Policy Adaptation in Hierarchical Attribute-based Access Control Systems

SAPTARSHI DAS and SHAMIK SURAL, Indian Institute of Technology Kharagpur, India
JAIDEEP VAIDYA and VIJAYALAKSHMI ATLURI, Rutgers Business School, USA

In Attribute-Based Access Control (ABAC), access to resources is given based on the attributes of subjects, objects, and environment. There is an imminent need for the development of efficient algorithms that enable migration to ABAC. However, existing policy mining approaches do not consider possible adaptation to the policy of a similar organization. In this article, we address the problem of automatically determining an optimal assignment of attribute values to subjects for enabling the desired accesses to be granted while minimizing the number of ABAC rules used by each subject or other appropriate metrics. We show the problem to be NP-Complete and propose a heuristic solution.

CCS Concepts: • **Security and privacy** → **Access control**;

Additional Key Words and Phrases: ABAC policy, policy adaptation, attribute value hierarchy

ACM Reference format:
Saptarshi Das, Shamik Sural, Jaideep Vaidya, and Vijayalakshmi Atluri. 2019. Policy Adaptation in Hierarchical Attribute-based Access Control Systems. *ACM Trans. Internet Technol.* 19, 3, Article 40 (August 2019), 24 pages.
https://doi.org/10.1145/3323233

1 INTRODUCTION

Protection of organizational resources from unauthorized accesses is crucial for any organization. Over the years, various access control models have been developed for achieving this goal. Traditional access control models include Mandatory Access Control (MAC) [13], Discretionary Access Control (DAC) [8], and Role-Based Access Control (RBAC) [14]. In MAC, each entity is assigned a pre-defined security level according to which accesses are determined. MAC was predominantly used in government and defense organizations where confidentiality is of foremost importance. In contrast, DAC enables an object owner to grant or revoke access privileges to other subjects. The authorized set of accesses is typically maintained in the form of an Access Control Matrix (ACM) [15]. While flexible, DAC has significant administrative overhead. Hence, Role-Based

Research reported in this publication was supported by the National Institutes of Health under award R01GM118574, and by the National Science Foundation under awards CNS-1564034, CNS-1624503 and CNS-1747728. The content is solely the responsibility of the authors and does not necessarily represent the official views of the agencies funding the research.
Authors' addresses: S. Das and S. Sural, Dept. of Computer Science & Engineering, Indian Institute of Technology Kharagpur, West Bengal, 721302, India; emails: saptarshidas13@iitkgp.ac.in, shamik@cse.iitkgp.ac.in; J. Vaidya and V. Atluri, MSIS Department, Rutgers University, 1 Washington Park Newark, NJ 07102-1803 USA; emails: jsvaidya@business.rutgers.edu, atluri@rutgers.edu.

1533-5399/2019/08-ART40 $15.00
https://doi.org/10.1145/3323233

Access Control (RBAC) [14] is used as the de facto standard means for access in most commercial organizations. Instead of directly assigning permissions to users, the permissions are associated with specific *roles*. Users acquire their necessary permissions by being assigned membership to suitable roles. This reduces administrative overhead significantly, since users with similar access requirements are grouped into the same set of roles and the requisite permissions are included in those roles. Thus, RBAC can provide considerable flexibility in access control in scenarios that involve a known set of users. However, in situations that involve resource sharing among multiple organizations, the set of users cannot be determined *a priori*. RBAC no longer remains the access control model of choice in this context.

In recent years, Attribute-Based Access Control (ABAC) [6] has emerged as the appropriate access control model for enforcing controlled access in dynamic environments. However, it obligates organizations using traditional access control models to migrate to ABAC [16]. In ABAC, access is mediated based on a set of *attributes* and their associated values, where each attribute represents a particular characteristic of a subject, object, or the environment in which the access request is being made. ABAC also includes a set of rules collectively called the organizational ABAC *policy*. Thus, any organization migrating to ABAC would require an ABAC policy to be identified. The process of constructing a sound and complete ABAC policy is called *policy engineering* [16], which is considered to be one of the most critical aspects of implementing ABAC. Existing policy engineering approaches can be classified as either top-down or bottom-up. A top-down approach first identifies the functionally independent units of the organizational processes and then associates them with required accesses. Finally, rules are constructed from the associated accesses. In contrast, bottom-up approaches capitalize on the existing accesses available in the organization to construct the ABAC policy. As a third alternative, our view is that, instead of developing the policy *ab initio*, adapting to the policy of a similar organization would speed-up the process of migration.

In this work, it is assumed that the new organization has the same set of subject attributes, object attributes, environment attributes, as well as operations as that of the existing organization whose policy it is trying to adapt to. For instance, a new university can adapt to the existing ABAC policy of another university. The assumption mentioned above is realistic in such a scenario, since the typical attributes of different universities tend to be similar. Almost all universities use *professor*, *associate professor*, *assistant professor*, *student*, *teaching assistant*, *secretary*, and so on, as the possible values for the subject attribute *designation*. Similarly, *assignment*, *transcript*, and so on, are the possible values for the object attribute *type*; while *fall*, *spring*, *summer*, and so on, are the possible values for the environment attribute *semester*. Likewise, a hospital can adapt to the policy of another hospital that has similar attribute values, such as *doctor*, *nurse*, *attendant*, *ward*, *prescription*, *test report*, *day shift*, *night shift*, and so on. We also emphasize that cross-domain policy adaptation, like between a university and a hospital, is not envisaged here. The above-mentioned process of adapting to an existing policy, referred to as *policy adaptation* [3], can be achieved by appropriate assignment of attribute values to the users (interchangeably called "subjects" in this article).

While a suitable assignment pertains to an association of meaningful attribute values to different subject attributes, one also needs to take into account any hierarchy in the values of the subject attributes. Hierarchy, which is inevitably present in subject attributes, determines the flow of control and authority in any organization. For instance, a subject attribute *Designation* can have a value *Project Manager* that is senior to the attribute value *Team Member* in the hierarchy. The implication is that, a subject having a hierarchically senior value is permitted to access all the resources accessible to a subject having a junior value in the hierarchy.

Utilizing the hierarchy of subject attribute values ensures assignment of subject attribute-value pairs to subjects in a way that is close to real-life situations. Moreover, it helps in reducing

various commonly used measures of goodness of a policy set, such as the number of rules, Weighted Structural Complexity (WSC) [11], and so on. Therefore, the assignment of values to the subject attributes with respect to their hierarchies results in a more suitable and meaningful mapping. Once the policy adaptation process is successfully completed, the shared policy can be used by the target organization for regulating access to its resources.

In Reference [3], we introduced the problem of policy adaptation. The current article significantly extends and expands that work by including attribute hierarchy and environmental attributes. As mentioned above, use of hierarchy makes the process more close to the requirements of real-world organizations. Additionally, use of environmental attributes is being done for the first time in policy engineering, to the best of our knowledge. Environmental attributes form another unique feature of ABAC, making it subsume all other access control models including temporal [9], spatial [1], and spatio-temporal extensions of RBAC [12]. To summarize, in this article, we propose a migration strategy that enables an organization using a traditional access control model to adapt to the existing ABAC policy of a similar organization by an optimal assignment of subject attribute values in the presence of attribute hierarchy and environmental conditions. Here, optimality is primarily in the number of rules required for each subject to fulfill all its authorized accesses. This ensures efficient evaluation of rules during policy enforcement, thus enabling faster access decisions. In addition to the number of rules, we also analyze the quality of the generated subject attribute-value pairs using the Weighted Structural Complexity (WSC) [11] measure, which quantifies the size of the assigned subject attribute-value pairs.

2 PRELIMINARIES

In this section, we introduce the terms and notations used for representing the various components of an ABAC system that will be used throughout the article. Then, we introduce the concept of attribute hierarchy in ABAC.

2.1 Components of ABAC Model

ABAC consists of subjects, objects, environmental conditions, and access control rules. A subject is usually a human or an active non-person entity, such as an autonomous service or application. An object or resource is an entity that needs to be protected from unauthorized access. An environment defines the context in which an access request is made, e.g., *time of day*, *location of access*, and so on. Every subject is associated with several attributes, such as *designation* and *experience*, which either individually or in combination compose an expression to identify a group of subjects having similar access rights. Similarly, for each object, appropriate values are assigned to a set of object attributes. Typical examples of object attributes include *file type*, *sensitivity level*, *date of creation*, and so on. Rules collectively define the access control policy of the organization. A set of formal notations is given below that will be used for precise formulation of the problem being addressed and its solution.

- S: Set of authorized users in an organization. Each element of S is represented as s_i, for $1 \leq i \leq |S|$.
- O: Set of organizational resources or objects. Each element of O is represented as o_i, for $1 \leq i \leq |O|$.
- E: Set of environmental conditions in which access requests can be made. Each element of E is represented as e_i, for $1 \leq i \leq |E|$.
- S_a: Set of subject attributes that can affect the decision when an access request is made. Each element of S_a is represented as sa_i, for $1 \leq i \leq |S_a|$. Each sa_i can be assigned values

Table 1. Subject Attributes and Their Possible Values

Subject attribute	Possible values of subject attribute
$S.Designation$	$\{DIR, HOD, REG, PROF, STU, CLRK\}$
$S.Department$	$\{CSE, ECE, FIN\}$

Table 2. Object Attributes and Their Possible Values

Object attributes	Possible values of object attribute
$O.Type$	$\{tender, a/c\_daily, assignment, qn\_paper\}$
$O.Department$	$\{CSE, ECE, FIN\}$

from a set of possible attribute values SV_i. Similarly, O_a and E_a represent the sets of object attributes and environment attributes, respectively.

