EXPRESSIVE POWER, SAFETY AND CLOUD IMPLEMENTATION OF ATTRIBUTE AND RELATIONSHIP BASED ACCESS CONTROL MODELS

by

TAHMINA AHMED, M.Sc.

DISSERTATION
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

COMMITTEE MEMBERS:
Ravi Sandhu, Ph.D., Chair
Jianwei Niu, Ph.D.
Gregory White, Ph.D.
Weining Zhang, Ph.D.
Ram Krishnan, Ph.D.

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Sciences
Department of Computer Science
December 2017

DEDICATION

I would like to dedicate this thesis to my mom Mrs. Nigar Ahmed and my dad Dr. Jalal Uddin Ahmed for their tremendous support, unconditional love and inspiration in every steps of my life. I also like to dedicate it to my kids Taheem Mustaneer, Taafeef Muntasir and Mysha Tazmeen who are my infinite source of energy to walk extra miles.

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation and profound gratitude to my supervising professor Dr. Ravi Sandhu who helped me with his inspiring ideas, critical comments, and constant encouragement. I learned from him how to challenge my own ideas to build a solid base. I am thankful to him for his guidance in every professional and personal aspects throughout my doctoral studies and beyond.

I like to thank Dr. Jaehong Park for his guidance, help and effort to organize my dissertation proposal.

I also like to thank Dr. Jianwei Niu, Dr. Gregory B. White, Dr. Weining Zhang and Dr. Ram Krishnan for their valuable comments, time and insight in organizing this dissertation.

I also like to thank Farhan Patwa, for his guidance and encouragement in learning and contributing to OpenStack.

I would like to acknowledge our faculties from the Computer Science department, Suzanne Tanaka from the ICS, Susan Allen and other staffs from CS department for their support throughout my doctoral studies. My gratitude also goes to all my friends and colleagues at UTSA- specially my fellow ICS labmates.

This Doctoral Dissertation was produced in accordance with guidelines which permit the inclusion as part of the Doctoral Dissertation the text of an original paper, or papers, submitted for publication. The Doctoral Dissertation must still conform to all other requirements explained in the Guide for the Preparation of a Doctoral Dissertation at The University of Texas at San Antonio. It must include a comprehensive abstract, a full introduction and literature review, and a final overall conclusion. Additional material (procedural and design data as well as descriptions of equipment) must be provided in sufficient detail to allow a clear and precise judgment to be made of the importance and originality of the research reported.

It is acceptable for this Doctoral Dissertation to include as chapters authentic copies of papers already published, provided these meet type size, margin, and legibility requirements. In such cases, connecting texts, which provide logical bridges between different manuscripts, are mandatory. Where the student is not the sole author of a manuscript, the student is required to make an explicit statement in the introductory material to that manuscript describing the students contribution to the work and acknowledging the contribution of the other author(s). The signatures of the Supervising Committee which precede all other material in the Doctoral Dissertation attest to the accuracy of this statement.

December 2017

EXPRESSIVE POWER, SAFETY AND CLOUD IMPLEMENTATION OF ATTRIBUTE AND RELATIONSHIP BASED ACCESS CONTROL MODELS

Tahmina Ahmed, Ph.D.
The University of Texas at San Antonio, 2017

Supervising Professor: Ravi Sandhu, Ph.D.

For the last few years Attribute Based Access Control (ABAC) has been emerging as the next dominant form of access control. According to a 2014 NIST special publication, "ABAC enables more precise access control model as it can consider numerous attributes in authorization decision." ABAC can unify the advantages of the traditional discretionary, mandatory and role-based access control models by using appropriate attributes, while going beyond the capabilities of these. ABAC has become recognized as a model expressive enough to define finer-grained and flexible authorization policies suitable for modern application domains such cloud computing and Internet of Things. Meanwhile, in recent years, various online social network (OSN) applications such as Facebook, Twitter and LinkedIn have become widely used. In OSNs, authorization for users' access to specific content is typically based on the interpersonal relationships between the accessing user and content owner. Recently ReBAC has been expanded to cover systems beyond OSNs. Efforts to combine ReBAC and ABAC have also been published.

This dissertation makes fundamental contributions to our understanding of ABAC and ReBAC from three perspectives. Firstly, it clarifies and resolves conflicting claims in the literature regarding the expressive power of ABAC and ReBAC. It has been argued, on one hand, that attributes can encode relationships so ABAC subsumes ReBAC. On the other hand, it has been claimed that the multilevel or composed relations of ReBAC (such as friend of friend) bring fundamentally new capabilities. This dissertation develops separate classifications of ABAC and ReBAC models with respect to salient structural and dynamic properties. It shows the equivalence, dominance or non-comparability of the expressive power of various model classes in these classifications. The results

of this analysis show that ABAC and ReBAC, when defined with sufficient generality, are equivalent in expressive power. For less general forms of ABAC and ReBAC the relative expressive power depends strongly on the details of the respective models.

Secondly, this dissertation analyzes the safety and expressive power of an existing ABAC model, viz. ABAC_α. ABAC_α is designed with just sufficient capabilities to configure commonly used forms of discretionary, mandatory and role-based access control. In particular ABAC_α restricts attribute values to be from finite fixed domains. The safety analysis of ABAC_α is shown to be decidable by providing a reduction from ABAC_α to safety decidable $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$, which is a structurally different ABAC model with finite fixed domains. Two enhanced versions of ABAC_α are defined. One of these is shown to be equivalent in expressive power to $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$. The other is shown to have undecidable safety and thus expressive power beyond $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$. The question of whether ABAC_α is strictly less expressive than $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ or equivalent to it, is left open.

Finally, the dissertation introduces a novel form of ReBAC model (OOReBAC) considering object-to-object relationship independent of users to control access of resources. A proof-of-concept implementation of OOReBAC for multicloud resource sharing using the open source OpenStack cloud platform and specifically its Swift object storage service is provided.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1: INTRODUCTION

Attribute Based Access Control has become very popular due to its generalized structure and flexibility to specify policy. The concept of using attributes for access control has been around for many years, e.g., the X.500 standard [45] was an early effort for managing object information with attributes. Attribute-based access control (ABAC) is considered one of the most generalized forms of access control as it can capture the salient features of discretionary access control (DAC), mandatory access control (MAC) and role-based access control (RBAC) using appropriate attributes such as access control lists, security labels and roles respectively [82], and bring in additional elements such as location and time. ABAC enables more precise access control as it can consider a higher number of discrete inputs into an access control decision [77]. Different ABAC models with rich policy languages and sophisticated features have been proposed [81, 82, 86, 103, 125, 139].

Meanwhile, in recent years, on-line social networks (OSNs), such as Facebook, Twitter and LinkedIn, have introduced an alternate form of authorization based on mostly the interpersonal relationships between the accessing user and the content owner, rather than on attributes. Different access control models have been proposed in this context [39, 43, 51–53, 68, 69]. These are generally called Relationship-Based Access Control (ReBAC) models. OSN ReBAC models mostly use user-to-user relationships [39, 43, 52, 53, 68, 69] while user-to-resource and resource-to-resource relationships have also been considered in some cases [41, 51]. Several access control models have been proposed for OSN ReBAC considering both single and multiple relationship types for authorization policy specification [41, 51, 52, 68]. Subsequently, additional models have been proposed to extend and generalize these OSN ReBAC models so that they can be applicable to computing systems beyond OSNs [19, 56, 67, 110].

1.1 Motivation

ABAC has been around for a long time and can be viewed as a generalization, unification and extension of earlier access control concepts including discretionary, mandatory and role-based

access control. ReBAC is relatively recent, with its initial motivation stemming from its essential application in online social networks but now generally regarded as having broader applicability. Both have considerable applications in industry, and are anticipated to continue being important for the foreseeable future. Though a number of formal models have been proposed for both ABAC and ReBAC and a considerable body of research has been published, still there is no well-accepted consensus ABAC or ReBAC model as we have seen for traditional access control models (viz. DAC, MAC and RBAC). One reason for this is there are very few attempts on doing formal study to analyze the core characteristics of ABAC and ReBAC. Existing literature mainly deals with developing and formalizing sophisticated models where most of them are domain specific.

ABAC for web services [139] proposed an ABAC model for web service authorization, while [125] defined an ABAC model for semantic web technology. UCON [103] was proposed to capture authorization continuity and attribute mutability. [86] defines an ABAC model for service oriented architecture considering requester's privacy preference. ABAC_α [82] is proposed to configure DAC, MAC and RBAC, while ABAC_β [81] extends ABAC_α to incorporate different RBAC extensions. NIST ABAC [77] provides a detailed explanation of ABAC concepts and considerations for deployment of enterprise ABAC capabilities. XACML [98] proposes a standardized mechanism to specify ABAC authorization policy, request and policy evaluation. Attribute-based encryption supports fine-grained sharing of encrypted data [34, 48, 91, 100, 106, 112]. On the other hand, ReBAC is relatively new. Most of the existing ReBAC models are defined for OSNs [31, 41, 43, 51–53, 68, 72, 101]. Recently a few works have been published which consider ReBAC beyond OSN, for general computing systems [39, 56, 58, 67, 69, 109]

1.2 Problem Statement

There is a fundamental lack of understanding regarding the relationship between ABAC and ReBAC, reflected in the fact that claims in the literature exist in support of the conflicting views that ABAC subsumes ReBAC on one hand, and that ReBAC brings additional capabilities beyond ABAC. At the same time there is a proliferation of ABAC models without a formal understand-

ing of their safety properties and relative expressive power. Finally, the potential of ReBAC has only recently been recoginzed and there remain many directions in which ReBAC models can be developed.

1.3 Summary of Contribution

The major contributions of this dissertation are as follows.

- A conceptual and semi-formal comparison between attribute and relationship based access control models
- Safety analysis of an existing attribute based access control model (ABAC_α)
- Safety and expressive power analysis of two enhancements of ABAC_α
- A formal representation of an object-to-object relationship-based access control model (OOReBAC) and its implementation in open source IaaS cloud platform Openstack object storage Swift

1.4 Organization of the Dissertation

Chapter 2 gives a brief background and preliminary concepts of ABAC, UCON, ReBAC, expressive power comparison framework and the open source cloud IaaS platform Openstack. Chapter 3 provides a conceptual and semi-formal comparison between Attribute and Relationship Based Access Control Models. Chapter 4 proves the decidability of safety in ABAC_α by reducing it to safety of finite domain pre-UCON (which is previously known to have decidable safety). It further develops an enhancement of ABAC_α, viz. $\mathrm{ABAC}_\alpha^{\mathrm{AM}}$, and shows that $\mathrm{ABAC}_\alpha^{\mathrm{AM}}$ is equivalent in expressive power to finite domain pre-UCON. It also presents another ABAC_α extension, viz. $\mathrm{ABAC}_\alpha^{\mathrm{MI}}$, with infinite domain entity attributes and shows that safety of $\mathrm{ABAC}_\alpha^{\mathrm{MI}}$ is undecidable. Chapter 5 develops an object-to-object relationship based access control model and presents a proof of concept implementation in open source cloud platform Openstack. Chapter 6 concludes the dissertation.

Chapter 2: BACKGROUND

This chapter reviews background material germane to the dissertation research contributions described in the following three chapters. This review includes a discussion of relevant ABAC, usage control (UCON) and ReBAC models, a formal framework for comparing expressive power of access control models and the open-source OpenStack platform for cloud computing.

2.1 The ABAC_α Model

ABAC has been studied for a long time and many different formal models have been proposed [77, 81, 82, 86, 103, 125, 139]. Several of these are application specific or limited to a specific domain. ABAC for web services [139] proposed an ABAC model for web service authorization, while [125] defined an ABAC model for semantic web technology. UCON [103] was proposed to capture authorization continuity and attribute mutability. [86] defines an ABAC model for service oriented architecture considering requester's privacy preference. ABAC_α [82] is proposed to configure DAC, MAC and RBAC, while ABAC_β [81] extends ABAC_α to incorporate different RBAC extensions. NIST ABAC [77] provides a detail explanation of ABAC concepts and considerations for deployment of enterprise ABAC capabilities. XACML [98] proposes a standardized mechanism to specify ABAC authorization policy, request and policy evaluation. Attribute-based encryption supports fine-grained sharing of encrypted data [34, 48, 91, 100, 106, 112].

In this section we particularly present a review of the ABAC_α model. This material is primarily relevant to the results developed in Chapter 4.

