Aich, S., Mondal, S., Sural, S., *et al.*: 'Role based access control with spatio-temporal context for mobile applications', *Trans. Comput. Sci.*, 2009, **5430**, (4), pp. 177–199

[8] Bertino, E., Andrea, B.P., Ferrari, E.: 'TRBAC: a temporal role-based access control model', *ACM Trans. Inf. Syst. Security*, 2001, **4**, (3), pp. 191–233

[9] Bhatti, R., Ghafoor, A., Bertino, E., *et al.*: 'X-GTRBAC: an XML-based policy specification framework and architecture for enterprise-wide access control', *ACM Trans. Inf. Syst. Security*, 2005, **8**, (2), pp. 187–227

[10] Sandhu, R., Bhamidipati, V., Munawer, Q.: 'The ARBAC97 model for role-based administration of roles', *ACM Trans. Inf. Syst. Security.*, 1999, **2**, (1), pp. 105–135

[11] Sharma, M., Sural, S., Vaidya, J., *et al.*: 'AMTRAC: an administrative model for temporal role-based access control', *Comput. Secur.*, 2013, pp. 201–218

[12] Bhatti, R., Ghafoor, A., Bertino, E., *et al.*: 'X-GTRBAC admin: a decentralized administration model for enterprise wide access control'. Proc. of the 9th ACM Symp. Access Control Models and Technologies, Yorktown Heights, USA, 2004, pp. 78–86

[13] Sharma, M., Sural, S., Atluri, V., *et al.*: 'An administrative model for spatio-temporal role based access control'. Proc. of the International Conference on Information Systems Security, Kolkata, India, 2013, pp. 375–389

[14] Jiang, X., Su, M., Shen, W.: 'FARBAC: a fine-grained administrative RBAC model'. Proc. of the 7th IEEE Annual Information Technology, Electronics and Mobile Communication Conf., 2015, pp. 1–4

[15] Ninghui, N.L., Tripunitara, M.V.: 'Security analysis in role-based access control', *ACM Trans. Inf. Syst. Security*, 2006, **9**, (4), pp. 391–420

[16] Uzun, E., Atluri, V., Sural, S., *et al.*: 'Analyzing temporal role-based access control models'. Proc. of the ACM Symp. Access Control Models and Technologies, Newark, USA, 2012, pp. 177–186

[17] Mondal, S., Sural, S., Atluri, V.: 'Towards formal security analysis of gtrbac using timed automata'. Proc. of the 14th ACM Symp. on Access Control Models and Technologies, SACMAT '09, 2009, pp. 33–42 (ACM, New York, NY, 2009). Available at http://doi.acm.org/10.1145/1542207.1542214

[18] Jha, S., Sural, S., Vaidya, J., *et al.*: 'Security analysis of temporal RBAC under an administrative model', *Comput. Secur.*, 2014, **46**, pp. 154–172

[19] Jha, S., Sural, S., Vaidya, J., *et al.*: 'Temporal RBAC security analysis using logic programming in the presence of administrative policies'. Proc. of the

Int. Conf. on Information Systems Security, Hyderabad, India, 2014, pp. 129–148
[20] Jin, X., Krishnan, R., Sandhu, R.: 'A unified attribute-based access control model covering DAC, MAC and RBAC'. Proc. of the 26th Annual IFIP WG 11.3 Conf. on Data and Applications Security and Privacy XXVI, Paris, France, 2012, pp. 41–55
[21] Zaman, B.K., Krishnan, R., Sandhu, R.: 'Constraints specification in attribute based access control', *Science*, 2013, **2**, (3), pp. 131–145
[22] Jha, S., Sural, S., Atluri, V*., et al.*: 'Enforcing separation of duty in attribute based access control systems'. Proc. of the Int. Conf. Information Systems Security, Kolkata, India, 2015, pp. 61–78
[23] Zaman, B.K., Krishnan, R., Sandhu, R.: 'Towards an attribute based constraints specification language'. Proc. of the IEEE Int. Conf. Social Computing, Alexandria, USA, 2013, pp. 108–113
[24] Xu, Z., Stoller, S.D.: 'Mining attribute-based access control policies', *IEEE Trans. Dependable Secur. Comput.*, 2015, **12**, (5), pp. 533–545
[25] Medvet, E., Bartoli, A., Carminati, B*., et al.*: 'Evolutionary inference of attribute-based access control policies', Evol. Multi-Criterion Optim., Portugal, 2015, pp. 351–365
[26] Gautam, M., Jha, S., Sural, S*., et al.*: 'Poster: Constrained policy mining in attribute based access control'. Proc. of the 22nd ACM Symp. Access Control Models and Technologies, Indiana, USA, 2017, pp. 121–123
[27] Jha, S., Sural, S., Atluri, V*., et al.*: 'Specification and verification of separation of duty constraints in attribute based access control', *IEEE Trans. Inf. Forensics Sec.*, 2017, **4**, pp. 897–911