- $F_s: S \times S_a \rightarrow \{k|k \text{ is a subject attribute value}\}$. It is a function that associates a value for any subject attribute to a subject. The functions F_o and F_e are similarly defined for object and environment, respectively. Essentially, assignment of attribute values to attributes of all the entities is performed by these functions.
- OP: Set of allowed operations in the system. Each element of OP is represented as op_i, for $1 \leq i \leq |OP|$.
- R: Set of rules that forms an ABAC policy. Each element of R is a rule represented as r_i, for $1 \leq i \leq |R|$.
- S_v: A set containing assignment of values to all the subject attributes, also referred to as subject attribute-value pairs, for all the subjects. Similarly, O_v and E_v represent the sets containing assignment of values to attributes for all objects and environments, respectively.

Each rule $r \in R$ is a 4-tuple of the form $< SC, OC, EC, op >$, where SC, OC, and EC represent a set of subject attribute-value pairs, a set of object attribute-value pairs, and a set of environment attribute-value pairs, respectively. $r[SC]$ represents the subject attribute-value pairs associated with rule r. $r[SC]$ is alternatively called the subject conditions associated with rule r. $r[OC]$, $r[EC]$, and $r[op]$ are also defined similarly. op represents any particular operation. Each attribute-value pair $av \in \{SC \cup OC \cup EC\}$ is an equality of the form $a = c$ or $a = not\ c$, where a is the name of an attribute and c is the value associated with a and $not\ c$ represents the set of all the possible values of a, except c. c is either a constant or a *don't care* represented as "–". An access or a permission is represented by a 4-tuple. For example, an access $< s, o, e, op >$ means that a subject s can perform an operation op on an object o under an environment condition e. The inclusion of environment attributes in ABAC gives it a dynamic nature, i.e., the permissions given to a subject can change with various environmental conditions.

Example 1. Consider a university ABC that has three departments: Computer Science and Engineering (CSE), Electronics and Communication Engineering (ECE), and Finance (FIN). Each subject can have one of the following designations: Director (DIR), Head of the Department (HOD), Registrar (REG), Professor (PROF), Finance Officer (FINO), Student (STU), and Clerk (CLRK). Each object can be of one of the following types: tender, daily account (a/c_daily), assignment, and question paper (qn_paper). The location of an access request can be either office or residence, and the time of an access request can be a weekday or a weekend. Let *Alice* and *Bob* be two subjects of the university. Each subject has two attributes, namely, *S.Designation* and *S.Department*. Tables 1, 2, and 3 respectively represent the set of subject, object, and environment attributes along with their possible values. The attributes and their corresponding values for the subjects are given in Table 4. Table 5 contains the ABAC rules of the university. Table 6 shows the values assigned to object attributes for each object.

Table 3. Environment Attributes and Their Possible Values

Environment attribute	Possible values of environment attribute
$E.Location$	{*office*, *residence*}
$E.Time$	{*weekday*, *weekend*}

Table 4. Subject to Subject Attribute Value Assignment

	S.Designation	S.Department
Bob	*STU*	*CSE*
Alice	*PROF*	*CSE*

Table 5. ABAC Policy to be Re-used Where Office, Residence, Weekday, and Weekend are Represented by off., res., wd, and wn, Respectively

	Subject part		Object part		Environment part	
	S.Designation	S.Department	O.Type	O.Department	E.Location	E.Time
r_1	DIR	-	-	-	-	-
r_2	HOD	CSE	tender	CSE	off.	wd
r_3	HOD	CSE	*not* tender	-	-	wd
r_4	HOD	ECE	*not* tender	-	-	wd
r_5	HOD	ECE	tender	ECE	off.	wd
r_6	REG	-	*not* tender	*not* FIN	-	-
r_7	PROF	-	a/c_daily	-	off.	wd
r_8	PROF	-	assignment	-	-	-
r_9	PROF	CSE	qn_paper	CSE	-	-
r_{10}	PROF	ECE	qn_paper	ECE	-	-
r_{11}	FINO	-	*not* tender	FIN	off.	-
r_{12}	STU	CSE	assignment	CSE	res.	wn
r_{13}	STU	ECE	assignment	ECE	res.	wn
r_{14}	CLRK	-	a/c_daily	-	off.	wd
r_{15}	REG	-	tender	*not* FIN	off.	wd
r_{16}	REG	-	*not* tender	FIN	off.	wd

From Tables 1–6, it can be seen that *Bob* is permitted to access o_6 using rule r_{12}, *Alice* gets the permission to access objects o_5, o_6, o_7, and o_8 using rules r_8, r_9, and r_{14}.

2.2 Attribute Value Hierarchy in ABAC

The concept of hierarchy is inherently present in every organization expressing the flow of authority and control. Hierarchies can be incorporated in ABAC by arranging the possible values of a subject attribute with respect to their hierarchy. Figure 1 represents an instance of hierarchy of attribute values for the subject attribute *S.Designation*. The attribute values are organized in the form of a Directed Acyclic Graph (DAG) where the attribute values at the top of the graph are more influential than those present at the lower levels.

It is natural that the relationship is directed, since if we say A and B are hierarchically related, one would need to know if A is senior to B or the other way around. Further, it is acyclic, since a cycle would imply (either directly or transitively) A is senior to B and at the same time B is senior to A, which is not a common occurrence in any organization.

For example, a directed arrow ($\rightarrow$) between the attribute values *Director* and *HOD* signifies that all the accesses permitted to the *HOD* are also permitted to the *Director*. The *Director* can have additional accesses beyond these. For instance, let there be three subjects named Alice, Bob, and Cathy. The objects that need to be accessed by these subjects are given below:

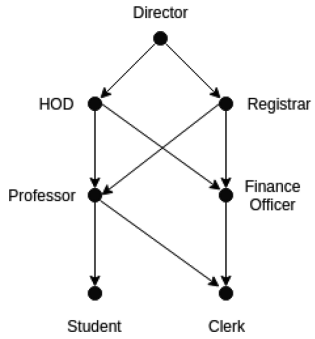

Fig. 1. Example hierarchy for subject attribute *S.Designation*.

$Alice - \{doc_1, doc_2, doc_3, doc_4\}$
$Bob - \{doc_1, doc_2, doc_5\}$
$Cathy - \{doc_2, doc_5\}$
In such a scenario, it can be said that *Bob* is more influential (higher in the hierarchy) than $Cathy$, since *Bob* can access all the objects accessible to $Cathy$ along with additional objects. However, although *Alice* is permitted to access a higher number of objects as compared to *Bob*, she is not considered to be more influential, since there is an object (doc_5) that can be accessed by *Bob* but not *Alice*.

Definition 2.1 (Attribute Value Hierarchy). The attribute value hierarchy for a particular subject attribute sa_i having the possible set of values SV_i is denoted by AVH, where $AVH \subseteq SV_i \times SV_i$ is a partial order on SV_i called the attribute value hierarchy, denoted as $\geq$.

3 POLICY ADAPTATION PROBLEM WITH ATTRIBUTE VALUE HIERARCHY (POLAP-AVH)

As discussed in Section 1, an organization aspiring to migrate to ABAC requires an ABAC policy, and the ability to adapt to an existing ABAC policy will result in a faster migration process. We particularly consider situations in which a target organization indeed finds the ABAC policy of a similar organization suitable for its own use. Here, similarity implies that the two organizations can make use of a common set of attributes and have similar access rules. Since the characteristics of the organizational resources are relatively more immutable compared to the properties of subjects, we consider the object attributes and their associated values as inputs. Thus, the subject attributes and their corresponding values can be decided algorithmically. From this perspective, one can define the policy adaptation problem using attribute value hierarchy (PolAP-AVH) as the problem of suitable assignment of values to the subject attributes with respect to an ABAC policy and a list of authorized accesses. It needs to be emphasized here that the target organization does not use the attribute-value assignments of the subjects or objects of the source organization. It initially assigns attribute-value pairs to its own objects and environment and then uses the proposed approach to determine the subject-to-subject attribute-value pair assignments, so using the policy being adapted to, all the requisite accesses in its ACL may be granted. This is motivated by the fact that any object in an organization is inherently associated with several attributes. For instance, a student record in a university is associated with attributes such as *department*, *course*, and so on, each having a set of possible values it can take, such as *CSE*, *ECE* (for the attribute *department*), and *DBMS*, *Algorithms* (for the attribute *course*). The organization system administrators/security officers would be aware of this information. Similarly, the environment attributes, such as *location*,

Table 6. Objects with Their Associated Attributes and Their Values

	O.Type	O.Department
o_1	tender	FIN
o_2	tender	CSE
o_3	tender	ECE
o_4	a/c_record	FIN
o_5	a/c_daily	FIN
o_6	assignment	CSE
o_7	assignment	ECE
o_8	qn_paper	CSE
o_9	qn_paper	ECE

Table 7. Objects Accessible Through Each Rule

Rule	Objects
r_1	o_1, o_2, o_3, o_4, o_5, o_6, o_7, o_8, o_9
r_2	o_2
r_3	o_4, o_5, o_6, o_7, o_8, o_9
r_4	o_4, o_5, o_6, o_7, o_8, o_9
r_5	o_3
r_6	o_6, o_7, o_8, o_9
r_7	o_5
r_8	o_6, o_7
r_9	o_8
r_{10}	o_9
r_{11}	o_4, o_5
r_{12}	o_6
r_{13}	o_7
r_{14}	o_5
r_{15}	o_2, o_3
r_{16}	o_4, o_5

Table 8. Rules Using Which Each Object can be Accessed

Object	Rules
o_1	r_1
o_2	r_1, r_2, r_{15}
o_3	r_1, r_5, r_{15}
o_4	r_1, r_3, r_4, r_{11}, r_{16}
o_5	r_1, r_3, r_4, r_7, r_{11}, r_{14}, r_{16}
o_6	r_1, r_3, r_4, r_6, r_8, r_{12}
o_7	r_1, r_3, r_4, r_6, r_8, r_{13}
o_8	r_1, r_3, r_4, r_6, r_9
o_9	r_1, r_3, r_4, r_6, r_{10}