2.1.1 The ABAC_α Formal Model (Review)

ABAC_α is an ABAC model that has "just sufficient" features to be "easily and naturally" configured to do DAC, MAC and RBAC [82]. The core components of this model are: users (U), subjects (S), objects (O), user attributes (UA), subject attributes (SA), object attributes (OA), permissions (P), authorization policy, creation and modification policy, and policy languages. The structure of

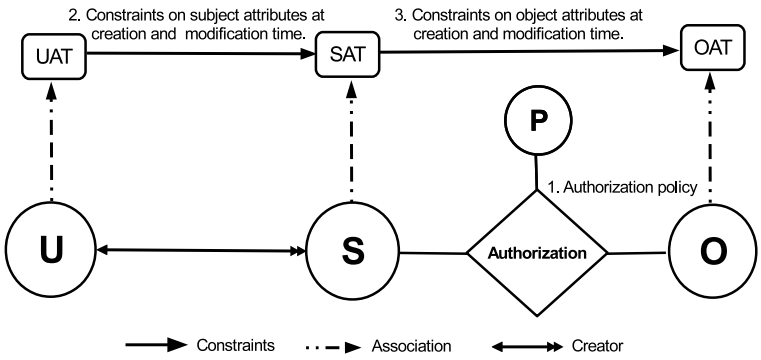


Figure 2.1: ABAC_α Model (adapted from [82])

ABAC_α model is shown in Figure 2.1. Table 2.1 gives the formal definition of ABAC_α.

Users, Subjects, Objects and their Attributes

Users (U) represent human beings in an ABAC_α system who create and modify subjects, and access resources through subjects. **Subjects** (S) are processes created by users to perform some actions in the system. ABAC_α resources are represented as **Objects** (O). Users, subjects and objects are mutually disjoint in ABAC_α, and are collectively called entities. **NAME** is the set of all names for various entities in the system. **Attributes** are set-valued or atomic-valued functions which take an entity (user, subject or object) and return a value from a finite set of atomic values. Each user, subject, object is associated with a finite set of user attributes (UA), subject attributes (SA) and object attributes (OA) respectively. Each attribute is a set-valued or atomic-valued function. **attType** is a function that returns type of the attribute, i.e., whether it is set or atomic valued. **SCOPE** represents the domain of an attribute which is a finite set of atomic values. Potentially infinite domain attribute such as location, age are represented as large finite domains. For each attribute att, SCOPE(att) can be an unordered, a totally ordered or a partially ordered set. **Range**(att) is a finite set of all possible atomic or set values for attribute att. Each attribute takes a user or a subject or an object, and returns a value from its range. **SubCreator** is a system function which specifies the creator of a subject. SubCreator is assigned by the system at subject creation time, and cannot change. UAVT, SAVT, OAVT are sets of all possible **Attribute Value Tuples** for users,

Table 2.1: ABAC_α Formal Model

Basic Sets and Functions

U, S, O are finite sets of existing users, subjects and objects
UA = {ua_1, ua_2, ... ua_l }, finite set of user attributes
SA = {sa_1, sa_2, ... sa_m }, finite set of subject attributes
OA = {oa_1, oa_2, ... oa_n}, finite set of object attributes
SubCreator: S $\rightarrow$ U. A system function, specifies the creator of a subject.
attType: UA $\cup$ SA $\cup$ OA $\rightarrow$ {set, atomic}
For each attribute att$\in$ UA $\cup$ SA $\cup$ OA:
SCOPE(att) denotes the finite set of atomic values for attribute att.
Range (att) represents a finite set of atomic or set values as the range of att.

$$\mathrm{Range(att)} = \begin{cases} \mathrm{SCOPE(att)} & \text{attType(att) = atomic.} \\ 2^{\mathrm{SCOPE(att)}} & \text{attType(att) = set.} \end{cases}$$

ua_i: U $\rightarrow$ Range(ua_i), $\mathrm{ua}_i \in$ UA
sa_j: S $\rightarrow$ Range(sa_j), $\mathrm{sa}_j \in$ SA
oa_k: O $\rightarrow$ Range(oa_k), $\mathrm{oa}_k \in$ OA

Tuple Notation

UAVT $\equiv \times_{i=1}^{l}$ Range(ua_i), set of all possible attribute value tuples for users
SAVT $\equiv \times_{j=1}^{m}$ Range(sa_j), set of all possible attribute value tuples for subjects
OAVT $\equiv \times_{k=1}^{n}$ Range(oa_k), set of all possible attribute value tuples for objects
uavtf: U $\rightarrow$ UAVT, current attribute value tuple for a user
savtf: S $\rightarrow$ SAVT, current attribute value tuple for a subject
oavtf: O $\rightarrow$ OAVT, current attribute value tuple for an object

Authorization Policy

P = {p_1, p_2, ... p_n}, a finite set of permissions.

- **Authorization on Object**

  For each p$\in$ P, Authorization$_p$(s:S,o:O) returns true or false.

  Specified in language LAuthorization.

Creation, Deletion and Modification Policy

Subject Creation Policy:

- **Subject Creation by User**

  ConstrSubCreatebyUser(u:U,s:NAME,$savt$:SAVT) returns true or false. Specified in language LConstrSub.

Subject Deletion Policy:

- **Subject Deletion by User**

  In ABAC_α subject deletion has fixed policy.

  A subject can be deleted only by its creator.

Subject Modification Policy:

- **Subject Modification by User**

  ConstrSubModbyUser(u:U,s:S,$savt$:SAVT) returns true or false. Specified in language LConstrSubMod.

Object Creation Policy:

- **Object Creation by Subject**

  ConstrObjCreatebySub(s:S,o:NAME,$oavt$:OAVT) returns true or false. Specified in language LConstrObj.

Object Modification Policy:

- **Object Modification by Subject**

  ConstrObjModbySub(s:S,o:O,$oavt$:OAVT) returns true or false. Specified in language LConstrObjMod.

Policy Languages

Each policy language is an instantiation of the Common Policy Language CPL that varies only in the values it can compare. Table 2.2 defines CPL for ABAC_α. Table 2.3 shows the set and $atomic$ instantiation for different languages.

Functional Specification

ABAC_α operations are formally specified in Table 2.4

Table 2.2: Definition of CPL

CPL
φ ::= $\varphi \wedge \varphi$ | $\varphi \vee \varphi$ | (φ) | $\neg \varphi$ | $\exists$ x $\in$ set.φ | $\forall$ x $\in$ set.φ | set setcompare set | $atomic \in set$ | $atomic$ atomiccompare $atomic$ setcompare ::= $\subset$ | $\subseteq$ | $\not\subseteq$ atomiccompare ::= $<$ | = | $\leq$

subjects and objects respectively. The functions uavtf, savtf and oavtf, return current attribute value tuples for a particular user, subject or object respectively.

Authorization Policy

ABAC_α authorization policy consists of a single authorization policy for each permission. **Permissions** are privileges that a user can hold on objects and exercise through subjects. It enables access of a subject on an object in a particular mode, such as read or write. P = {p_1, p_2, ... p_n} is a finite set of permissions. Each **Authorization Policy** is a boolean function which is associated with a permission, and takes a subject and an object as input and returns true or false based on the boolean expression built from attributes of that subject and object.

Creation and Modification Policy

User creation, attribute value assignment of user at creation time, user deletion and modification of a user's attribute values is done by security administrator, and is outside the scope of ABAC_α. Subject creation and assigning attribute value to subject during creation time is constrained by the values of user attributes. Only creator is allowed to terminate and modify attributes of a subject. Modification of subject attributes is constrained by the creating user's attribute values, and existing and new attribute values of the concerned subject.[1] Objects are created by subjects. Object creation and attribute value assignment at creation time is constrained by creating subject's attribute values and proposed attribute value for the object. Modification of object attribute value is constrained by subject and object's existing attribute values and proposed attribute values for object. ABAC_α has

[1] In the original definition of ABAC_α [82] subject creation and modification have identical policies. However, a correct configuration of MAC in ABAC_α requires different policies for these two operations. Hence, we define ABAC_α here to have separate policies for these two operations.

Table 2.3: Definition of Languages for ABAC_α

Language	*set*	*atomic*
LAuthorization	setsa(s) | setoa(o)	atomicsa(s) | atomicoa(o)
LConstrSub	setua(u) | setsa′(s)	atomicua(u) | atomicsa′(s)
LConstrSubMod	setua(u) | setsa(s) | setsa′(s)	atomicua(u) | atomicsa(s) | atomicsa′(s)
LConstrObj	setsa(s) | setoa′(s)	atomicsa(s) | atomicoa′(o)
LConstrObjMod	setsa(s) | setoa(o)| setoa′(o)	atomicsa(s) | atomicoa(o)| atomicoa′(o)

Table 2.4: Functional Specification of ABAC_α Operations

Operations	**Conditions**	**Updates**
Access$_p$(s,o)	s∈ S ∧ o∈ O ∧ Authorization$_p(s, o)$	
CreateSubjectbyUser (u,s:NAME,$savt$:SAVT)	u∈U ∧ s∉S ∧ ConstrSubCreatebyUser(u, s, $savt$)	S′ = S∪{s} SubCreator(s) = u savtf(s) = $savt$
DeleteSubjectbyUser (u,s:NAME)	s∈S ∧ u∈U ∧ SubCreator(s) = u	S′ = S\{s}
ModifySubjectAttbyUser (u,s:NAME,$savt$:SAVT)	u∈U ∧ s∈S ∧ SubCreator(s) = u ∧ ConstrSubModbyUser(u, s, $savt$)	savtf(s) = $savt$
CreateObjectbySubject (s,o:NAME,$oavt$:OAVT)	s∈S ∧ o∉O ∧ ConstrObjCreatebySub(s,o,$oavt$)	O′ = O∪{o} oavtf(o) = $oavt$
ModifyObjectAttbySubject (s,o:NAME,$oavt$:OAVT)	s∈S ∧ o∈O ∧ ConstrObjModbySub(s,o,$oavt$)	oavtf(o) = $oavt$

subject deletion however there is no object deletion. An existing subject can be deleted only by its creator.

Policy Languages

Each policy is expressed using a specific language. CPL is the common policy language part for each language. Each language is a CPL instantiation with different values for set and $atomic$. CPL is defined in Table 2.2.

Authorization Policy: The boolean expression of authorization policy is defined using the language LAuthorization which is a CPL instantiation where set and $atomic$ refers to the set and atomic valued attribute of concerned subject and object.

Creation and Modification Policy: Subject creation, subject attribute modification, object creation and object attribute modification policies are all boolean expressions and defined using LConstrSub, LConstrSubMod, LConstrObj and LConstrObjMod respectively. LConstrSub is a CPL in-

stantiation where set and $atomic$ refers to the set and atomic valued attribute of creating user and proposed attribute values for subject being created. LConstrSubMod is a CPL instantiation where set and $atomic$ refers to the set and atomic valued attribute value of concerned user and subject and proposed attribute value for subject. LConstrObj is a CPL instantiation where set and $atomic$ refers to the set and atomic valued attribute value of creating subject and proposed attribute value for object being created. LConstrObjMod is a CPL instantiation where set and $atomic$ refers to the set and atomic valued attribute value of concerned subject and object and proposed attribute values for the object.

Functional Specification

ABAC_α functional specification has six operations: access an object by a subject, creation of subject and object, deletion of subject, modification of subject and object attributes. Each ABAC_α operation has two parts: condition part and update part. Table 2.4 shows the specification of condition and update parts for ABAC_α operations.

2.2 The $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ Model

Usage control (UCON) was introduced by Park and Sandhu [103]. The family of UCON$_{ABC}$ models integrates Authorization(A), oBligation(B) and Conditions(C). UCON covers continuity (ongoing control) and mutability along with authorization, obligation and conditions. Among the family of UCON$_{ABC}$ models UCON$_{preA}$ covers pre-authorization of access, which is the most common mode of access control.

2.2.1 The $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ Model (Review)

In usage control authorization model entities are subjects and objects, and subjects are a subset of objects. Each object has a unique identifier and a finite set of attributes. Attributes can be mutable or immutable. Usage control Pre-Authorization model ($\mathrm{UCON}_{\mathrm{preA}}$) evaluates authorization decisions of permission prior to the execution of commands. Figure 2.2 shows the components of

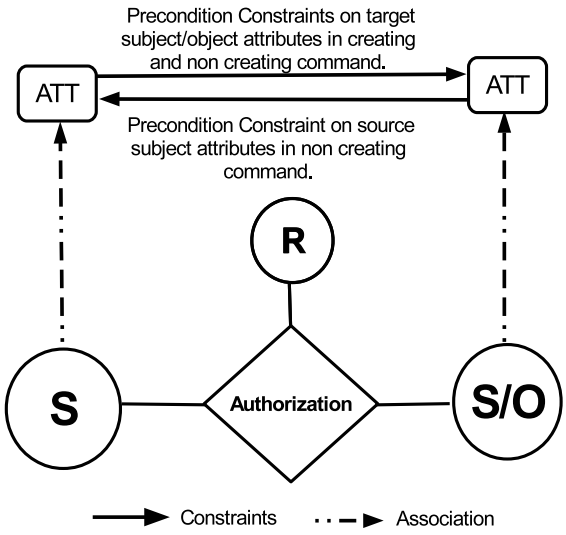


Figure 2.2: $\mathrm{UCON}_{\mathrm{preA}}$ Model.