Table 9. Desired Accesses Along with Environment Conditions Where Office, Residence, Weekday, and Weekend are Represented by off., res., wd, and wn, Respectively

	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8	o_9
s_1	(-,-)	(-,-)	(-,-)	(-,-)	(-,-)	(-,-)	(-,-)	(-,-)	(-,-)
s_2	0	(off.,wd)	0	(-,wd)	(-,wd)	(-,wd)	(-,wd)	(-,wd)	(-,wd)
s_3	0	0	(-,wd)	(-,wd)	(-,wd)	(-,wd)	(-,wd)	(-,wd)	(-,wd)
s_4	0	(-,-)	(-,-)	(-,-)	(-,-)	(-,-)	(-,-)	(-,-)	(-,-)
s_5	0	0	0	0	(off.,wd)	(-,-)	(-,-)	(-,-)	0
s_6	0	0	0	0	(off.,wd)	(-,-)	(-,-)	0	(-,-)
s_7	0	0	0	0	0	(res.,wn)	0	0	0
s_8	0	0	0	0	0	0	(res.,wn)	0	0
s_9	0	0	0	(off.,-)	(off.,-)	0	0	0	0
s_{10}	0	0	0	0	(off.,wd)	0	0	0	0

day of week, and so on, and their possible values, such as *IP address* (for the attribute *location*) and *Monday*, *Tuesday*, and so on, (for the attribute *day of week*) that affect access decisions would usually be available with them.

While any valid assignment of subject attribute-value pairs would suffice for this primary requirement, we introduce an optimization requirement to PolAP-AVH that necessitates the attribute value assignments to be such that the number of rules used by each subject to fulfill all its required accesses is minimum. Further, we introduce the concept of hierarchical attribute values for the subject attributes. We extract the hierarchical information from the given ABAC policy and show that using this hierarchical information generates more meaningful assignments of attribute values to subjects and further reduces the number of rules used by a subject. For notational simplicity, we refer to the optimization problem as PolAP-AVH throughout the article. The formal definition of PolAP-AVH is given below.

Definition 3.1 (PolAP-AVH). Given an ABAC policy P, a set S of subjects, set O of objects, set O_a of object attributes, set O_v of attribute-value assignments of all objects, set E of environment conditions, set E_a of environment attributes, set E_v of attribute-value assignments of all environment conditions, set OP of possible operations, and a desired set of accesses, in the form of an access control list (ACL), L, determine all the subject attribute hierarchies and using it assign subject attribute-value pairs to all subjects such that all the desired accesses are satisfied and each subject uses the minimum number of rules to gain all the accesses in L.

Solving PolAP-AVH enables an organization to adapt to an existing ABAC policy by appropriate assignment of subject attribute-value pairs to its users. A solution to PolAP-AVH must not allow any additional accesses except the ones contained in L, i.e., there should not be any security violation.

3.1 Complexity Analysis

In this sub-section, we do a formal analysis of the complexity of PolAP-AVH. We show that the problem is NP-Complete. To develop the proof, we first formulate a decision version of PolAP-AVH.

Definition 3.2 (Decision Version of PolAP-AVH (D-PolAP-AVH)). Given R, L, S, O, O_a, O_v, E_a, E_v, OP as mentioned in Definition 3.1, and an integer k, does there exist a possible assignment of subject attribute-value pairs such that the required accesses in L are satisfied and each $s \in S$ uses a set of rules $R^{'}$ to obtain all its desired accesses and $|R^{'}| \leq k$?

For proving NP-completeness of D-PolAP-AVH, we make use of a known NP-Complete problem, namely, the Minimum Hitting Set (MHS) problem, which is defined below.

Definition 3.3 (Minimum Hitting Set (MHS) Problem). Given a universal set U and a collection ST of subsets of U where each $st_i \subseteq U$ and $\bigcup_{i=1}^{|ST|} st_i = U$, find the smallest subset $H \subseteq U$ such that every set in ST is hit by H, i.e., $\forall st_i \in ST : st_i \cap H \neq \phi$.

Definition 3.4 (Decision Version of Minimum Hitting Set (D-MHS) Problem). Given a collection ST of subsets of a universal set U and an integer t, does there exist a hitting set H such that $|H| \leq t$.

Theorem 1. *D-PolAP-AVH is NP-Complete.*

Proof. Given a certificate consisting of a set of rules R, a collection RS containing subsets of rules in R, an assignment of subject attribute-value pairs such that a subject requires a set of rules $R^{'}$ to fulfill all its desired accesses and an integer k, it can be verified in polynomial time whether $|R^{'}| \leq k$ by counting the number of rules in $R^{'}$. It can also be verified in polynomial time that R and RS are consistent with $R^{'}$ by checking that each set of rules in RS contains at least one rule from $R^{'}$. Thus, D-PolAP-AVH is in NP.

Now, we prove that D-MHS $\leq_p$ D-PolAP-AVH. Let, $< U, S, k >$ be an instance of D-MHS, where U is the universal set containing n elements, i.e., $U = \{u_1, u_2, \ldots, u_n\}$. Without loss of generality, we can assume each u_i to be an integer (i.e., each $u_i \in \mathbb{Z}$), since U is either finite or countably infinite. S consists of m subsets $s_1, s_2, \ldots, s_m$, where each $s_i \subseteq U$ and $\bigcup_{i=1}^{m} s_i = U$, and k is an integer.

Let $\mathcal{F} : \mathbb{Z} \rightarrow \mathbb{Z}^4$ be a function such that $\mathcal{F}(u_i) = (u_i, u_i + 1, u_i + 2, u_i + 3)$. Let us denote $(u_i, u_i + 1, u_i + 2, u_i + 3)$ by r_i. Therefore, $\mathcal{F}(u_i) = r_i$ and a rule of D-PolAP-AVH r_i can be constructed from an element $u_i \in U$, where u_i, $u_i + 1$, $u_i + 2$ and $u_i + 3$ respectively represent a subject attribute value, an object attribute value, an environment attribute value, and an operation. Thus, using $\mathcal{F}$, we can construct an ABAC rule from a given element in constant time. $\mathcal{F}$ takes each $u_i \in U$ and returns a rule r_i. Therefore, from a set of elements U, we can obtain a set of rules R in

$O(n)$ time, where $|U| = n$. Similarly, from S containing m subsets of U, a collection of subsets of D-PolAP-AVH rules RS can be constructed using the function $\mathcal{F}$.

Thus, from the given instance of D-MHS, an instance of D-PolAP-AVH $< R, RS, k^{'} >$ can be constructed using the function $\mathcal{F}$, where R is a set of n authorization rules a user *Alice* needs to fulfill all her required accesses, i.e., $R = \{r_1, r_2, \ldots, r_n\}$. RS consists of m subsets $rs_1, rs_2, \ldots, rs_m$, where each $rs_i \subseteq R$ such that each rs_i corresponds to a set of rules that *Alice* can use to fulfill a particular access, and finally setting $k^{'} = k$.

Let $R^{'}$ be a solution to the constructed instance of D-PolAP-AVH such that $|R^{'}| \leq k$ and every $rs_i \cap R^{'} \neq \phi$. Therefore, $< R, RS, k >$ returns $TRUE$ and $R^{'} \subseteq R$. Let H be a solution to the instance of D-MHS. To complete the proof, we show that the solution to the instance of D-MHS (consisting of H) returns $TRUE$, i.e., $|H| \leq k$ if and only if the solution to the instance of D-PolAP-AVH (consisting of $R^{'}$) returns $TRUE$, i.e., $|R^{'}| \leq k$.

Now, a solution to the instance of D-MHS can be obtained from the given instance of D-PolAP-AVH as follows: Since for each $u_i \in U$, there is a corresponding $r_i \in R$, there is always an element in U for a rule in R. Similarly, for each subset rs_i of rules, we can obtain a subset of elements $s_i \in S$. H can be constructed by selecting the elements in U corresponding to the rules in $R^{'}$. Now, if $R^{'}$ contains at least one element from each of the subset of rules in RS, then H will also contain at least one element from each of the subset of elements in S, i.e., for each $s_i \in S$, $s_i \cap H \neq \phi$ and $|H| = k$. Therefore, H is a valid solution to the instance of D-MHS and $< U, S, k >$ returns $TRUE$.

4 HEURISTIC APPROACH FOR SOLVING POLAP-AVH

In Section 3, it has been established that the problem of determining a minimal set of rules for each subject is NP-Complete. Therefore, it is not likely that PolAP-AVH can be solved deterministically in polynomial time. This necessitates employing a heuristic approach to solve PolAP-AVH, which we present in this section. A greedy heuristic utilized for solving the minimum hitting set problem [4] is suitably used in our proposed solution. The overall algorithm for solving PolAP-AVH is given in Algorithm 1. For each subject, it computes an appropriate assignment of values to all the attributes and a set of rules by which the subject can acquire all the permitted accesses in L when a set of objects O, a set of subjects S, a set O_v containing attribute-value pairs corresponding to each object, a list of desired accesses L, and a policy R are given as input.

Algorithm 1 involves three phases, the first consisting of four steps, the second having a single step, followed by a third phase involving two steps. In the following sub-sections, we explain each phase in detail.

4.1 Rule Set Generation

For each subject, this phase constructs a collection of sets that compose rules where each set of rule corresponds to a specific access allowed to the subject. The rule set generation phase involves four steps as enumerated below:

Step 1.1 Generate a set of rules corresponding to each object
This is the first step of phase 1 of Algorithm 1 (Line 1). For each object, it finds the set of rules that can be used to access that object. The detailed procedure for this step is given in Algorithm 2. It takes a set of objects O, a set O_v containing attribute-value pairs corresponding to each object, and a set of rules R as inputs and generates a set RO that, for each object, contains a set of rules through which that object can be accessed. Essentially, for each object, Algorithm 2 iterates over all the rules in R. All the object attribute-value pairs of a rule r ($r[OC]$) also being associated with

ALGORITHM 1: Policy adaptation

```
1: procedure PoLAP-AVH(S, O, O_v, L, R)
2:     RO ← GENERATE_RFO(O, O_v, R)
3:     OR ← GENERATE_OFR(O, O_v, R)
4:     OS ← GENERATE_AFS(S, L)
5:     AH ← GENERATE_HIERARCHY(R, SA, OR)
6:     SR ← [ ]
7:     S_v ← [ ]
8:     for i ← 1 to |S| do                                   ▷ iterating over all subjects
9:         RS ← generate_SORO(i, O_v, RO, OR, OS)            ▷ generates sets of rules for a subject
10:        if RS == Null then
11:            exit
12:        H ← GENERATE_RFS(RS)                              ▷ generate minimal set of rules for a subject
13:        insert(SR, H)
14:        for j ← 1 to |H| do
15:            S_v[i] ← S_v[i] ∪ {H[j][i]}                   ▷ assign attribute values to a subject
16:    for i ← 1 to |S_v| do
17:        for j ← 1 to |S_v[i]| do
18:            if |S_v[i][j]| ≥ 2 then                       ▷ checking for multi-valued attributes
19:                for k ← 1 to |S_v[i][j]| do
20:                    temp ← max(AH[j])
21:                    if AH[j][k]! = temp then              ▷ comparing the hierarchy values
22:                        remove S_v[i][j][k] from S_v[i][j]   ▷ removing attribute value having a lower
   hierarchy value
23:    return S_v                  ▷ return a set containing attribute value assignments for all subjects
```

ALGORITHM 2: Generate rules for each object

```
1: procedure GENERATE_RFO(O, O_v, R)
2:     RO ← [ ]
3:     for i ← 1 to |O| do                                   ▷ iterating over all objects
4:         temp ← [ ]
5:         for j ← 1 to |R| do                               ▷ iterating over all rules
6:             if R[j][OC] ⊆ O_v[i] then                     ▷ checking whether an object is accessible by a rule
7:                 temp ← temp ∪ {R[j]}
8:             insert(RO, temp)
9:     return RO
```

an object o (Line 6) signifies that the object o can be accessed using rule r. The worst-case time complexity of this algorithm is $O(|O||R||O_v|)$.

Step 1.2 Generate a set of objects corresponding to each rule
The second step (Line 3, Algorithm 1) of this phase is the dual of the first step, i.e., it iterates over all the rules and, for each rule, it generates the set of objects that can be accessed using that rule. The detailed steps are given in Algorithm 3. The inputs to this step are similar to Step 1.1. For each rule $r \in R$, the object attribute-value pairs associated with r are compared with the set of object attribute-value pairs associated with each object. If the object attribute-value pairs associated with a rule $r \in R$ are also associated with o (Line 6), it indicates that object o can be accessed through r. So, o is added to the set of objects for r (Line 7). Essentially, for each rule, a set of objects is

ALGORITHM 3: Generate objects for each rule

```
1: procedure GENERATE_OFR(O, O_v, R)
2:     OR ← [ ]
3:     for i ← 1 to |R| do                                  ▷ iterating over all the rules
4:         temp ← [ ]
5:         for j ← 1 to |O_v| do          ▷ iterating over attribute value assignments for all objects
6:             if R[i][OC] ⊆ O_v[j] then   ▷ checking whether a rule can be used to access an object
7:                 temp ← temp ∪ {O[j]}
8:             insert(OR, temp)
9:     return OR
```

ALGORITHM 4: Generate accesses for each subject

```
1: procedure GENERATE_AFS(S, L)
2:     OS ← [ ]
3:     for i ← 1 to |S| do                                  ▷ iterating over all the subjects
4:         temp ← [ ]
5:         for j ← 1 to |L| do                    ▷ iterating over all the desired accesses
6:             if S[i] ∈ L[j] then   ▷ checking whether a subject is present in an access
7:                 insert(temp, (L[j][1], L[j][2]))
8:         insert(OS, temp)
9:     return OS
```

generated such that each object in the set can be accessed through that rule. The worst-case time complexity of this algorithm is $O(|R||O||O_v|)$.

Step 1.3. Generate the set of accesses for each subject
The third step (Line 4, Algorithm 1) determines the object-operation pairs permitted for each subject. The algorithm for this step is given in Algorithm 4, which takes a set of subjects S and a list L of desired accesses as inputs and generates a set OS containing sets of object-operation pairs entitled to each subject. For each subject s, the algorithm iterates over all the accesses in L (Line 5). If s is present in an access (Line 6), the object and operation associated with that access are added to the set of object-operation pairs for s. For instance, if an access (s, o, op) is encountered, (o, op) is added as an object-operation pair allowed to s. The worst-case time complexity of Algorithm 4 is $O(|S||L|)$

Step 1.4. Generate collection of rule sets for each subject
The fourth step (Lines 6–11, Algorithm 1) generates a collection of sets of rules, where each set consists of rules that can be used to access a specific object-operation pair allowed to a subject. Therefore, for a subject, the collection of sets contains a set of rules corresponding to each object-operation pair permitted to the subject. The collection of rule sets for subjects is generated by constructing a Subject-Object-Rule-Object (SORO) tree for each subject. The detailed steps of the algorithm for building a SORO tree is given in Algorithm 5, whose worst-case time complexity is $O(|L| + |L||R| + |S||R|)$.

Definition 4.1 (Subject-Object-Rule-Object (SORO) Tree). A SORO tree is an n-ary tree having 4 levels. The root (first level) of a SORO tree is a subject. In the second level, the object-operation pairs permitted to that subject are added as child nodes. The third level is constructed using the rules through which the objects in the second level can be accessed. The fourth level is constructed using the objects accessible through the rules in the third level.

ALGORITHM 5: Generate rule-sets for each subject

```
1:  procedure GENERATE_SORO(i, O_v, RO, OR, OS)
2:      ST ← OS[i]                          ▷ obtain all the object-operation pairs entitled to ith subject
3:      ST' ← [ ]
4:      for each (o_a, op_b) in ST do       ▷ iterating over all object-operation pairs in ST
5:          ST' ← ST' ∪ {o_a}
6:      RS ← [ ]                            ▷ RS will finally contain the sets of rules for the ith subject
7:      for each (o_c, op_d) in ST do       ▷ iterate over all object-operation pairs in ST
8:          T ← RO[c]                       ▷ obtain all the rules corresponding to an object in an object-operation pair
9:          for each r_e in T do            ▷ iterate over all rules obtained
10:             if r_e[op] ≠ op_d then      ▷ check whether the operations in the object-operation pair and the
    rule match
11:                 RO[c] ← RO[c] − {r_e}   ▷ discard the rule
12:         insert(RS, RO[c])
13:     RS' ← RS
14:     for j ← 1 to |RS'| do               ▷ iterating over all rule sets
15:         for each r_f in RS'[j] do       ▷ iterating over rules in each rule set
16:             UT ← OR[f]                  ▷ obtain the objects corresponding to each rule
17:             if UT − ST' ≠ Null then     ▷ checking for unauthorized accesses
18:                 RS[j] ← RS[j] − {r_f}   ▷ pruning a rule allowing unauthorized access
19:                 if RS[j] = Null then
20:                     exit
21:     return RS
```

The SORO tree for a given subject s is constructed by first initializing the tree with s as the root. All the object-operation pairs permitted to s are added as child nodes of s. These object-operation pairs are obtained from the sets of object-operation pairs generated by Algorithm 4. This completes the second level of the SORO tree. For each object-operation pair in the second level, the rules through which they can be accessed are added as child nodes of the object-operation pair to complete the third level of the SORO tree. The rules that allow access to a specific object-operation pair are obtained from the sets of rules generated by Algorithm 2. Finally, for each rule in the third level, objects that can be accessed using that rule are added as child nodes of the rule. The objects accessible through a given rule are obtained from the sets of objects generated by Algorithm 3. Construction of the SORO tree is followed by a comparison of the objects in the leaf nodes of the SORO tree with the objects present in the second level. If any extra object is found in the leaf nodes, the parent of that extraneous object in the third level is removed from the SORO tree along with all its children. An extraneous object in the leaf node represents an unauthorized access. Note that, in this work, we assume that it is indeed possible to find attribute values for each subject (and, correspondingly, object and environment) such that by using the rules in the policy to be adapted to, all required accesses are satisfied while no extraneous accesses are enabled. Assuming that one or more such assignments exists, our proposed approach will find an assignment that minimizes the number of rules used by each subject to get its desired access. Removal of rules allowing unauthorized accesses is termed as *pruning*. After pruning, the rules in the third level of the SORO tree are grouped into sets with respect to the object-operation pairs present in the second level. This marks the end of the first phase of Algorithm 1. At this stage, for each subject, we have a collection of sets of rules.

ALGORITHM 6: Generate attribute value hierarchy

```
1:  procedure GENERATE_HIERARCHY(R, SA, OR)
2:      AH ← [ ]
3:      for each subject attribute sa_i ∈ SA do                ▷ iterating over all subject attributes
4:          SV_i ← all values of attribute sa_i
5:          OSV ← [ ]
6:          for each value sv_ij ∈ SV_i do                     ▷ iterating over all the values of a particular attribute
7:              objects ← [ ]
8:              rules ← all rules where sv_ij is present
9:              temp ← objects accessible using all the elements of rules from OR
10:             objects ← objects ∪ {temp}
11:         insert(OSV, objects)
12:         initialize matrix G of size |SV_i| × |SV_i| with each G[i][j] := 0   ▷ initializing the adjacency
    weight matrix
13:         for each G[i][j] ∈ G do
14:             if i == j||OSV[i] == OSV[j] then
15:                 G[i][j] := ∞                               ▷ ensures no self-loop in the hierarchy graph
16:             if OSV[j] ⊂ OSV[i] then
17:                 G[i][j] := −1                              ▷ assignment of weight to an edge
18:         G' ← [ ]
19:         for each node n ∈ G do
20:             G' ← G' ∪ SELECT_EDGES(G, n)
21:             G' ← construct a graph from all the selected edges
22:             for each node n ∈ G' do
23:                 temp ← [ ]
24:                 hierarchy_n ← height(G') − level(n) + 1    ▷ computing the hierarchy value
25:                 insert(temp, hierarchy_n)
26:         insert(AH, temp)
27:     return AH
```

4.2 Attribute Value Hierarchy Generation for Subject Attributes

The steps for generating attribute value hierarchy are described in this sub-section.