$\mathrm{UCON}_{\mathrm{preA}}$ model.

The $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ model, i.e., pre-authorization UCON with finite attributes, is defined through a usage control scheme [107], as follows.

1. Object schema OS_{Δ}, is of the form $\{\mathrm{a}_1\colon \sigma_1, \ldots, \mathrm{a}_n\colon \sigma_n\}$ where each a_i is the name of an attribute and σ_i is a finite set specifying a_i's domain. $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ considers single object schema for different objects and considers only atomic values for each domain σ_i.

2. UR = $\{\mathrm{r}_1, \mathrm{r}_2, \ldots\ \mathrm{r}_k\}$, a set of usage rights, where r_i defines a permission enabled by a usage control command.

3. UC = $\{\mathrm{UC}_1, \mathrm{UC}_2, \ldots\ \mathrm{UC}_l\}$, a set of usage control commands.

4. ATT =$\{\mathrm{a}_1, \mathrm{a}_2, \ldots\ \mathrm{a}_n\}$, a finite set of object attributes.

5. AVT = $\sigma_1 \times \ldots \times \sigma_n$, set of all possible attribute value tuples.

6. avtf: $\mathrm{O} \rightarrow$ AVT, returns existing attribute value tuple of an object.

7. Each command in UC is associated with a right and has two formal parameters s and o, where s is a subject trying to access object o with right r. A single right can be associated

with more than one command. Number of commands (l) $\geq$ number of rights (k). There are two types of usage control commands, Non-Creating Command and Creating Command. Each command has a precondition part and an update part. Table 2.5 shows the structure of non-creating and creating command of $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$.

(a) In $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ non-creating command, f_b(s,o) is a boolean function which takes the attribute values of s and o and returns true or false. If the result is true then the PreUpdate is performed with zero or more attributes of s and o independently updated to new values computed from their attribute values prior to the command execution. Also the usage right r is granted. Otherwise the command terminates without granting r. f_1 and f_2 are the computing functions for new values.

(b) In $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ creating command, f_b(s) is a boolean function which takes the attribute values of s and returns true or false. If the result is true then the PreUpdate is performed with creating object o and zero or more attributes of s will be updated to new values computed from the attribute values of s. All attributes of the newly created object o are assigned computed attribute values. Also the usage right r is granted. Otherwise the command terminates without granting r. f_1 and f_2 are the computing functions for new values.

(c) In $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ deleting command, $f_b(\mathrm{o}_1,\mathrm{o}_2)$ is a boolean function which takes the attribute values of o_1 and o_2 and returns true or false. If the result is true then the PreUpdate is performed with deleting object o_2 and zero or more attributes of o_1 will be updated to new values computed from the attribute values of o_1 and o_2. Also the usage right r is granted. Otherwise the command terminates without granting r. f_1 is the computing functions for new values of o_1.

Table 2.5: $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ Command Structure

Non-Creating Command	Creating Command	Deleting Command

Command_Name$_r$(s,o)
PreCondition: f_b(s,o)$\rightarrow$ {true,false};
PreUpdate: s.a_{i_1} := $f_{1,a_{i_1}}$(s,o);
$\vdots$
s.a_{i_p} := $f_{1,a_{i_p}}$(s,o);
o.a_{j_1} := $f_{2,a_{j_1}}$(s,o);
$\vdots$
o.a_{j_q} := $f_{2,a_{j_q}}$(s,o);
	Command_Name$_r$(s,o)
	PreCondition:f_b(s)$\rightarrow$ {true,false};
	PreUpdate: create o;
	s.a_{i_1} := $f_{1,a_{i_1}}$(s);
	$\vdots$
	s.a_{i_p} := $f_{1,a_{i_p}}$(s);
	o.a_{j_1} := $f_{2,a_{j_1}}$(s);
	$\vdots$
	o.a_{j_q} := $f_{2,a_{j_q}}$(s);
		Command_Name$_r$(o_1,o_2)
		PreCondition:f_b(o_1,o_2)$\rightarrow$ {true,false};
		PreUpdate: delete o_2;
		o_1.a_{i_1} := $f_{1,a_{i_1}}$(o_1,o_2);
		$\vdots$
		o_1.a_{i_p} := $f_{1,a_{i_p}}$(o_1,o_2);

2.3 ReBAC Models

As OSNs have gained popularity, several ReBAC models have been introduced to capture various authorization policies. More recently, researchers have proposed extended ReBAC models applicable to other computing systems beyond OSNs. In this section, we review these two types of ReBAC models. These models are particularly relevant for Chapter 3.

2.3.1 ReBAC for Online Social Networks

Fong et al. [68] presented a Facebook-like access control model, featuring four types of policies that cover four different aspects of access in OSNs. The four policies include user search, traversal of the social graph, communication between users and normal access to objects owned by users. The policy vocabulary supports expressing some topology-based properties, such as k common friends and k clique. The model uses single relationship types between users.

Carminati et al. [43] proposed an access control model which considers type, depth and trust metrics of user-to-user relationship between accessing user and target user. It also considers multiple types of relationships between users. In [41], Carminati et al. proposed a model which utilizes semantic web technology. This model considers multiple type relationships between users and resources.

Cheng et al. [52] proposed a user-to-user relationship based access control model with a regular expression-based policy specification language. Social graph of UURAC contains user-to-user

relationships only. The connection between resources and users are referred to as controlling user (e.g., owner, tagged user). URRAC model [51] extends UURAC to include user-to-resource and resource-to-resource relations. In both models, multiple types of relationships are supported, and policy language can specify relationship path patterns between accessing user and target resource or user.

Subsequently Cheng et al. [53] defined an attribute-aware ReBAC model to express the contextual nature of relationships and users. The authors have extended their earlier UURAC model to incorporate node attributes and relationship attributes. They further introduced the concept of a graph attribute such as count which is associated with the relationship graph other than with a particular node or edge.

Bennett et al. [31] proposed a ReBAC model that considers multiple types of relationships between users and demonstrates how conflicts and potential misconfigurations can be automatically detected using the Alloy Analyzer [1]. Pang et al. [101] proposed an access control scheme for OSN where they have taken hybrid logic approach to use public information along with relationships.

2.3.2 ReBAC Beyond Online Social Environment

Fong et al. [67] proposed a formal ReBAC model intending to widen the application of ReBAC beyond social computing. The model considers multiple relationship types between users with directional relationships and access contexts, and uses a modal logic language for policy specification. The connection between users and resources is maintained through a system function called 'resource owner.' Fong et al. [69] extended the policy language of [67] and characterized its expressiveness. Subsequently they defined hybrid logic for ReBAC which can express complex relationship requirements [39].

Crampton et al. [56] proposed the RPPM model that can be applied to general computing system. The model considers users, resources and other logical and physical entities (i.e., files, folders, organizations, etc.) as nodes of a labeled relationship graph. Policies are defined using

path conditions. The model allows multiple types of relationship between different entities. The model uses a two-stage decision process: it first computes the path between requester and the requested resource and tries to find matches from a list of policies, and then it determines whether those policies are authorized. Rizvi et al. [110] demonstrated an implementation of RPPM model in an open-source medical record system. Subsequently they extended their model to be interoperable with RBAC [109]. Recently Crampton et al. [58] proposed a framework for inter-operating multiple ReBAC model instances by initiating request in one system to target resource in a second system.

Most ReBAC models consider user-to-user and possibly user-to-resource relationships. Very few of consider resource-to-resource relationships. Models that consider resource-to-resource relationships typically do so through users. Recently Ahmed et al. [19] proposed a ReBAC model which considers object-to-object relationships without intervening users, and demonstrated an implementation of the model in the OpenStack's [9] object storage, Swift [13]. This is further discussed in Chapter 5.

All the models reviewed so far are operational models. Recently a number of ReBAC administrative models have also been proposed for general purpose ReBAC [50, 57, 127] which consider graph dynamics such as adding/deleting nodes (entities) and or edges (relationships). In particular, [50] introduces the concept of dependent edge in ReBAC and considering dependencies during edge deletion.

2.4 Expressive Power Comparison Framework

Expressive power comparison is a fundamental problem in access control and has been extensively studied in the literature [24, 25, 47, 93, 99, 117, 119, 122, 131, 132]. Expressive power of an access control model is more precisely the expressive power of the schemes of that model, which captures the notion of policies that can be represented in systems based on that model's schemes [131]. Tripunitara and Li [131] defined a formal framework for comparing expressive power of access control models which is based on simulations that preserve security properties. In their framework

they presented two types of simulations: reductions and state matching reductions. They showed that state matching reductions are necessary and sufficient for preserving compositional security properties. A state matching reduction from scheme A to scheme B ensures that all states of scheme A can be realized in scheme B. Conversely a state matching reduction from scheme B to scheme A ensures that all states of scheme B can be realized in scheme A. If the former is true, then scheme B is at least as expressive as scheme A. If both the former and latter are true, schemes A and B are equivalent in expressive power. This notion of equivalence is also called bisimulation in the broader computer science literature.

An access control model is defined to be a set of access control schemes, i.e., the schemes expressible using the constructs of the model. An access control scheme is a state transition system $\langle \Gamma, \Psi, Q, \vdash \rangle$, where Γ is a set of states, Ψ is a set of state transition rules, Q is a set of queries and $\vdash$: $\Gamma \times$ Q $\rightarrow \{true, false\}$ is the entailment relation. The notion of state-matching reduction and expressive power equivalency are formally defined as follows.

Definition 1. State Matching Reduction:

Given two schemes A and B and a mapping A to B, $\sigma : (\Gamma^A \times \Psi^A) \cup Q^A \rightarrow (\Gamma^B \times \Psi^B) \cup Q^B$, we say that the two states γ^A and γ^B are equivalent under the mapping σ when for every $q^A \in Q^A, \gamma^A \vdash^A q^A$ if and only if $\gamma^B \vdash^B \sigma(q^A)$. A mapping σ from A to B is said to be a state-matching reduction if for every $\gamma^A \in \Gamma^A$ and every $\psi^A \in \Psi^A, \langle \gamma^B, \psi^B \rangle = \sigma(\langle \gamma^A, \psi^A \rangle)$ the following properties hold.

1. For every γ_1^A in scheme A such that $\gamma^A \xrightarrow{*}_{\psi} \gamma_1^A$, there exists a state γ_1^B such that $\gamma^B \xrightarrow{*}_{\psi} \gamma_1^B$ and γ_1^A and γ_1^B are equivalent under σ.

2. For every γ_1^B in scheme B such that $\gamma^B \xrightarrow{*}_{\psi} \gamma_1^B$, there exists a state γ_1^A such that $\gamma^A \xrightarrow{*}_{\psi} \gamma_1^A$ and γ_1^B and γ_1^A are equivalent under σ.

Definition 2. Expressive Power Equivalency of Access Control Models:

Given two access control models $\mathcal{M}$ and $\mathcal{M}'$, if for every scheme in $\mathcal{M}$ there exists a state-matching reduction from it to a scheme in $\mathcal{M}'$, and vice versa, then we say that $\mathcal{M}$ and $\mathcal{M}'$ are equivalent in expressive power.

In order to show that a reduction from model $\mathcal{A}$ to model $\mathcal{B}$ is state matching, we have to show the following:

1. For each scheme in $\mathcal{A}$, construct a mapping σ^A that maps A to a scheme B in $\mathcal{B}$.
2. Prove the σ^A mapping from A to B satisfies the following two requirements for a state-matching reduction:
   (a) For every state γ_1^A reachable from γ^A under the mapping σ^A there exists a reachable state in B scheme that is equivalent (answers all the queries in the same way).
   (b) For every state γ_1^B reachable from γ^B under the mapping σ^A there exists a reachable state in A scheme that is equivalent (answers all the queries in the same way).

In order to show that models $\mathcal{A}$ and $\mathcal{B}$ are equivalent in expressive power, we have to show:

- There exists a state matching reduction from $\mathcal{A}$ to $\mathcal{B}$.
- There exists a state matching reduction from $\mathcal{B}$ to $\mathcal{A}$.

2.5 The Openstack Cloud Platform

OpenStack [9] is a free and open-source software platform for cloud computing. It is mostly deployed as infrastructure-as-a-service (IaaS). Openstack makes virtual servers and other resources available to customers. The software platform consists of interrelated components that control diverse, multi-vendor hardware pools of processing, storage, and networking resources throughout a data center. Users either manage it through a web-based dashboard, through command-line tools, or through RESTful web services. In this section, we briefly describe Openstack object storage Swift. Specifically, we focus on the access control architecture of Swift. The implementation described in Chapter 5 utilizes Swift.

2.5.1 Swift Storage Structure

Swift is a highly available, distributed, eventually consistent object/blob store. Organizations can use Swift to store lots of data efficiently, safely, and cheaply [13]. Swift users use RESTful API [14] to upload or download objects to and from Swift object storage. Inside Swift, a project is assigned as an account. The account holds containers. Containers are similar to directories, however containers cannot be nested. A user associated with a Swift account can have multiple containers. To manage accounts, containers and objects Swift uses account servers, container servers and object servers accordingly.

Swift Authorization for Object Access

In OpenStack object storage Swift authorization (request to an object access) is currently done by Access Control List (ACL). Swift has two levels of ACL: Account Level ACL and container level ACL [15]. Container Level ACL is associated with containers in terms of read (download any object of that container) or write (upload an object in the container) or list [35]. Account ACLs allow users to grant account level access to other users. The limitations of Swift authorizations are:

- It cannot express object level ACL. To specify object level ACL every object needs to be stored in a separate container.
- It cannot give user access to a particular object if the user is not a member of the account/project.
- It doesn't support multicloud resource sharing.

Chapter 3: COMPARISON OF REBAC AND ABAC

Attribute-based access control (ABAC) expresses authorization policy via attributes while relationship-based access control (ReBAC) does so via relationships. While ABAC concepts have been around for a long time, ReBAC is relatively recent emerging with its essential application in online social networks. Even as ABAC and ReBAC continue to evolve, there are conflicting claims in the literature regarding their comparison. It has been argued that ABAC can subsume ReBAC since attributes can encode relationships. Conversely there are claims that the multilevel (or indirect) relations of ReBAC bring fundamentally new capabilities. So far there is no rigorous comparative study of ABAC vis a vis ReBAC. This chapter presents a comparative analysis of ABAC and ReBAC, and shows how various ReBAC features can be realized with different types of ABAC. We first identify several attribute types such as entity/non-entity and structured attributes that significantly influence ABAC or ReBAC expressiveness. We then develop a family of ReBAC models and a separate family of ABAC models based on the identified attribute types, with the goal of comparing the expressive power of these two model families. Further, we identify different dynamics of the models that are crucial for model comparison. We also consider different solutions for representing multilevel relationships with attributes. Finally, the ABAC and ReBAC model families are compared in terms of relative expressiveness and performance implications.

3.1 Attribute Types

In our comparison and classification for ReBAC and ABAC models, attributes play an important role. In this section we identify and discuss various types of attributes based on several different criteria. Some of these attribute types are crucial for ABAC and ReBAC comparison as their existence in a model strongly influences its expressiveness and performance. Others are not quite significant for our comparison purpose. In the next two sections, we use these attribute types to classify ReBAC and ABAC models to facilitate comparison between them.

We classify attribute types using five different criteria. Specifically the criteria are based on (1)

how attribute value(s) are structured, (2) what the attribute scope is, (3) boundedness of attribute range, (4) attribute association and (5) attribute mutability.

Depending upon the type of attribute value, there can be three types of attributes.

- **Atomic-valued or Single-valued Attribute:** If an attribute has at most one value associated with it at any one point in time, it is called atomic-valued or single-valued attribute [11, 82]. For example, gender attribute can have only a single value at a given time.

- **Set-valued or Multi-valued Attribute:** If an attribute can have more than one value associated with it at any one point in time, it is called set-valued or multi-valued attribute. For example, a person can have more than one phone number [11, 82].

- **Structured Attribute**: A structured attribute has a number of single or multi-valued sub-attributes [12]. For example, a Person-Info attribute can have sub-attributes of name, age and phoneNumber.

Depending upon the scope of the attribute, an attribute can either be an Entity Attribute or Non-entity Attribute.

- **Entity Attribute:** An entity is a thing which can be distinctly identified. A specific person, company an object or event is an example of entity [49]. Entity attribute takes an entity as input and returns another entity, a set of entities, or a structured tuple containing at least one entity. For example, an attribute value of parent of a person, owner of an object or friend of a person is another person, i.e., another entity.

- **Non-entity Attribute:** An attribute whose range is not defined on the set of entities in the system is called a non-entity attribute. For example, user's age or gender does not include another entity as its value. The concept of non-entity attribute depends upon what is defined as entities in the system. For example, suppose roles or organizations are entities in a system, and the range of attributes "assigned-roles" and "worksAt" are a set of roles and a set of organizations, respectively. In that case both attributes are entity attributes. If roles and organizations are not defined as entities in the system, these are non-entity attributes.

Depending upon whether the range of an attribute is bounded or not, attributes can be either finite domain attribute or infinite domain attribute.

- **Finite Domain Attribute:** Range of this attribute type is a finite set of attribute value (e.g., gender, role).

- **Infinite Domain Attribute:** Range of this attribute type is a countably infinite set of attribute values (e.g, time). Particularly important for access control, is that entity attributes where new entities can be created without bound are infinite domain attributes. This is required to accommodate unbounded creation of subjects and objects, which is the typical assumption in most systems.

Considering the association of an attribute we can have two types of attributes [77, 81, 139]

- **Contextual or Environmental Attribute**: These attributes are independent and not associated with any specific user, subject, object or other entity in the system. They are global and associated with system. For example, $current\text{-}time$ is system-wide information and not associated with any specific entity [81]. Other examples include system status, network security level, and so on [77, 139].

- **Meta Attribute**: Meta attributes are attributes of an attribute. Unlike regular attributes that are associated with entities, meta attributes are associated with other attributes. For example a user is associated with a role and the role is associated with a task. Here, the role is an attribute, and the task is a meta attribute [81].

Considering the mutability of attributes there are two types of attribute [103].

- **Mutable Attribute:** Mutable attributes are changed as a consequence or side effect of users' access or activity.

- **Immutable Attribute:** Immutable attributes can be changed only by direct administrative activity of a user or administrator.

The notions of entity/non-entity, finite/infinite domain, atomic-valued/set-valued/structured attributes are important for ReBAC-ABAC comparison as they are key attribute types that will strongly influence expressibility of relationships between entities or configurability of relationship graph.

Unlike these key attribute types, contextual/environmental attribute is a special type of attribute, not related to entities. Meta attribute defines relationship between attributes. Mutability is special feature specified in usage control for consumable authorization. These type of attributes are not relevant to ReBAC-ABAC comparisons with respect to expressiveness or performance. In the following, we will further discuss the definitions of these key attribute types as well as some assumptions for the rest of this chapter.

Attribute Definitions for ReBAC and ABAC Comparison

For our ReBAC and ABAC comparison, we consider entity and non-entity, finite and infinite domain, atomic-valued, set-valued and structured attributes. Below, we define these key attribute types (except for single-valued, multi-valued and structured attributes which have been adequately defined above).

Definition 3. Entity Attribute: An attribute att_i is an entity attribute if

i. range of att_i is a set of entities (i.e. att_i: $\text{E}_j \rightarrow \text{E}_k$),

ii. range of att_i is a powerset of entities (i.e. att_i: $\text{E}_j \rightarrow 2^{\text{E}_k}$), or

iii. att_i is a structured attribute with at least one sub-attribute being an entity attribute.

For example, if user is defined as an entity in the system and best-friend is an atomic or set-valued attribute of user then best-friend is an entity attribute. At any instant each entity set is fixed but can change over time if the system allows entity changes (i.e., creation or deletion of entities.). If att_i is a structured attribute and at least one sub-attribute of att_i is an entity attribute then att_i is also an entity attribute. For example let's say 'roleInfo(roles,assignedby)' is a structured attribute which has 'roles' and 'assignedby' as sub-attributes. Here 'roles' is non-entity attribute whose

range is set of roles however 'assignedby' is an entity attribute whose range is set of users. So 'roleInfo' is an entity attribute.

Definition 4. Non-Entity Attribute: An attribute att_i is a non-entity attribute if it is not an entity attribute.

Examples are phoneNumber and age. Note that if att_i is a structured attribute then every sub-attribute of att_i must be a non-entity attribute for att_i to be a non-entity attribute.

Definition 5. Finite Domain Attribute: An attribute domain is finite if the range of the attribute does not grow over time.

For example, 'gender' is a finite domain attribute. Also, 'roles' and 'security clearance' are finite domain attributes if the system does not allow new roles or security clearances to be added over time.

Definition 6. Infinite Domain Attribute: An attribute domain is infinite if the range of the attribute grows over time.

For example, in an OSN, if a new user can be created so he or she can be a friend of other users, the friend attribute is an infinite domain attribute as the range of friend is changed over time.

Finally, we introduce the familiar concept of attribute function composition [29, 71].

Definition 7. Attribute Function Composition: Nesting two or more attribute functions to form a single new function is known as attribute function composition. The composition of two attribute functions f : X $\rightarrow$ Y and g : Y $\rightarrow$ Z yields a function which maps x $\in$ X to g(f(x)) $\in$ Z. Composition is denoted as g $\circ$ f, where g is a function whose domain includes the range (or codomain) of f. We write (g $\circ$f)(x) as g(f(x)) for convenience.

A function $h(x) = f_n(.....f_2(f_1(x))....)$ which is the composition of n functions (same or different), say f_1 to f_n, is also said to be a composite function. Intuitively, composing two or more functions is a chaining process in which the output of the first function becomes the input of the second one, and the output of the (k-1)th function becomes the input of the kth function.

Assumptions

For ease of our comparison, all the ReBAC and ABAC models considered in this paper comply with the following assumptions.

1. *All non-entity attributes are finite domain.* Attributes such as role, department, title, gender, etc., typically admit only a small number of finite values by their intrinsic nature. Attributes such as location can be ever finer grained, so in principle could be regarded as infinite domain but a large finite domain should be adequate. Time being modeled as a finite domain has similar issue. For our purpose a finite domain assumption is reasonable.

2. *Each entity has a countably infinite set for all possible entities of that type.* For example if users, subjects and objects are the only entities defined in a particular system then the countably infinite sets for users, subjects and objects are $\mathcal{U}$, $\mathcal{S}$ and $\mathcal{O}$. The existing set of users, subjects and objects at any moment are U, S, O respectively where U, S, O are finite sets, and U $\subset \mathcal{U}$, S $\subset \mathcal{S}$ and O $\subset \mathcal{O}$.

3. *Identity of an entity is not reusable*. If an entity gets deleted, its identity cannot be used for another entity that is created after the deletion.

4. *All entity attribute functions are partial functions defined on existing entities only*. For example let $\mathcal{U}$ is the countably infinite set of all possible users, and U the finite set of current users (U $\subset \mathcal{U}$). An entity attribute function $f : U \rightarrow Y$ is defined only for elements of U and is undefined for elements in $\mathcal{U}$-U. We understand $f : U \rightarrow Y$ for an entity set U to mean that U will change with time but is finite at any moment. Note that if the system allows creation of entities then the entity attributes have infinite or unbounded domain. If the system doesn't allow any entity creation or deletion then the entity attributes form a finite domain.

5. *For attribute function composition inner attribute functions should always be entity attributes*. We require that a non-entity attribute can only occur as the outermost function

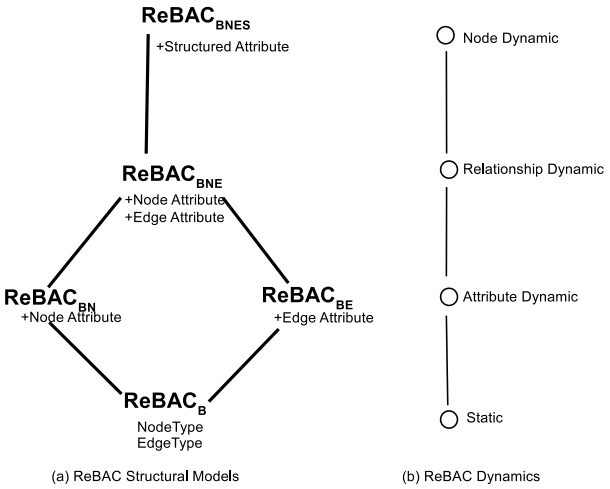


Figure 3.1: ReBAC Framework

in a composition. So for a composition $f_n(.....f_2\ (f_1(x))....)$, for 1 $\leq$ i$\leq$ n-1, f_i must be an entity attribute function, while f_n can be either entity or non-entity attribute.

6. For any set valued attribute function f defined on set X, we understand $f(X)$ to mean $\bigcup_{x_i \in X} f(x_i)$. So an attribute function composition friend(friend("Alice")) means $\bigcup_{u_i \in friend(\text{“}Alice\text{”})} friend(u_i)$

7. We understand that structured attribute is a multivalued tuple of atomic and or set-valued attributes. So it is more expressive than atomic or set valued attributes. Structured attribute can express atomic or set-valued attribute by having a single sub-attribute.

3.2 ReBAC Classification

In this section we develop a ReBAC framework including a family of structural models. The framework is illustrated in Figure 3.1 and consists of two components. Specifically, Figure 3.1(a) shows a family of structural models while Figure 3.1(b) shows the different types of dynamics found in ReBAC models.