Step 2.1. Generation of Attribute Value Hierarchy for Each Subject Attribute
The second phase is composed of a single step (Line 6, Algorithm 1). This phase constructs the hierarchy of the attribute values for each subject attribute. The procedure for generating the attribute value hierarchy is given in Algorithm 6. This step takes a set of rules R, a set of subject attributes SA, and a set OR containing sets of objects that can be accessed using each rule (generated using Algorithm 3) as input. At first, for each attribute sa, this step finds all the possible values that the attribute can have (Line 3). Next, the rules are grouped according to the values of sa, which is followed by determining the objects accessible by each of the group of rules (Lines 4–10). At this stage, we have a list of objects corresponding to each value of attribute sa. Using this, an adjacency matrix G is constructed that consists of rows and columns corresponding to each of the possible values of sa. Each element $G[i][j]$ is assigned values as follows:

$G[m][n]$ is set to ∞ when both the row and the column correspond to the same attribute value or both sv_{im} and sv_{in} cover the same set of subjects. When sv_{im} covers more objects in addition to the ones covered by sv_{in}, $G[m][n]$ is set to -1. Using G, a graph is constructed. For computing the hierarchy, it is required to remove certain edges from G to obtain a DAG. For example, if there are three edges $a \rightarrow b$, $b \rightarrow c$ and $a \rightarrow c$, we need to remove the edge $a \rightarrow c$. Essentially, this is

ALGORITHM 7: Select edges

```
1:  procedure SELECT_EDGES(G, s)
2:      topological_sort(G)                          ▷ topologically sort the nodes of the graph
3:      for each node v ∈ V of G do
4:          d[v] := ∞
5:          π[v] := nil
6:      d[s] := 0
7:      S := Φ
8:      for each node u, taken in topologically sorted order do
9:          for each node v ∈ Adj[u] do
10:             if d[v] > d[u] + w(u, v) then        ▷ checking if a shorter path exists
11:                 d[v] := d[u] + w(u, v)
12:                 π[v] := u
13:         S := S ∪ (u, v)
14:     return S
```

the problem of finding the longest path between a pair of vertices in a DAG. The procedure for finding the paths is given in Algorithm 7, which basically selects the edges that are to be kept in the hierarchy graph. Finally, the attribute values in the leaf nodes of the graph are assigned the value of 1 and the value is incremented by 1 for each of the higher levels (Line 24). Thus, the values that are higher in the hierarchy tree are assigned higher values. The worst-case time complexity of Algorithm 7 is $O(|S_v|^2)$, where $|S_v|$ is the maximum number of values an attribute can take.

4.3 Generation of Minimal Rule Set and Subject Attribute-value Pairs

The third phase works in two steps. From the collection of sets of rules obtained at the end of the first phase, for each subject, this phase initially creates a minimal set of rules; using which, the subject can satisfy its required accesses and then assigns attribute-value pairs to the subjects.

Step 3.1. Determine Minimal Rule Set for Subjects
The first step of the third phase (Line 12, Algorithm 1) computes a minimal set of rules required by each subject to satisfy all the desired accesses. It takes the rule sets generated by the previous step and applies the *GENERATE_RFS* procedure given in Algorithm 8. The minimal set of rules is computed using a greedy heuristic, where in each step the rule that is a member of the maximum number of rule sets is selected. The greedy heuristic used in this approach is a $log(n)$-approximate solution where n is the number of rules. In situations where there are more than one minimum hitting sets, any one is randomly selected. This procedure is similar to finding a minimum hitting set from a collection of subsets of a universe. Here, in each step, the element occurring in the maximum number of un-hit sets is selected.

Step 3.2. Subject Attribute-value Pairs Generation
The final step (Lines 14–23) of Algorithm 1 takes the set of rules generated in Step 3.1 as input. For each subject, the subject attribute-value pairs associated with each rule in the minimal set of rules for the subject are assigned to that subject. In case of multiple attribute values assigned to a subject attribute, the attribute value with higher hierarchy is selected. At the end of the second phase of Algorithm 1, for each subject, we obtain a set of rules by which all the desired accesses can be satisfied. Also, we get the subject attribute-value pairs assigned to all the subjects. Finally, for the subject attributes that have multiple values assigned (Line 18), the values that are hierarchically senior are selected, and the rules containing hierarchically junior values are removed from the set of rules (Line 22).

ALGORITHM 8: Generate minimal set of rules for subject

```
1: procedure GENERATE_RFS(RS)
2:     U ← [ ]
3:     for i ← 1 to |RS| do
4:         U ← U ∪ {RS[i]}
5:     RS' ← [ ]
6:     while RS ≠ Null do
7:         select u_i from U s.t. u_i hits maximum number of sets from RS
8:         remove already hit sets from RS
9:         RS' ← RS' ∪ {u_i}
10:    return RS'
```

4.4 Illustrative Example

We now elaborate on the workings of our proposed approach using an illustrative example involving a realistic scenario, where a new university wants to adapt to the ABAC policy of an existing university. A similar situation arises while setting up a new branch of an existing organization. Let us consider the university ABC mentioned in Section 2. Let subject s_5 be an employee of ABC. The policy to be adapted to is given in Table 9. The objects along with their attributes and corresponding values are listed in Table 6. From the desired list of accesses permitted to each of the subjects in Table 9, we can see that the subject s_5 is permitted to access the objects o_5, o_6, o_7, and o_8 under certain environment conditions. Although we consider the operation of the proposed solution on all the entities, we show the subject attribute-value pair assignment to only one subject (s_5) for the sake of brevity.

In Step 1.1 of our proposed solution to PolAP-AVH, for each object, we determine a set of rules by which the object can be accessed. Algorithm 2 generates a set of rules for each object by comparing the object attribute-value pairs associated with the object and the rules. The objects and the rules by which they can be accessed is given in Table 8. We can observe that the objects o_5, o_6, o_7, and o_8 can be accessed using the sets of rules $\{r_1, r_3, r_4, r_7, r_{11}, r_{14}, r_{16}\}$, $\{r_1, r_3, r_4, r_6, r_8, r_{12}\}$, $\{r_1, r_3, r_4, r_6, r_8, r_{13}\}$, and $\{r_1, r_3, r_4, r_6, r_9\}$, respectively. In this example, we have considered only one operation, i.e., *access*.

Now, Step 1.2 determines the set of objects accessible using each rule. This is achieved using Algorithm 3 by comparing the object conditions associated with each rule to the object attribute-value pairs associated with each object. The rules and the corresponding objects accessible through them are given in Table 7. Thus, we obtain the objects that can be used by each rule obtained in Step 1.1.

In Step 1.3, we find object-operation pairs allowed for each subject. The permitted accesses are listed in Table 9. Our chosen subject, i.e., s_5, is permitted to access objects o_5, o_6, o_7, and o_8, as discussed previously.

Step 1.4 finds a set of rules corresponding to each object that s_5 is permitted to access. Here, we construct a SORO tree for s_5 using Algorithm 5. The SORO tree for s_5 is given in Figure 2. The root of the SORO tree is s_5. In the second level, the objects accessible by s_5 are represented as children of s_5. The third level consists of the rules through which each of the objects can be accessed. The fourth level consists of objects accessible through each rule in the third level. If any rule allows any extraneous access, that rule is removed through pruning. For example, rule r_3 allows s_5 to access the object o_4, which is an extraneous permission, as it is not present in Table 9. So, rule r_3 is not present in the final set of rules that provides s_5 access to o_5. The final set of rules after pruning is given under the fourth level of Figure 2.

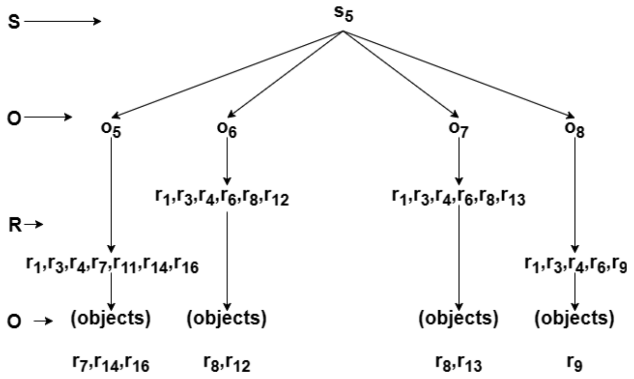

Fig. 2. SORO tree for s_5.