The goal of this framework is to build a classification of ReBAC models that facilitates comparison with ABAC models. While there are many sophisticated proposals on ReBAC policy expression mechanisms such as incoming versus outgoing policy, policy individualization, modal/hybrid/

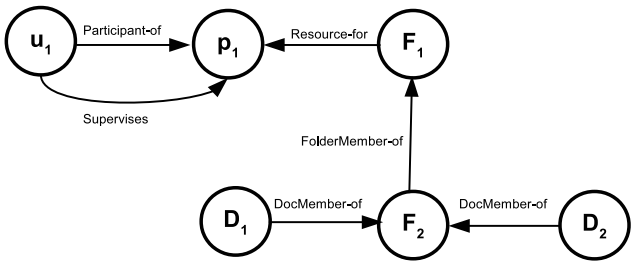


Figure 3.2: An Example of a Relationship Graph Expressible in ReBAC$_B$ [56]

first order/propositional logic based policies, this framework does not focus on policy specification. Rather it is independent of policy languages and focusses on structural and dynamic aspects of ReBAC.

Figure 3.1(a) depicts ReBAC models in a Hasse diagram with increasing capabilities as we go upwards in this partial order. In ReBAC, entities are represented as nodes in a relationship graph, and relations as entity to entity edges. We use the terms "node" and "entity" as synonyms, and likewise for the terms "edge" and "relation". The base model ReBAC$_{\mathrm{B}}$ allows for multiple node types (e.g., user, resource project, organization, group, etc.) and multiple directed or undirected edge types (e.g., friend, coworker, spouse, parent, etc.). Figure 3.2 shows an example relationship graph [56] expressible in ReBAC$_{\mathrm{B}}$. Most of the relationship graphs permitted in existing ReBAC models, including [51, 52, 56, 67, 68], can be expresed with the capabilities of ReBAC$_{\mathrm{B}}$.

ReBAC$_{\mathrm{BN}}$ adds node attributes to ReBAC$_{\mathrm{B}}$. Node attributes enable consideration of entity attributes along with relationships in authorization policies. For example, in a professional social network we may have a policy that an employee of an organization o_1 can connect to a recruiter of organization o_2 only if the recruiter is not already connected to any employees of o_1. In this case, the organization attribute of users (nodes) needs to be considered along with professional relationships. Another example is an online dating site where a single male user wants to connect a single female who has less than 4^{th} degree connection with him through only his female friends and is at least two years younger than him. Here we need to consider gender, age and relationship depth along with relationships. Such attribute-aware ReBAC is discussed in greater detail in [53]. Figure 3.3 shows an example relationship graph with node attributes.

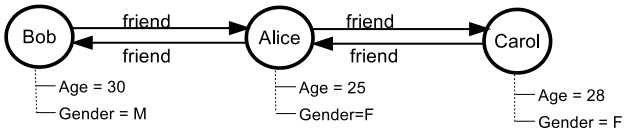


Figure 3.3: An Example of Node Attributes in Relationship Graph Expressible in ReBAC$_{BN}$

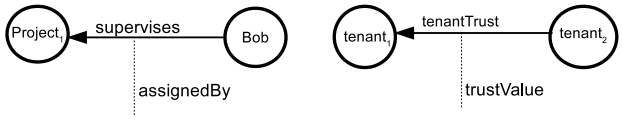


Figure 3.4: An Example of Edge Attributes in Relationship Graph Expressible in ReBAC$_{BE}$

ReBAC$_{\text{BE}}$ extends ReBAC$_{\text{B}}$ with edge attributes. For example, some ReBAC models use trust value of relationships to show the connection strength between users [42, 43]. In general, when a ReBAC authorization policy needs to consider some properties of relationships beyond relationship types, the relationship graph needs edge attributes to store and express those criteria, such as proposed in [53]. Figure 3.4 provides an example of edge attributes in a relationship graph. Here "Bob" is assigned to supervise "Project$_1$" and "assignedBy" is an edge attribute for relationship type "supervises" which specifies who has assigned "Bob" as supervisor. Similarly "tenant$_1$" has "tenantTrust" relationship with "tenant$_2$" and here "trustValue" specifies the strength of how much "tenant$_2$" trusts "tenant$_1$".

ReBAC$_{\text{BNE}}$ brings together the two separately motivated extensions of ReBAC$_{\text{BN}}$ and ReBAC$_{\text{BE}}$, such as in [53]. Following common practice, node and edge attributes in these models are atomic or set-valued attributes.

Recently Cheng et al. [50] proposed a ReBAC administrative model where they introduced the concept of dependent edge in relationship graph. A dependent edge example of MT-RBAC [50] is shown in Figure 3.5. Here user u owned by tenant x (with relationship type UO) can be "assigned to" a role r (with relation type UA) which is "owned by" tenant y (with relationship type RO) only if tenant y trusts tenant x (with relationship type TT). This particular tenant-trust relationship needs to be considered during role assignment or any time the trust-relationship between x and y changes. If the tenant's trust relationship is revoked at some point of time, the role assignment needs to be revoked as well. In order to configure this scenario using attributes, we need to store a paired set

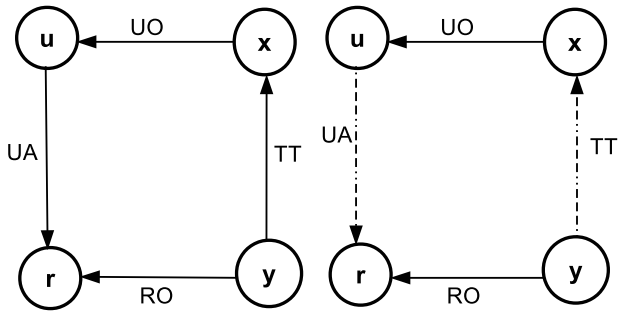


Figure 3.5: Example of Dependent Edge Expressible in ReBAC$_{BNES}$ [50]

of the role values and the required trust relationship. This additional information allows the model to consider cascading revocation [33, 62, 75] of dependent edges. This edge dependency in a graph cannot be captured using edge types or atomic or set valued edge attributes. To be precise, we will need structured attributes which can store multiple relevant attributes as a single attribute in a certain structure. For the above scenario, the structured attribute can store information of those edges that are required to create another edge. For instance, "dependsOn" attribute of relationship type UA can store a tuple of three sub-attributes: (sourceNode, targetNode, relationshipType), hence, (y,x,TT) for the example above. Consider another example where "securityLabel" is an object attribute. If a graph needs to store the information who has assigned a particular "securityLabel" to an object, we can use a structured attribute where sub-attributes are (label, assignedBy). If relationship graph only considers atomic or set valued attributes it won't be able to store this information. Our final model ReBAC$_{\text{BNES}}$ considers structured attributes for both nodes and edges. This completes our discussion of Figure 3.1(a).

Considering the changes or dynamism in ReBAC there are 4 dynamics shown in Figure 3.1(b). The dynamics are as follows.

- **Static:** In a static ReBAC model, attribute values, nodes and edges of the graph remain unchanged. A static graph is used for access only. Actions such as add or delete relationship between two entities (add or delete edges in the relationship graph), add or delete entities (add or delete nodes in relationship graph are not allowed) and change of attribute values are not allowed.

- **Attribute Dynamic**: ReBAC that allows changes of node attribute and edge attribute values are attribute dynamic ReBAC. For example, consider Hobby is a node attribute of users in a social network. Suppose Hobby("Alice")={gardening, painting}. Recently "Alice" gets interested to do "knitting" and wants to change her hobby in the social network site. If the system allows her to update her hobby as Hobby("Alice") = {gardening, painting, knitting} then it is an attribute dynamic ReBAC.

- **Relationship Dynamic**: ReBAC that allows changes of relationships between entities (add or delete edges in the relationship graph) is called relationship dynamic. Examples include establishing a new relationship between two entities, or deleting an existing relationship between two entities. We consider relationship dynamic to include attribute dynamic, since for ReBAC models which have edge attributes adding a new relationship requires assignment of attribute values of that edge.

- **Node Dynamic**: ReBAC that allows changes of entities is called as node dynamic ReBAC. Some examples are creating or deleting a user or resource in a relationship graph. Here we consider node creation implies possible relationship establishment and attribute value assignments when ReBAC models have attributes for nodes and or edges. Hence, node dynamic includes attribute dynamic (for some cases) and relationship dynamic.

Each ReBAC dynamic can be combined with any of the ReBAC structural models excluding ReBAC_B. ReBAC_B can only have static, relationship dynamic and node dynamic. However ReBAC_B cannot have attribute dynamic as it doesn't have any attributes. Thus, attribute dynamism is irrelevant for ReBAC_B.

3.3 ABAC Classification

In this section, we develop a set of structural models for ABAC with capabilities to configure the ReBAC models defined in Section 3.2. We define the ABAC models by considering attribute types that are necessary to capture relationships and relationship graphs as shown in Figure 3.6(a).

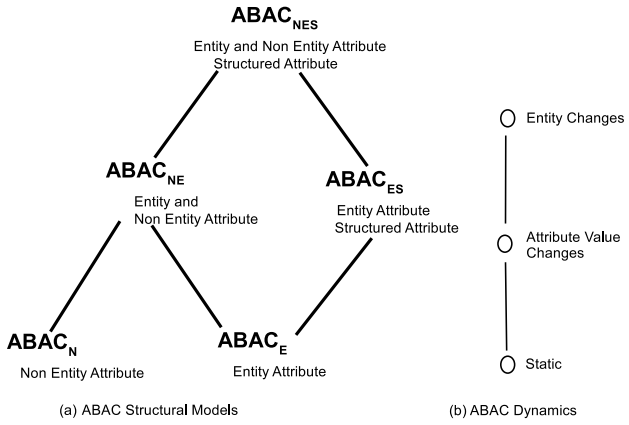


Figure 3.6: ABAC Framework

Specifically, we consider entity and non-entity, finite and infinite domain, and atomic-valued, set-valued and structured attributes. As shown in Figure 3.6(b), we also identify the dynamics of ABAC models. While this is not the most general framework for ABAC, it facilitates comparative analysis of relative expressiveness of ABAC and ReBAC.

Figure 3.6(a) depicts ABAC models with increasing capabilities as we go upwards in this Hasse diagram. $ABAC_N$ considers non-entity attributes only. According to our assumption 5 in Section 3.1, non-entity attribute cannot configure relationship composition, hence $ABAC_N$ is incomparable to $ReBAC_B$. $ABAC_N$ can only have attributes such as name, gender, location etc.

$ABAC_E$ considers entity attributes only and can configure $ReBAC_B$ model which has multiple relationship types and multiple entity types. Most of the ReBAC models fall under this category [51, 52, 56, 67, 68]. For example, consider the system graph in Figure 3.2. To configure it with $ABAC_E$ we need the following.

- entity types = {user, project, file, directory}
- user attributes = {Participant-of, Supervises},
  file attributes ={Resource-for, FileMember-of},
  project attributes = { },
  directory attributes ={DirectoryMember-of}.

$ABAC_{NE}$ considers both entity and non-entity attributes which is similar to considering node

attributes along with multiple relationship types and multiple entity types as in ReBAC_{BN}. For example, in Figure 3.2, suppose the user has attributes {name, gender, age} and files have attributes {securityLabel, size}. Using ABAC_{NE} we can configure these node attributes with non-entity attributes.

ABAC_{ES} considers structured entity attributes which can configure relationships and edge attributes of relationship graph. Figure 3.4 shows some simple edge attributes in relationship graphs. To configure the relationship graph "Bob supervises Project_1" in ABAC, we need to have entity attribute "supervises" for user so we can express supervises (Bob)= {"Project_1"}. In addition, to express the edge attribute "assignedBy", we will need a structured attribute of user "assignedBy", so we can express assignedBy(Bob) = ("Project_1","supervises", "Alice"). Here the sub-attributes for "assignedBy" are (targetNode, relationshipType, assignedByUser). The same is true for the tenantTrust relationship between tenant_1 and tenant_2. Here we can configure the trustValue with structured attribute trustValue (tenant_2) = ("tenant_1", 'tenantTrust', 0.5). Consider the example in Figure 3.5 where the edge (u, r, UA) is dependent on edge (y, x, TT). This dependency can be represented using a structured attribute for edge. To configure this structured edge attribute in ABAC, we need to have dependentEdge(u) = ("r","UA", {(y,x,TT)}).

ABAC_{NES} considers entity and non-entity structured attributes which can configure relationships, node attributes and edge attributes. This completes our discussion of Figure 3.6(a).

There are three types of ABAC systems in terms of possible changes, which we call ABAC dynamics. Figure 3.6(b) shows the dynamics as follows.

- **Static ABAC**: Nothing gets changed. In this type of ABAC, everything is static. Change of attribute values (i.e., assigning new values to attributes) or change of entities (i.e., adding or deleting entities) are not allowed.

- **Attribute Value Changes**: This ABAC allows changes of attribute values (assigning new values to attributes).

- **Entity Changes**: This ABAC allows new entity creation and/or deletion. We understand

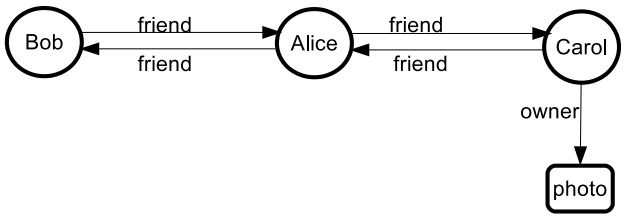


Figure 3.7: Relationship Graph for Example 1

that entity changes also includes attribute value changes as it needs assigning new values to attributes.

Each ABAC model shown in the Figure 3.6(a) can be combined with any dynamics shown in Figure 3.6(b).

3.4 Expressing MultiLevel Relationships With Attributes

Entity attributes can directly configure one-level relationship such as parent, spouse, owner. Only entity attribute is allowed for attribute function composition. ReBAC is all about expressing authorization policy with multilevel or composite relationship (friend ∘ friend, friend ∘ parent etc.). In this subsection, we propose two methods of composite relationship expression using attributes.

1. **Attribute Composition or Chaining:** Attribute chaining is exemplified by attribute function composition as defined in Section 3.1. Traditional ABAC uses direct attribute value of a user to specify policy, while attribute chaining approach allows to specify a policy through composition of attribute functions. This approach requires runtime computation for relationship composition just like ReBAC.

2. **Composite Attribute:** In this approach, all possible or required paths of a relationship graph are captured as attributes. When an update occurs in the relationship graph, this approach needs to update attributes of directly and indirectly related entities. Here the terms possible and required are used in the sense that the maximum possible depth of a graph depends upon its size while required depth means the limited depth required to specify authorization policy.

We discuss both concepts with some examples below.

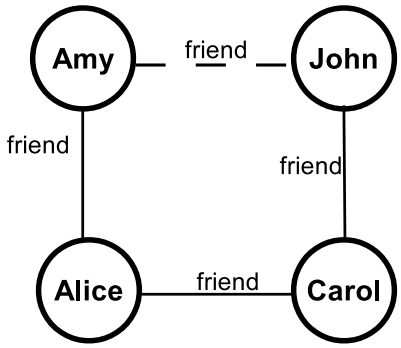


Figure 3.8: Relationship Graph for Example 2

Example 1: Consider the relationship graph in Figure 3.7. Let's assume the policy for photo only allows access by the owner or owner's friend.

Attribute Composition or Chaining : To configure this scenario with attribute composition approach, each user should have two entity attributes "friend" and "owner" and the authorization policy would check whether a particular user is in owner("photo") or friend(owner("photo")). According to this policy "Carol" and "Alice" can access "photo", but "Bob" cannot. If "friend" relationship between "Alice" and "Bob" is removed, it is necessary to update friend("Bob") and friend ("Alice").

Composite Attribute: In this approach, to express the relationship graph and policy, ABAC should have user attributes, "friend" and "friendOfFriend", as well as object (i.e., photo) attributes, "owner", "friendOfOwner" and "friendOfFriendOfOwner". Here, "friendOfFriend", "friendOfOwner" and "friendOfFriendOfOwner" are composite attributes. The authorization policy would check whether a particular user is in owner("photo") or friendOfOwner("photo"). Here, owner ("photo") = {"Carol"}, friendOfOwner("photo") = {"Alice"}, friendOfFriendOfOwner("photo") = {"Bob"}, friendOfFriend("Bob") = {"Carol"}, friendOfFriend("Carol") = { "Bob"}. If "friend" relationship between "Alice" and "Bob" is removed, it is necessary to update friend("Bob"), friend ("Alice") and friendOfFriend("Bob"). This action also requires indirect updates on friendOfFriend("Carol") and friendOfFriendOfOwner("photo").

Example 2: Consider Figure 3.8 where "Alice" has friends "Carol" and "Amy". "Amy" and "Carol" both have a common friend "John". So "John" is Alice's friend ∘ friend through "Carol"

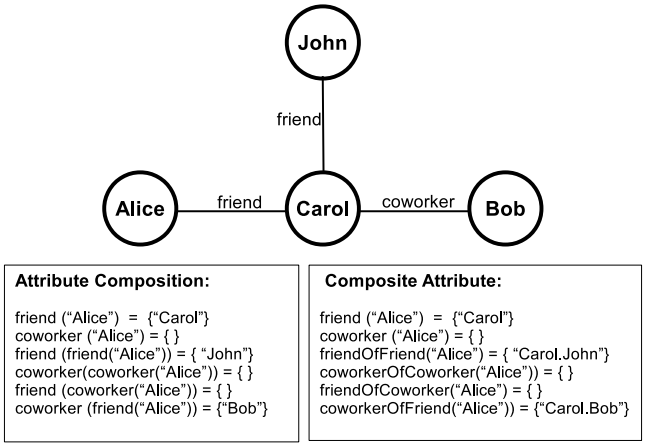


Figure 3.9: Attribute Composition and Composite Attribute for the Relationship Graph of Example 3

and “Amy”. Removing the relationship between “Amy” and “John” shouldn’t remove “John” from “Alice”’s friendOfFriend list. This means, instead of simply storing friendOfFriend(“Alice”) = {“John”}, we need to store friendOfFriend(“Alice”) = { “Amy.John”, “Carol.John”}. Storing such path information as an attribute value would ensure availability of accurate attribute values. As demonstrated in this example, it is often not sufficient to store only the end user information as an attribute value in case composite attributes are used.

Example 3: Consider another example with the simple relationship graph shown in Figure 3.9.

Attribute Composition or Chaining: In this approach we need to have two entity attributes for users, “friend” and “coworker”. To express a policy that verifies a composite relationship such as friend ∘ friend, coworker ∘ friend or friend ∘ coworker, we can use attribute composition such as friend(friend(“Alice”))= {“John”}, coworker(friend(“Alice”)) = {Bob}, and friend(coworker(“Bob”)) = {“John”}.

Composite Attribute: In this approach, we need to have “friend”, “coworker”, “friendOfFriend”, “friendOfCoworker”, “coworkerOfFriend” as attributes, so we can express relationship paths that might be found in policies without the use of chaining attributes. This approach has maximum depth limit in expressing relationship based policy dependent on the attribute configuration. Every entity attributes defined in this approach should have a fixed relationship depth. For example “friend” and “coworker” express one level relationships while “friendOfFriend”, “friendOf-

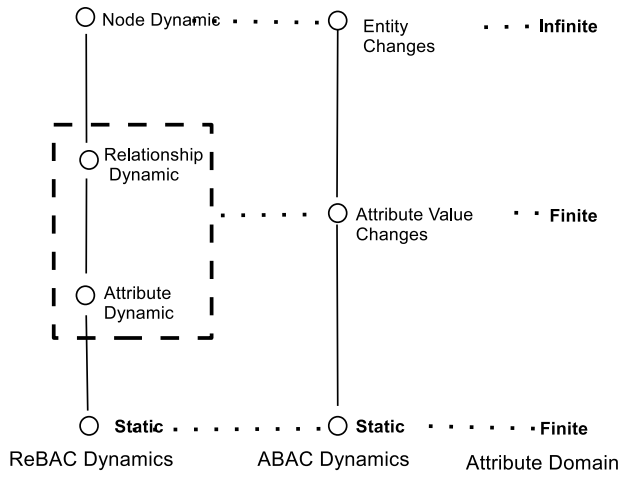


Figure 3.10: Comparison Between ReBAC and ABAC with respect to Dynamics and Attribute Domain

Coworker" and "coworkerOfFriend" express two level relationships.

3.5 Comparison: ABAC vs. ReBAC

In this section we compare ReBAC with ABAC, using the classifications of Sections 3.2 and 3.3. We conduct a conceptual comparison using two metrics: i) dynamics and ii) structural models. As the goal of this paper is to provide high level comparison, we do not provide any formally defined models or policy specifications. In order to use the formal framework of [131] to compare expressive power it is necessary to give detailed formal specifications of access control models. This limits comparison results to the very specific models that have been fully specified. We rather seek an intuitive but rigorous and insightful comparison between structurally comparable models.

In this work, we assume only entity attributes can configure relationships and non-entity attributes are finite domain attributes. We have shown that multilevel relationships can be configured with either attribute composition or with composite attributes. ReBAC node attributes can be configured using ABAC atomic or set-valued, and entity or non-entity attributes. ReBAC edge attributes can be configured using ABAC structured attributes of entities. From ReBAC point of view, if ABAC has only non-entity attributes, it means ReBAC graph structure has disconnected nodes with node attributes only. If ABAC has the capability to define entity attributes, it can be

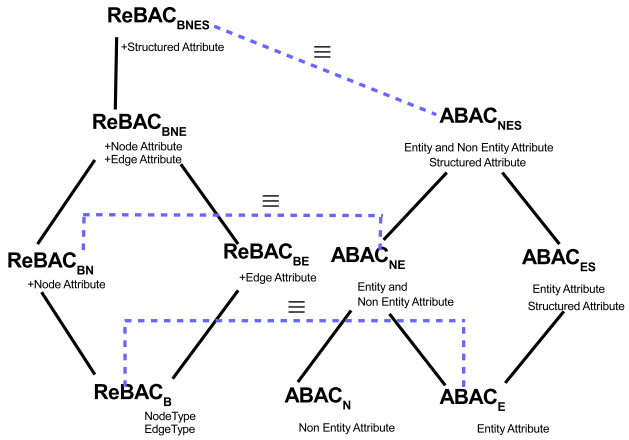


Figure 3.11: Equivalence of ReBAC and ABAC Structural Classification

configured to express relationships. Structured entity attributes can be configured as atomic or set-valued edge attributes or structured node attributes in relationship graph.

3.5.1 Comparison on Dynamics

Figure 3.10 shows a three-way alignment of ReBAC and ABAC dynamics with finite/infinite attribute domains. We understand this alignment to mean the following. The statement that ABAC_X is equivalent to ReBAC_Y is to be interpreted as given below.

- Static and finite attribute domain ABAC_X is equivalent to static ReBAC_Y.
- ABAC_X that allows change of attribute values with finite domain attribute is equivalent to relationship dynamic (which includes attribute dynamic where it is applicable) ReBAC_Y.
- ABAC_X that allows entity changes and infinite domain entity attribute is equivalent to node dynamic ReBAC_Y.

This alignment and interpretation allows us to avoid explicit consideration of all combinations of dynamics and models, which would be overwhelming. It does impose an obligation to consider all three levels of dynamics from Figure 3.10 in making equivalence claims.

We also have the following general result.

Theorem 1. *Finite domain ABAC cannot configure ReBAC that changes entities in the relationship*

graph (i.e., node dynamic ReBAC).

Proof. (Sketch) Entity changes in ReBAC entail creating new entities in the system and deleting existing ones. In order to configure any kind of ReBAC we need entity attributes in ABAC. Changes of entity from ReBAC requires changing the range of entity attribute for ABAC to potentially unbounded size. A finite domain ABAC cannot have attributes that changes its range over time in this manner. □

3.5.2 Comparable Structural Models for ReBAC and ABAC

In this sub-section we compare the ReBAC and ABAC structural models from Figures 3.1(a) and 3.6(a) respectively. Figure 3.11 shows the equivalence of different ABAC and ReBAC models (with blue dotted lines). Figure 3.12 shows the non-equivalence of different ABAC and ReBAC models (purple dotted line shows one model is incomparable with another while green dotted line shows one model is more expressive than another).