Table 10. Rules and Objects Corresponding to Each Attribute Value for *S.Designation*

Attribute values	Rules	Objects
DIR	r_1	$o_1, o_2, o_3, o_4, o_5, o_6, o_7, o_8, o_9$
HOD	r_2, r_3, r_4, r_5	$o_2, o_3, o_4, o_5, o_6, o_7, o_8, o_9$
REG	r_6, r_{15}, r_{16}	$o_2, o_3, o_4, o_5, o_6, o_7, o_8, o_9$
PROF	r_7, r_8, r_9, r_{10}	o_5, o_6, o_7, o_8, o_9
FINO	r_{11}	o_4, o_5
STU	r_{12}, r_{13}	o_6, o_7
CLRK	r_{14}	o_5

Table 11. Adjacency Matrix with Weights for *S.Designation*

	DIR	HOD	REG	PROF	FINO	STU	CLRK
DIR	∞	-1	-1	-1	-1	-1	-1
HOD	0	∞	∞	-1	-1	-1	-1
REG	0	∞	∞	-1	-1	-1	-1
PROF	0	0	0	∞	0	-1	-1
FINO	0	0	0	0	∞	0	-1
STU	0	0	0	0	0	∞	0
CLRK	0	0	0	0	0	0	∞

Now, we come to the second phase of our proposed solution, i.e., Step 2.1. Here, we determine the attribute value hierarchy for the subject attributes. In this example, we only generate the attribute value hierarchy of the subject attribute *S.Designation*. At first, we determine the rules corresponding to each of the attribute values of *S.Designation*. Next, we find the objects corresponding to the group of rules. The attribute values with their corresponding sets of rules and objects are given in Table 10. Using this table, we construct the adjacency matrix from which we can obtain the hierarchy graph for *S.Designation*. The hierarchy adjacency matrix is given in Table 11. The final hierarchy graph obtained using Algorithm 6 is the same as the attribute value graph given in Figure 1. The hierarchy values assigned to *DIR* is 4, *HOD* and *REG* are assigned a value 3, *PROF* and *FINO* are assigned a value of 2, and the remaining ones, i.e., *STU* and *CLRK*, are given a value of 1.

In Step 3.1 of the third phase, Algorithm 8 finds the minimal set of rules for s_5. In this case, it generates three minimal sets of rules for s_5; namely, $\{r_7, r_8, r_9\}$, $\{r_8, r_9, r_{14}\}$, and $\{r_8, r_9, r_{16}\}$. Let us randomly select the set $\{r_8, r_9, r_{14}\}$.

Finally, in Step 3.2, we assign the subject attribute-value pairs to s_5. The subject attribute-value pairs assigned to s_5 are $S.Designation : PROF, CLRK$ and $S.Department : CSE$. Now, $S.Designation$ has been assigned two values, out of which, $PROF$ has a higher hierarchy value of 2 as compared to $CLRK$ having a hierarchy value of 1. Therefore, we keep the attribute-value $S.Designation : PROF$, which is more meaningful, as a subject cannot be a professor and a clerk at the same time. Now, as $CLRK$ is removed from the attribute value for the attribute $S.Designation$, rule r_{14}, from which $CLRK$ was assigned, is removed from the minimal set of rules for s_5. Therefore, the set of rules to be used by s_5 becomes $\{r_8, r_9\}$. Thus, in addition to generating more meaningful assignment of attribute values, using hierarchy also reduces the number of rules used by each subject.

We would like to emphasize here that the proposed approach uses the same ABAC policy (the complete rule set) of the source organization in the target organization. In the algorithm, initially, for each user a collection of candidate rule sets is generated, using which the user can get all its accesses. The final set of rules is a minimal subset of these candidate rules, which ensures that all the desired accesses as per the ACL are obtained and yet no extraneous access is generated. It may be noted that more than one user can make use of the same rule to get access to the same set of objects. If there are several such cases, it implies that a higher number of rules is generalizable. Thus, the final rule set is much smaller than the candidate rule sets and, hence, administration of the resultant set of rules is much easier. Further, if some of the rules in the source policy are not required in the target organization, those rules are pruned after all the assignments have been done.

5 EXPERIMENTAL RESULTS

While organizations are gradually migrating to ABAC, data sets from such organizations have not yet been made publicly available. In the absence of real data sets, we have evaluated our proposed approach on the benchmark data sets introduced by the authors of Reference [20], which has been widely used for studies similar to ours, as they closely resemble real-world data in organizations using ABAC. However, as these data sets involve only a small number of subjects and objects, we also test our proposed approach on various synthetically generated ABAC policies, lists of accesses, lists of subjects, objects, and attribute value assignments for the objects. The algorithms for the proposed approach were implemented in Python 3.5.4 and executed on a 2.5GHz Intel i7 CPU having 16GB of RAM. The notations used for presenting the results are: number of subjects ($|S|$), number of objects ($|O|$), number of subject attributes ($|S_a|$), number of object attributes ($|O_a|$), number of rules in the ABAC policy ($|R|$), average number of rules used by a subject when attribute value is not considered ($|RS|$), and average number of rules used by each subject computed by taking attribute value hierarchies into account ($|RS_H|$). T and T_H represent the time for executing the algorithm not considering and considering hierarchical attribute values. Both T and T_H are measured in seconds. Additionally, we also show that the effect inclusion of hierarchical attribute values of the various entities on the *Weighted Structural Complexity* (WSC) of the generated set of subject attribute-value pairs. WSC of an assignment of subject attribute-value pairs S_v with n subjects is defined as [11]:

$$WSC(S_v) = w \times \sum_{i=1}^{n} WSC(S_v[i]), \tag{1}$$

where $S_v[i]$ is the set of subject attribute-value pairs associated with the ith subject and $WSC(S_v[i])$ represents the number of subject attribute values assigned to the ith subject.

Table 12. Average Size of Rule Set and Time for the Data Sets in Reference [20]

Dataset	$\|S\|$	$\|S_a\|$	$\|O\|$	$\|O_a\|$	$\|R\|$	$\|RS\|$	WSC	$T(in\ s.)$
Online video library	12	2	13	2	6	1.23	1.17	0.01
Healthcare	21	6	16	7	11	1.33	1.67	0.02
University	20	6	34	5	10	1.40	1.82	0.02
Project management	16	7	40	6	19	1.25	1.75	0.03

For instance, if Bob is associated with the attribute-value pairs $\{Designation = Professor\}$ and $\{subjects\ taken = OS, DBMS\}$, $WSC(Bob) = 3$. w is a user-defined weight. We have set $w = \frac{1}{n}$, i.e., WSC represents the average number of attribute values assigned to each subject.

Note that, as mentioned earlier, we assume that it is indeed possible to find attribute values for each subject (and correspondingly, object and environment) such that by using the rules in the policy to be adapted to, all required accesses are satisfied while no extraneous accesses are enabled. Therefore, there is no need to evaluate correctness, since it is guaranteed by design (i.e., both precision and recall will be 1). In the generated policies for the synthetic data sets, we have generated the accesses and policy in such a way that this assumption holds.

The average number of rules used by a subject and the time taken for execution of the proposed algorithm for policy adaptation for the various data sets given in Reference [20] are shown in Table 12. It is observed that, as the number of subjects and objects increases, the average number of rules used by each subject also increases. This can be attributed to the increase in the number of objects. Subjects are likely to access more objects if the number of objects is increased, which, in turn, increases the number of rules required to fulfill the accesses. The execution time of the algorithm also increases with increasing number of subjects, as it takes one iteration of the algorithm for generating a minimal set of rules for one subject. Since only single values are assigned to all the attributes for all the data sets, there is no removal of attribute values due to the presence of attribute values that are higher in the hierarchy. Therefore, in spite of attribute hierarchies being identified, the average number of rules used by each subject remains the same even when the attribute value hierarchies are used.

Next, we evaluate our proposed algorithm on various synthetic data sets. The synthetic data sets were generated by varying the number of subjects, objects, subject attributes, object attributes, and environment attributes. Each attribute has a distinct set of possible values. Moreover, we tried to ensure that the distribution of possible values of various attributes is as close to real life as possible. For instance, for some of the attributes, the corresponding values are uniformly distributed; whereas for some other attribute, the frequency of occurrence of some of the values is kept higher than that of the others. This mimics real-world scenarios; for example, that of a university, where the number of faculty members is less than the number of students while various departments have a roughly similar number of students barring a few exceptions. Fixing the number of subjects, objects, and the number of object attributes results in two cases: (i) variation in the number of subject attributes when the number of rules is kept constant and (ii) variation in the number of rules while keeping the number of subject attributes constant. Likewise, two cases arise when the number of subjects, objects, and object attributes is kept constant: (i) variation in the number of object attributes when the number of rules is fixed and (ii) variation in the number of rules when the number of object attributes is fixed.

Table 13 shows the variation in the average number of rules used by a subject and execution time with the number of rules and the number of subject attributes. It is observed that the average number of rules used by each subject increases when the number of rules is increased. This can be attributed to the fact that, in the presence of a higher number of rules, a subject is more likely

Table 13. Average Size of Rule Set and Time for Different Number of Rules and Subject Attributes

$\|S\| = 100, \|O\| = 200, \|O_a\| = 5$											
$\|R\| = 20$						$\|R\| = 30$					
$\|S_a\|$	$\|RS\|$	$\|RS_H\|$	WSC	$T(in\ s.)$	$T_H(in\ s.)$	$\|S_a\|$	$\|RS\|$	$\|RS_H\|$	WSC	$T(in\ s.)$	$T_H(in\ s.)$
3	2.98	2.98	2.29	0.95	1.02	3	4.39	3.98	2.58	1.52	1.64
5	3.14	2.97	2.27	1.30	1.47	5	3.53	2.84	2.22	2.66	2.80
7	2.48	2.16	2.14	1.68	1.97	7	4.43	4.04	2.79	2.76	2.97
$\|R\| = 40$						$\|R\| = 50$					
$\|S_a\|$	$\|RS\|$	$\|RS_H\|$	WSC	$T(in\ s.)$	$T_H(in\ s.)$	$\|S_a\|$	$\|RS\|$	$\|RS_H\|$	WSC	$T(in\ s.)$	$T_H(in\ s.)$
3	5.24	4.66	3.57	2.16	2.34	3	5.35	4.58	3.53	2.23	2.34
5	5.58	4.68	3.55	2.31	2.46	5	5.97	5.69	4.42	2.93	3.01
7	5.50	4.43	3.41	2.20	2.29	7	6.68	6.38	4.83	4.13	4.31