Theorem 2. *$ABAC_N$ is incomparable to $ReBAC_B$.*

Proof. (Sketch) ABAC_N has only non-entity attributes which cannot configure relations as discussed earlier. □

Theorem 3. *$ABAC_E$ and $ReBAC_B$ are equivalent in expressive power.*

Proof. (Sketch) To prove this we need to show

- ABAC_E can configure ReBAC_B
- ReBAC_B can configure ABAC_E

For the former, ABAC_E has entity attributes which can configure relationships via the techniques discussed in Section 3.4. For the latter, ABAC_E can be expressed as ReBAC_B where the entity attributes are relationship types and entities are nodes in the graph. □

Corollary 1. *$ABAC_N$ is incomparable to $ABAC_E$*

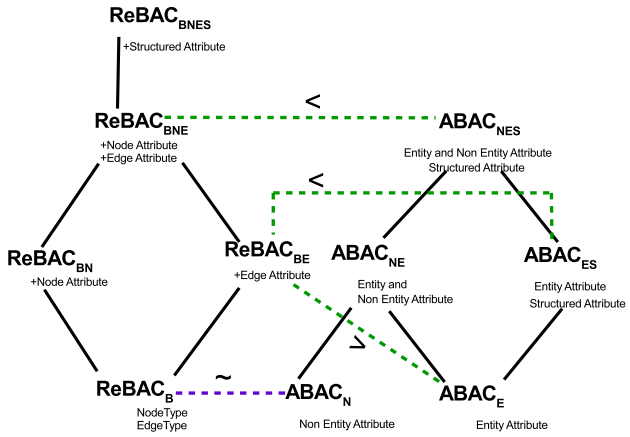


Figure 3.12: Non-Equivalence of ReBAC and ABAC Structural Classification

Proof. (Sketch) Theorem 2 proves that ABAC_N is incomparable to ReBAC_B and Theorem 3 proves that ABAC_E and ReBAC_B are equivalent in expressive power. The corollary follows. □

Theorem 4. *$ABAC_{NE}$ and $ReBAC_{BN}$ have equivalent expressive power*

Proof. (Sketch) With entity attributes ABAC_NE can configure relationships of ReBAC_BN and with non-entity attributes ABAC_NE can configure non-entity node attribute of ReBAC_BN. So ABAC_NE can configure ReBAC_BN. Conversely ReBAC_BN can express entity attributes as relationships and non-entity attributes as node attributes in the relationship graph. So ReBAC_BN can configure ABAC_NE. □

Theorem 5. *$ABAC_E$ is less expressive than $ReBAC_{BE}$*

Proof. (Sketch) Entity attributes of ABAC_E can be configured with relationships of ReBAC_BE. So ReBAC_BE can configure ABAC_E. On the other hand we have seen in Section 3.3 that structured attributes are required to configure edge attributes in ABAC. For example consider Figure 3.4 where “tenantTrust" has “trustValue" as edge attribute. Without structured entity attribute, ABAC_E cannot configure this example of ReBAC_BE. □

Theorem 6. *$ABAC_{ES}$ is more expressive than $ReBAC_{BE}$*

Proof. (Sketch) By definition ABAC_ES has structured entity attributes while ReBAC_BE does not have structured attributes. We have seen in section 3.3 with structured valued entity attribute

$ABAC_{ES}$ can configure relationships, nodes and atomic or set-valued edge attributes of $ReBAC_{BE}$. So $ABAC_{ES}$ can configure $ReBAC_{BE}$. On the other hand $ReBAC_{BE}$ cannot configure more than one-level structured entity attributes because it can have only atomic or set valued edge attribute. A 2-level structured entity attribute means at least one subattribute is also a structured attribute. So $ReBAC_{BE}$ cannot configure $ABAC_{ES}$. □

Theorem 7. *$ABAC_{NES}$ is more expressive than $ReBAC_{BNE}$*

Proof. (Sketch) Essentially similar proof as the previous theorem. □

Theorem 8. *$ABAC_{NES}$ and $ReBAC_{BNES}$ have same expressive power*

Proof. (Sketch) $ABAC_{NES}$ has structured entity and non-entity attributes while $ReBAC_{BNES}$ has labeled relationship graph (multiple types of relationships) with multiple types of nodes (entities) and structured node and edge attributes. Section 3.3 has shown that $ABAC_{NES}$ can configure relationships, nodes and structured attributes for nodes and edges. So $ABAC_{NES}$ can configure $ReBAC_{BNES}$. On the other hand $ReBAC_{BNES}$ can configure entities with nodes, structured entity and non-entity attributes with structured entity and non-entity node attributes respectively. So $ReBAC_{BNES}$ can configure $ABAC_{NES}$. This proves that $ABAC_{NES}$ and $ReBAC_{BNES}$ have same expressive power. □

3.5.3 Performance Comparison

So far we have considered the theoretical expressive power equivalence between ABAC and ReBAC. There are clearly some differences between them in terms of performance. ReBAC does runtime computation of authorization. Even if relationship graph is static and nothing changes, ReBAC still needs to repeat the same computation. To eliminate this massive redundant computation load researchers have considered caching of relationship paths [55]. In Section 3.4 we proposed two solutions for multilevel relationship expression in ABAC, viz., attribute composition and composite attributes. Attribute composition is similar to ReBAC in expressing policy, while composite attribute is more like caching of path information. Attribute composition has polynomial

complexity for authorization policy and constant complexity for update, on the other hand composite attribute has constant complexity in policy authorization and polynomial time complexity on update to maintain relationship changes.

Performance also depends upon the characteristics of the system. A number of variances regarding system characteristics such as relationship dynamics, node dynamics and density of relationships between nodes (entities) affect performance. For meaningful performance comparison we need to formally define specific comparable models considering both approach, do their implementation and configure the system for different dynamics (attribute dynamics, node dynamics, relationship dynamics and density dynamics).

3.5.4 Choices Of Models

Attribute composition or ReBAC approach puts the load on runtime computation, while caching or composite attribute may need significant update load. If relationship graph changes frequently, the caching or composite attribute approach needs to have excessive updates to keep the path information up-to-date.

The choice of models depends on node dynamics, relationship dynamics and the density of relationships between nodes (entities) in the system. If the relationship density of a system is high, adding or deleting a largely connected node will affect quite a large number of relationships in the system. For a static system or a system with non-entity attribute change, regardless of whether the graph is dense or sparse composite attribute is the best approach for relationship expression. If the system has huge node dynamics and relationship dynamics, and relationship density is also high attribute composition would be the best solution. If the system is in the middle between these two extremes then we can think of an hybrid approach where both attribute composition and composite attribute are used in the same model. For example to achieve p level relationship composition we can use m level composite attribute and n level attribute composition where $p = n \times m$. To specify it more clearly we can say that a composite attribute with 4 level relationship expression capability such as ffff(u) or an attribute composition with 4 level relationship expression capability

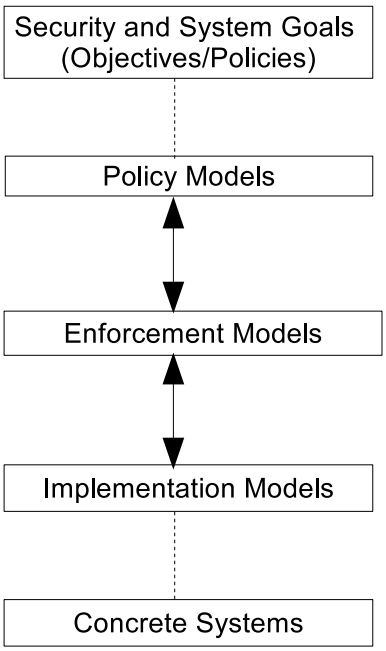


Figure 3.13: PEI Framework [114]

such as f(f(f(f(u)))) can be expressed with a composite attribute of 2 level relationship expression capability using 2 level attribute composition ff(ff(u)). This means ffff(u) = f(f(f(f(u)))) = ff(ff(u)).

Application context for security has the well established 3 layers (Policy P, Enforcement E and implementation I or PEI) [113, 114, 118], as shown in Figure 3.13. Policy level P is all about expressibility, modularity and convenience to express policy and independent of implementation detail. From expressibility point of view both the approaches are equal as we have already shown the equivalence of policy expression at the P layer. E layer is responsible for enforcement architecture wherein performance would come into consideration. Depending on the dynamics characteristics we conjecture that some hybrid combination of ABAC with attribute composition and composite attribute would be optimal for most situations.

Chapter 4: SAFETY AND EXPRESSIVE POWER OF ABAC_α AND ITS ENHANCEMENTS

This chapter analyzes the safety of an existing ABAC model, viz. ABAC_α [82], proposes two enhanced versions of ABAC_α, viz. $\mathrm{ABAC}_\alpha^{\mathrm{AM}}$ and $\mathrm{ABAC}_\alpha^{\mathrm{MI}}$, and analyzes the decidability boundary and comparative expressive power for these extensions. $\mathrm{ABAC}_\alpha^{\mathrm{AM}}$ maintains the safety decidability result, while $\mathrm{ABAC}_\alpha^{\mathrm{MI}}$ is Turing complete and thereby has undecidable safety. This results are developed by a comparative study with another existing attribute based access control model $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ [107]. Precise expressive power comparison of ABAC_α and $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ is left open.

Figure 4.1 summarizes the central results of this chapter.

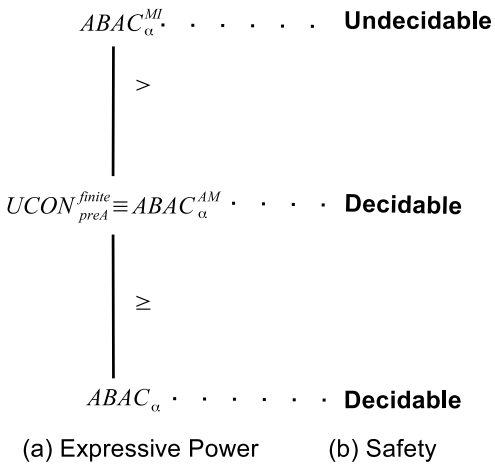


Figure 4.1: Comparison of ABAC_α and its Enhancements.

4.1 Safety of ABAC_α

Safety analysis is a fundamental problem for any access control model. Recently, it has been shown that the pre-authorization usage control model with finite attribute domains, viz. $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ has decidable safety [107]. ABAC_α is a pre-authorization model and requires finite attribute domains, but is otherwise quite different from $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$. This section gives a state-matching reduction from ABAC_α to $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$. The notion of state-matching reductions was defined by Tripunitara

and Li [131], as reductions that preserve security properties including safety. It follows that safety of ABAC_α is decidable. The following first presents a reduction from ABAC_α to $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ then proves that the reduction is state matching.

4.1.1 Reduction from ABAC_α to $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$

This subsection defines a reduction from ABAC_α to $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$. $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ PreConditions are command specific boolean functions while ABAC_α policies are boolean expressions. For attribute value update ABAC_α uses direct value from the range of the attribute while $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ uses value computing functions (see Table 2.5) to compute the value of the attributes. To relate the machinery of the two models we introduce some additional notations. One is policy evaluation functions and sets of eligible attribute value tuples for creation and modification of subjects and objects of ABAC_α. Another one is PreCondition evaluation functions of $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$. Policy evaluation functions pre-evaluates the policies for a specific set of existing and proposed attribute value tuples and return true or false. Set of eligible attribute value tuples are the authorized attribute value tuples to do a certain operation, e.g. create/modify a subject or create/modify an object. This reduction then defines a single $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ command for each element of eligible attribute value tuples set. PreCondition evaluation function pre-computes precondition for a specific set of existing attribute value, a specific command and a specific computed attribute value tuple, and returns true or false.

Policy Evaluation Functions for ABAC_α

Each Policy evaluation function evaluates corresponding policy and returns true or false.

Authorization Policy Evaluation Function: ChkAuth(p,$savt$, $oavt$) returns true or false. This function evaluates the authorization policy Authorization$_p$ to determine whether a subject with attribute value $savt$ is allowed to have permission p on an object with attribute value tuple $oavt$.

Creation and Modification Policy Evaluation Functions:

- ChkConstrSubCreatebyUser($uavt$,$savt$) returns true or false. It evaluates the subject creation policy ConstrSubCreatebyUser as to whether a user with attribute value tuple $uavt$ is

allowed to create a subject with attribute value tuple $savt$.

- ChkConstrSubModbyUser($uavt, savt_1, savt_2$) returns true or false. It evaluates the subject modification policy ConstrSubModbyUser as to whether a user with attribute value tuple $uavt$ is allowed to modify a subject with attribute value tuple $savt_1$ to $savt_2$.

- ChkConstrObjCreatebySub($savt, oavt$) returns true or false. It evaluates the object creation policy ConstrObjCreatebySub as to whether a subject with attribute value tuple $savt$ is allowed to create an object with attribute value tuple $oavt$.

- ChkConstrObjModbySub($savt, oavt_1, oavt_2$) returns true or false. It evaluates the object modification policy ConstrObjModbySub as to whether a subject with attribute value tuple $savt$ is allowed to modify an object with attribute value tuple $oavt_1$ to $oavt_2$.

Sets of Eligible Attribute Value Tuples

Using the policy evaluation functions for ABAC_α we define 4 eligible sets for attribute value tuples as follows.