Table 14. Average Size of Rule Set and Time for Different Number of Rules and Object Attributes

$\|S\| = 100, \|O\| = 200, \|S_a\| = 5$											
$\|R\| = 40$						$\|R\| = 50$					
$\|O_a\|$	$\|RS\|$	$\|RS_H\|$	WSC	$T(in\ s.)$	$T_H(in\ s.)$	$\|O_a\|$	$\|RS\|$	$\|RS_H\|$	WSC	$T(in\ s.)$	$T_H(in\ s.)$
3	2.91	2.59	2.15	0.88	1.25	3	3.88	3.43	2.64	1.79	1.94
5	3.75	3.64	2.88	1.10	1.21	5	4.58	4.01	3.36	2.02	2.11
7	3.84	3.50	2.72	0.79	0.86	7	4.20	4.18	3.42	1.74	1.81
$\|R\| = 40$						$\|R\| = 50$					
$\|O_a\|$	$\|RS\|$	$\|RS_H\|$	WSC	$T(in\ s.)$	$T_H(in\ s.)$	$\|O_a\|$	$\|RS\|$	$\|RS_H\|$	WSC	$T(in\ s.)$	$T_H(in\ s.)$
3	3.48	3.20	2.42	3.16	3.42	3	5.28	5.08	3.86	3.35	3.54
5	5.23	4.84	4.43	2.82	2.98	5	5.68	5.22	4.14	2.73	2.86
7	5.88	5.37	3.36	2.92	3.03	7	6.58	6.27	5.71	3.44	3.89

to use a higher number of rules to gain all the desired accesses. For 20, 30, 40, and 50 rules, the percentage of rules that need to be evaluated in case of an actual access request are only 15.7%, 14.76%, 13.95%, and 13.36% (computed using the highest value of $|RS|$ for a particular number of rules), respectively. Moreover, when the hierarchy of the values of the subject attributes is utilized, the percentage of rules to be evaluated for deciding an access drops to 14.9%, 13.46%, 11.7%, and 12.76% (computed using the highest value of $|RS_H|$ for a particular number of rules), respectively. Therefore, in case of an actual access request, only a very small fraction of rules is required to be evaluated for deciding an access. With increase in the number of rules, the fraction of rules to be evaluated for deciding an access reduces. Thus, adapting to policies with higher number of rules is more advantageous. In contrast, there is relatively less variation in the average number of rules used by each subject with increase in the number of subject attributes. The execution time of the algorithm increases with the number of rules. The WSC of the assigned subject attribute-value pairs vary with the variation in the number of rules used by each subject. As the subject attribute-value pairs are assigned from the rules used by each subject, the WSC increases with the number of rules used by each subject.

Table 14 shows the effect of varying the number of rules and object attributes on the average number of rules used by a subject and the execution time of the algorithm. Unlike Table 13, here it can be seen that the average number of rules used by a subject increases with increase in the

Table 15. Average Size of Rule Set and Time for Different Number of Subjects and Objects

$\|S_a\| = 5, \|O_a\| = 5, \|R\| = 50$											
$\|S\| = 20$						$\|S\| = 50$					
$\|O\|$	$\|RS\|$	$\|RS_H\|$	WSC	$T(in\ s.)$	$T_H(in\ s.)$	$\|O\|$	$\|RS\|$	$\|RS_H\|$	WSC	$T(in\ s.)$	$T_H(in\ s.)$
20	3.70	2.75	2.05	0.04	0.05	20	3.78	2.82	2.39	0.07	0.09
50	5.19	3.51	2.62	0.08	0.10	50	5.04	4.32	3.58	0.14	0.17
100	5.17	4.65	4.19	0.16	0.20	100	5.62	4.81	3.22	0.44	0.55
200	5.45	4.85	3.98	0.73	0.80	200	6.52	4.98	3.87	1.48	1.64
$\|S\| = 100$						$\|S\| = 200$					
$\|O\|$	$\|RS\|$	$\|RS_H\|$	WSC	$T(in\ s.)$	$T_H(in\ s.)$	$\|O\|$	$\|RS\|$	$\|RS_H\|$	WSC	$T(in\ s.)$	$T_H(in\ s.)$
20	4.51	3.76	3.02	0.18	0.21	20	3.51	2.69	2.26	0.40	0.48
50	5.22	4.67	4.14	0.28	0.34	50	5.26	4.27	2.57	1.39	1.52
100	5.66	5.32	3.38	1.43	1.55	100	6.18	4.79	3.24	3.25	3.42
200	5.84	5.16	3.31	1.92	2.04	200	6.48	5.80	5.11	7.44	7.44
$\|S\| = 200$						$\|S\| = 500$					
$\|O\|$	$\|RS\|$	$\|RS_H\|$	WSC	$T(in\ s.)$	$T_H(in\ s.)$	$\|O\|$	$\|RS\|$	$\|RS_H\|$	WSC	$T(in\ s.)$	$T_H(in\ s.)$
500	5.97	5.43	3.56	15.54	18.75	500	6.78	5.91	5.18	67.89	73.82
1000	6.08	5.26	3.02	43.78	51.18	1000	6.67	6.02	5.08	285.76	330.17
2000	6.32	5.48	4.14	1157.61	1477.28	2000	7.08	6.48	5.98	4387.54	5682.16
5000	6.47	5.53	3.38	2577.09	3169.45	5000	7.35	6.61	6.02	12864.56	15233 .24

number of object attributes. This can be accredited to the use of additional rules for satisfying the increasing number of object attributes and their associated values. A similar trend can be observed when the number of rules is increased, i.e., the average number of rules used by a subject increases with increase in the number of rules. For 20, 30, 40, and 50 rules, the percentage of rules to be evaluated in case of an access request is found to be 19.20%, 15.27%, 14.7%, and 13.16% (computed using the highest value of $|RS|$ for a particular number of rules), respectively. When the hierarchy is considered, the percentage of rules to be evaluated in case of an access request drops to 18.20%, 13.92%, 13.42%, and 12.54% (computed using the highest value of $|RS_H|$ for a particular number of rules), respectively. The execution time of the algorithm increases with number of rules. As seen in Table 13, similar variation is observed in the WSC of the assigned subject attribute-value pairs.

The results given so far are on a fixed number of subjects and objects. Table 15 shows the variation of the average number of rules used by each subject and the time taken for policy adaptation with variation in the number of subjects and objects. It is seen that the average number of rules used by each subject increases with increase in the number of objects. Increase in the number of objects results in subjects accessing more objects. Accessing more objects makes it more likely for the subjects to use more rules to access them. The execution time of the algorithm increases with the number of both subjects and objects. It is also observed that the WSC of the assigned subject attribute-value pairs is independent of the number of subjects and objects.

We next show the effect of inclusion of hierarchy in object attribute values and subject attribute values along with the subject attribute value hierarchies in Table 16. It has already been observed that inclusion of hierarchical attribute values for subject attributes results in a better assignment of attribute values to the different subjects, i.e., there is a drop in the WSC value for the assigned set of subject attribute-value pairs. Similarly, inclusion of hierarchical attribute values for object and environment attributes also results in removal of attribute values having lower influence, thereby reducing the WSC.

Table 16. Variation in WSC Due to Inclusion of Hierarchical Attributes

$\|S\| = 100, \|O\| = 100, \|R\| = 50$								
$\|S_a\|$	WSC_{old}	WSC_{new}	$\|O_a\|$	WSC_{old}	WSC_{new}	$\|E_a\|$	WSC_{old}	WSC_{new}
3	3.95	3.58	3	4.15	3.49	3	4.01	3.68
5	4.96	4.42	5	4.59	4.07	5	4.63	4.27
7	5.27	4.83	7	5.44	5.16	7	5.58	4.72
10	6.29	5.57	10	5.86	5.39	10	6.13	5.59
15	7.74	6.84	15	7.52	6.63	15	8.25	7.47

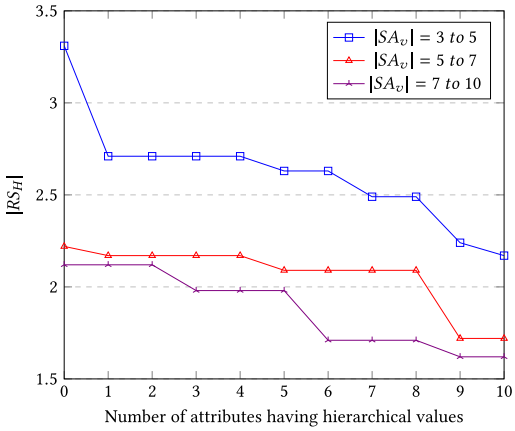

Fig. 3. Average size of rule set with the number of attributes with hierarchical values.

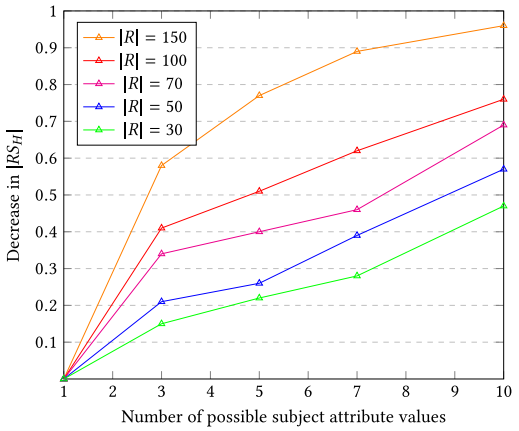

Fig. 4. Measure of decrease in the average size of rule sets with the number of possible values of subject attributes.

So far, we have seen the effect of varying various parameters on the average number of rules used by a subject. Next, we explore the impact of considering the obtained attribute hierarchy data on policy adaptation. It has already been observed in Tables 13, 14, and 15 that using the extracted attribute hierarchy data for policy adaptation helps in reducing the percentage of rules to be evaluated for deciding an access. Apart from this, as seen in Section 4.4, using the attribute value hierarchy data results in more appropriate and realistic assignment of subject attribute-value pairs to the subjects. It is imperative that not all subject attributes will have hierarchical values. Figure 3 shows the effect of varying the number of subject attributes with hierarchical attribute values on the average number of rules used by each subject. It is seen that for different ranges in the number of values a particular attribute can have, the average number of rules used by each subject decreases with the number of attributes having hierarchical attribute values. It may be noted that if there is no change in the average number of rules for an attribute with hierarchical values, it implies no new rules were removed due to inclusion of the attribute.

Finally, in Figure 4, we show reduction in the average number of rules with the number of possible attribute values. For various policy sizes, it is observed that the higher the number of possible values an attribute can have, the higher the reduction in the average number of rules used by each subject when hierarchical attribute values are taken into account. This can be attributed to the fact that when an attribute can have a higher number of possible values, each subject attribute is more likely to be assigned a higher number of values and it is more likely to have more hierarchical levels for a higher number of possible attribute values.

6 RELATED WORK

While work on ABAC policy mining is still in its infancy, there is a considerable body of literature on role mining in RBAC. Colantonio et al. [2] use attribute data and an access control list for mining role-based policies. Molloy et al. [11] form RBAC policies with low weighted structural complexity (WSC). These policies consist of meaningful roles such that the user to role mapping can be expressed as a predicate over a set of subject attribute-value pairs. One major limitation of these algorithms is that they use attributes like *uid,* which uniquely identify a user in a user to role assignment. This results in some users not being characterized by a subject attribute-value pair expression. In another work, Molloy et al. [10] utilize machine learning algorithms for role mining. They use supervised learning techniques to train classifiers for associating permissions and roles. Their algorithm takes permissions, roles, and role to permission assignments as training data. The output is a support vector machine (SVM) classifier that automatically allocates new permissions to roles. In addition, they also consider another scenario where classifiers are trained using a supervised learning algorithm to automate user to role assignments. The users, roles, user attribute values, and user to role assignments are given as input to the classifier. Lim et al. [7] use evolutionary algorithms to construct and improve security policies. Though they do not consider ABAC policy mining, some complex problems involving risk-based policies are surveyed. Vaidya et al. [19] identify the requirements for similarity evaluation from the context of access control and propose a formal framework built upon change detection that computes similarity between policies and other multi-policy management operations such as policy migration. Their framework evaluates similarities between security policies by taking into account various access control semantics and, given a set of policies, they find a common organizational policy with the lowest cost of migration.

Xu et al. [20] propose one of the early approaches for mining ABAC policies using a bottom-up approach. Their algorithm constructs an ABAC policy from Access Control Lists (ACLs) and attribute data. It selects specific accesses from the given ACL to construct candidate rules that are further generalized to cover additional accesses in the given ACL. After covering the given ACL with the constructed candidate rules, the candidate rules are simplified and merged to improve the constructed ABAC policy. Finally, the algorithm selects the highest-quality candidate rules that are added to the generated policy. While this approach is capable of constructing an ABAC policy, the number of attribute-value pairs in the constructed rules is also crucial. Rules consisting of a large number of attribute-value pairs result in increased time for access decisions when an access request is made. Therefore, besides minimizing the number of attribute-value pairs in the total policy, constraining the number of attribute-value pairs associated with each rule is also helpful. Gautam et al. [5] consider a variant of the policy mining problem for ABAC that takes an ACM as input and constructs a minimal set of ABAC authorization rules in such a way that each rule has at most a fixed number of associated attribute-value pairs and the weight of the constructed policy is also minimal.

Talukdar et al. [17] show that the problem of policy mining in ABAC is similar to that of identifying functional dependencies in database tables. In this context, they propose an ABAC policy mining algorithm called ABAC-FDM that exhaustively enumerates all possible subject-object pairs. As ABAC-FDM is an exhaustive approach, it is not scalable. To address this limitation, they propose a more efficient ABAC policy mining algorithm called ABAC-SRM, which uses a bottom-up approach to initially identify a set of candidate rules, and from the set of candidate rules, the most general rules are selected to construct an ABAC policy. Vaidya et al. [18] propose the minimal perturbation role mining problem for RBAC that enables an organization to migrate from an existing set of roles to a new set of roles with minimum disruption. Migration of roles is required for

addressing the gradual changes that occur and also the new requirements in organizational processes. In a similar manner, PolAP-AVH can also be used to migrate to a new assignment of subject attribute-value pairs to handle any change in the functional requirements of an organization. In spite of such similarities, our work differs from Reference [18] as PolAP-AVH helps organizations to migrate to ABAC from other access control models. Additionally, it also facilitates sharing of resources among organizations.

Recently, we proposed the first known algorithm for effective policy adaptation in ABAC [3]. Although this approach is also based on the assignment of appropriate values to subject attributes for different subjects, it does not consider the hierarchy of different attribute values. Nor does it consider environmental conditions. As seen in Section 4.4, extracting hierarchical information from the policy to be re-used and utilizing it for policy adaptation results in the assignment of more meaningful subject attribute-value pairs. Moreover, it helps in further reducing the number of rules used by each subject.

7 CONCLUSION AND FUTURE WORK

In this article, we have introduced the problem of policy adaptation using hierarchical attribute values to provide an effective path for migrating to ABAC using an existing policy. The optimal policy adaptation problem using hierarchical attribute values has been shown to be NP-Complete, and a heuristic solution has been proposed and evaluated. Further, this work is the first to consider environment attributes seamlessly with the subject and object attributes, an aspect commonly ignored in ABAC policy mining work.

However, a limitation of this work is that the organization pursuing policy adaptation needs to have the same set of attributes as the organization whose policy it is planning to adapt to. Future work in this area would be to design approaches that facilitate policy adaptation among organizations having attribute sets that do not exactly match. A possible direction is to use ontology-based attribute set mapping. Additionally, novel heuristics can be proposed that further reduce the number of generated rules.

REFERENCES

[1] E. Bertino, B. Catania, M. L. Damiani, and P. Perlasca. 2005. GEO-RBAC: A spatially aware RBAC. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies (SACMAT'05)*. 29–37.

[2] A. Colantonio, R. D. Pietro, and N. V. Verde. 2012. A business-driven decomposition methodology for role mining. *Comput. Sec.* 31, 7 (2012), 844–855.

[3] S. Das, S. Sural, J. Vaidya, and V. Atluri. 2017. Policy adaptation in attribute-based access control for inter-organizational collaboration. In *Proceedings of the International Conference on Collaboration and Internet Computing*. 136–145.

[4] Andrew Gainer Dewar and Paola Vera Licona. 2016. The minimal hitting set generation problem: Algorithms and Computation. In *SIAM J. Discrete Mathematics* 31, 1 (2016), 63–100.

[5] M. Gautam, S. Jha, S. Sural, J. Vaidya, and V. Atluri. 2017. Poster: Constrained policy mining in attribute based access control. In *Proceedings of the ACM Symposium on Access Control Models and Technologies*. 121–123.

[6] V. C. Hu, D. Ferraiolo, D. R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone. 2014. *Guide to Attribute-Based Access Control (ABAC) Definition and Considerations*. Technical Report. NIST Special Publication 800-162.

[7] Y. T. Lim. 2010. *Evolving Security Policies*. Ph.D. Dissertation, University of York, UK.

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

[9] B. Mitra, S. Sural, J. Vaidya, and V. Atluri. 2017. Migrating from RBAC to temporal RBAC. *IET Inform. Sec.* 11 (2017), 294–300.

[10] Ian Molloy, Hong Chen, Tiancheng Li, Qihua Wang, Ninghui Li, Elisa Bertino, Seraphin Calo, and Jorge Lobo. 2008. Mining roles with semantic meanings. In *Proceedings of the Symposium on Access Control Models and Technologies (SACMAT'10)*. 21–30.

[11] I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. Calo, and J. Lobo. 2010. Mining roles with multiple objectives. *ACM Trans. Inform. Syst. Sec.* 13, 4 (2010), 36:1–36:35.

[12] I. Ray and M. Toahchoodee. 2007. A spatio-temporal role-based access control model. In *Proceedings of the 21st IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec'07)*. 211–226.
[13] R. S. Sandhu. 1993. Lattice-based access control models. In *Computer* 26, 11 (1996), 9–19.
[14] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. 1996. Role-based access control models. *IEEE Comput.* 29, 2 (1996), 38–47.
[15] Ravi S. Sandhu and Pierangela Samarati. 1994. Access control: Principle and practice. In *IEEE Commun. Mag.* 32, 9 (1994), 40–48.
[16] D. Servos and S. L. Osborn. 2017. Current research and open problems in attribute-based access control. In *ACM Computing Surveys* 49, 4 (2017), 65:1–65:45.
[17] T. Talukdar, G. Batra, J. Vaidya, V. Atluri, and S. Sural. 2017. Efficient bottom-up mining of attribute based access control policies. In *Proceedings of the International Conference on Collaboration and Internet Computing*. 339–348.
[18] Jaideep Vaidya, Vijayalakshmi Atluri, Qi Guo, and Nabil Adam. 2008. Migrating to optimal RBAC with minimal perturbation. In *Proceedings of the ACM Symposium on Access Control Models and Technologies (SACMAT'08)*. 11–20.
[19] J. Vaidya, B. Shafiq, V. Atluri, and D. Lorenzi. 2015. A framework for policy similarity evaluation and migration based on change detection. In *Proceedings of the International Conference on Network and System Security*. 191–205.
[20] Z. Xu and S. D. Stoller. 2015. Mining attribute-based access control policies. In *IEEE Transactions on Dependable and Secure Computing (TDSC)*. 533–545.

Received February 2018; revised January 2019; accepted February 2019