Definition 8. Set of user-subject-creatable-tuples

UAVTCrSAVT $\subseteq$ UAVT $\times$ SAVT

UAVTCrSAVT = $\{\langle i, j\rangle \mid$ i $\in$ UAVT $\wedge$ j $\in$ SAVT

$\wedge$ ChkConstrSubCreatebyUser(i,j)$\}$

Definition 9. Set of user-subject-modifiable-tuples

UAVTModSAVT $\subseteq$ UAVT $\times$ SAVT $\times$ SAVT

UAVTModSAVT = $\{\langle i, j, k\rangle \mid$ i $\in$ UAVT $\wedge$ j$\in$ SAVT

$\wedge$ k $\in$ SAVT $\wedge$ ChkConstrSubModbyUser(i,j,k)$\}$

Definition 10. Set of subject-object-creatable-tuples

SAVTCrOAVT $\subseteq$ SAVT $\times$ OAVT

SAVTCrOAVT = $\{\langle i, j\rangle \mid$ i $\in$ SAVT $\wedge$ j $\in$ OAVT

$\wedge$ ChkConstrObjCreatebySub(i,j) $\}$

Definition 11. Set of subject-object-modifiable-tuples

$$\text{SAVTModOAVT} \subseteq \text{SAVT} \times \text{OAVT} \times \text{OAVT}$$

$$\text{SAVTModOAVT} = \{\langle i, j, k\rangle \mid \text{i} \in \text{SAVT} \wedge \text{j} \in \text{OAVT}$$

$$\wedge\ \text{k} \in \text{OAVT} \wedge \text{ChkConstrObjModbySub(i,j,k)}\}$$

PreCondition Evaluation Functions for $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$

PreCondition evaluation functions of $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ check the PreConditions of $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ commands and return true or false.

- **CheckPCNCR(**$uc_r, avt_1, avt_2, avt_3, avt_4$**)** returns true or false. It evaluates the PreCondition f_b and PreUpdate of non-creating command uc_r as to whether a subject with attribute value tuple avt_1 is allowed to execute command uc_r on an object with attribute value tuple avt_2 and if allowed whether it modifies subject's attribute value tuple from avt_1 to avt_3 and object's attribute value tuple from avt_2 to avt_4.

- **CheckPCCR(**$uc_r, avt_1, avt_2, avt_3$**)** returns true or false. It evaluates the PreCondition f_b and PreUpdate of creating command uc_r as to whether a subject with attribute value tuple avt_1 is allowed to execute the command uc with right r and if allowed whether it creates an object with attribute value tuple to avt_2 and modifies it's own attribute value tuple from avt_1 to avt_2.

Reduction from ABAC_α **to** $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$

The reduction is presented showing the configuration of $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ object schema, rights and commands to simulate ABAC_α. Table 4.1 shows the reduction.

Object Schema of $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$**:** Every ABAC_α entity (user, subject, object) is represented as a $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ object and the attribute entity_type specifies whether a particular $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ object is an ABAC_α user, subject or object. User, subject and object attributes of ABAC_α are represented as $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ object attributes. There is no user creation in ABAC_α so $\mathrm{U}^{\mathrm{ABAC}_\alpha}$ is a finite set. ABAC_α function SubCreator is configured here with a mandatory $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ object attribute whose domain is the finite set of users ($\mathrm{U}^{\mathrm{ABAC}_\alpha}$). To determine which user is the creator of an

Table 4.1: Reduction from ABAC_α to $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$

Object Schema(OS_Δ):
[entity_type:{user, subject, object}, user_name: U^{ABAC_α}, SubCreator: U^{ABAC_α}, isDeleted: {true,false}, ua_1:Range(ua_1), . . . , ua_m:Range(ua_m), sa_1:Range(sa_1), . . . , sa_n:Range(sa_n), oa_1:Range (oa_1), . . . , oa_p: Range (oa_p)]
Attributes:
ATT = {entity_type, user_name, SubCreator, isDeleted} $\cup$ $UA^{\mathrm{ABAC}_\alpha}$ $\cup$ $SA^{\mathrm{ABAC}_\alpha}$ $\cup$ $OA^{\mathrm{ABAC}_\alpha}$
Usage Rights:
UR= P^{ABAC_α} $\cup$ {d}
Commands:
$\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ commands are defined in Tables 4.2 and 4.3

ABAC_α subject, $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ object needs to have another mandatory attribute user_name whose range is also the finite set U^{ABAC_α}. ABAC_α has a subject deletion operation. In [107] it is shown that deletion of a subject can be simulated by using a special boolean attribute isDeleted which has a boolean domain. We consider "NULL" as a special attribute value for any atomic or set valued attribute. It is assigned to an attribute which is not appropriate for a particular entity. We need to add "NULL" in the range of UA, SA and OA for this reduction. As there is no user deletion and object deletion in ABAC_α, isDeleted would be "NULL" for both users and objects. The $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ attribute set ATT is {entity_type, user_name, SubCreator, isDeleted} $\cup$ $UA^{\mathrm{ABAC}_\alpha}$ $\cup$ $SA^{\mathrm{ABAC}_\alpha}$ $\cup$ $OA^{\mathrm{ABAC}_\alpha}$.

$\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ **usage rights UR:** In this reduction each ABAC_α permission is considered as a usage right in $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ and additionally a dummy right d is introduced. Each $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ command associates with a right. We use dummy right d for association with the commands which are defined to configure ABAC_α operations. The set of usage rights $UR^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ is thereby P^{ABAC_α} $\cup$ $\{d\}$.

$\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ **commands:** ABAC_α operations are reduced to specific $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ commands. A single ABAC_α operation requires definition of multiple $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ commands to account for different attribute value combinations. The reduction defines a creating command for each element of UAVTCrSAVT and SAVTCrOAVT, and a non-creating command for each element of UAVTModSAVT and SAVTModOAVT. For example consider an ABAC_α subject creation policy where a user u with attribute value tuple $uavt$ is allowed to create a subject s with attribute value tuple

Table 4.2: $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ Creating Commands

For each $\langle i, j \rangle \in$ UAVTCrSAVT **CreateSubjectbyUser_ij$_d$(s, o)** PreCondition: s.entity_type = user $\wedge \langle s.ua_1, \ldots, s.ua_m \rangle = \langle i_1, \ldots, i_m \rangle$ PreUpdate: create o o.entity_type = subject o.user_name = NULL o.SubCreator = s.user_name o.isDeleted = false o.ua$_1$ = NULL $\vdots$ o.ua$_m$ = NULL o.sa$_1$ = j$_1$ $\vdots$ o.sa$_n$ = j$_n$ o.oa$_1$ = NULL $\vdots$ o.oa$_p$ = NULL	For each $\langle i, j \rangle \in$ SAVTCrOAVT **CreateObjectbySubject_ij$_d$(s,o)** PreCondition: s.entity_type = subject $\wedge$ s.isDeleted = false $\wedge \langle s.sa_1, \ldots, s.sa_n \rangle = \langle i_1, \ldots, i_n \rangle$ PreUpdate: create o o.entity_type = object o.user_name = NULL o.SubCreator = NULL o.isDeleted = NULL o.ua$_1$ = NULL $\vdots$ o.ua$_m$ = NULL o.sa$_1$ = NULL $\vdots$ o.sa$_n$ = NULL o.oa$_1$ = j$_1$ $\vdots$ o.oa$_p$ = j$_p$

Table 4.3: $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ Non-Creating Commands

For each r $\in$ UR$^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \setminus$ {d} **Access$_r$(s, o)** PreCondition: ChkAuth(r,avtf(s),avtf(o)) PreUpdate: N/A	**DeleteSubjectbyUser$_d$(s, o)** PreCondition: s.entity_type =user $\wedge$ o.entity_type = subject $\wedge$ o.SubCreator = s.user_name $\wedge$ o.isDeleted = false PreUpdate: o.isDeleted = true
For each $\langle i, j, k \rangle \in$ UAVTModSAVT **ModifySubjectAttbyUser_ijk$_d$(s,o)** PreCondition: s.entity_type = user $\wedge$ o.entity_type = subject $\wedge$ o.isDeleted = false $\wedge$ o.SubCreator = s.user_name $\wedge \langle s.ua_1, \ldots, s.ua_m \rangle = \langle i_1, \ldots, i_m \rangle$ $\wedge \langle o.sa_1, \ldots, s.sa_n \rangle = \langle j_1, \ldots, j_n \rangle$ PreUpdate: o.sa$_1$ = k$_1$ $\vdots$ o.sa$_n$ = k$_n$	For each $\langle i, j, k \rangle \in$ SAVTModOAVT **ModifyObjectAttbySub_ijk$_d$(s,o)** PreCondition: s.entity_type = subject $\wedge$ o.entity_type = object $\wedge$ s.isDeleted = false $\wedge \langle s.sa_1, \ldots, s.sa_n \rangle = \langle i_1, \ldots, i_n \rangle$ $\wedge \langle o.oa_1, \ldots, s.oa_p \rangle = \langle j_1, \ldots, j_p \rangle$ PreUpdate: o.oa$_1$ = k$_1$ $\vdots$ o.oa$_p$ = k$_p$

$savt$, so by definition $\langle uavt, savt\rangle \in$ UAVTCrSAVT. For each element $\langle i, j\rangle \in$ UAVTCrSAVT this reduction has a command named CreateSubjectbyUser_ij(s, o) which creates an object o with entity_type = subject. There are no changes to the attributes of s while the attributes of the newly created o are set to the values in j. This is shown on the left hand side of Table 4.2. The right hand side of Table 4.2 similarly shows the $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ commands to simulate the ABAC_α object creation by subject operation. Turning to Table 4.3 the top left quadrant shows the $\mathrm{Access}_r^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}(s, o)$ commands, each of which simulates the ABAC_α command $\mathrm{Access}_p^{\mathrm{ABAC}_\alpha}(s, o)$ for r = p. There is no PreUpdate in these commands. The top right quadrant of Table 4.3 shows the simulation of the $\mathrm{DeleteSubject}^{\mathrm{ABAC}_\alpha}(u, s)$ by $\mathrm{DeleteSubject}_d^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}(s, o)$. The bottom half of Table 4.3 shows the reduction of the ABAC_α modify attribute commands, by user (left side) and by subject (right side). In both cases only the target's attributes are modified with user (left side) or subject (right side) attributes remaining unchanged as per ABAC_α semantics.

4.1.2 Safety of ABAC_α

In this subsection we show that safety of ABAC_α is decidable. We prove that the reduction provided in the previous subsection is state matching, so it preserves security properties including safety. Decidable safety for ABAC_α then follows from decidable safety for $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$. In order to show that a reduction from ABAC_α and $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ is state matching, we have to show the following:

1. Represent ABAC_α and $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ models as ABAC_α and $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ schemes
2. Construct a mapping $\sigma^{\mathrm{ABAC}_\alpha}$ that maps ABAC_α to $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$
3. Prove that $\sigma^{\mathrm{ABAC}_\alpha}$ mapping from ABAC_α to $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ satisfies the following two requirements for state matching reduction:
   (a) for every state $\gamma_1^{\mathrm{ABAC}_\alpha}$ reachable from $\gamma^{\mathrm{ABAC}_\alpha}$ under the mapping $\sigma^{\mathrm{ABAC}_\alpha}$ there exists a reachable state in $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ scheme that is equivalent (answers all the queries in the same way)

(b) for every state $\gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ reachable from $\gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ under the mapping $\sigma^{\mathrm{ABAC}_\alpha}$ there exists a reachable state in ABAC_α scheme that is equivalent (answers all the queries in the same way)

ABAC_α Scheme

An ABAC_α scheme consists of $\langle \Gamma^{\mathrm{ABAC}_\alpha}, \Psi^{\mathrm{ABAC}_\alpha}, Q^{\mathrm{ABAC}_\alpha}, \vdash^{\mathrm{ABAC}_\alpha} \rangle$. Where

- $\Gamma^{\mathrm{ABAC}_\alpha}$ is the set of all states. Where each state $\gamma^{\mathrm{ABAC}_\alpha} \in \Gamma^{\mathrm{ABAC}_\alpha}$ is characterized by $\langle$ U_γ, S_γ, O_γ, UA, SA, OA, uavtf, savtf, oavtf, P, SubCreator$\rangle$ where U_γ, S_γ, O_γ are set of users, subjects objects respectively in state γ.
- $\Psi^{\mathrm{ABAC}_\alpha}$ is the set of state transition rules which are all ABAC_α operations defined in Table 2.4.
- $\mathrm{Q}^{\mathrm{ABAC}_\alpha}$ is the set of queries of type:
  1. Authorization$_p(s, o)$ for p$\in \mathrm{P}^{\mathrm{ABAC}_\alpha}$, s$\in \mathrm{S}^{\mathrm{ABAC}_\alpha}$, o$\in \mathrm{O}^{\mathrm{ABAC}_\alpha}$.
  2. ConstrSubCreatebyUser$(u, s, savt)$ for $u \in \mathrm{U}^{\mathrm{ABAC}_\alpha}$, $s \notin \mathrm{S}^{\mathrm{ABAC}_\alpha}$, $savt \in \mathrm{SAVT}^{\mathrm{ABAC}_\alpha}$.
  3. ConstrSubModbyUser$(u, s, savt)$ for $u \in \mathrm{U}^{\mathrm{ABAC}_\alpha}$, $s \in \mathrm{S}^{\mathrm{ABAC}_\alpha}$, $savt \in \mathrm{SAVT}^{\mathrm{ABAC}_\alpha}$.
  4. ConstrObjCreatebySub$(s, o, oavt)$ for $s \in \mathrm{S}^{\mathrm{ABAC}_\alpha}$, $o \notin \mathrm{O}^{\mathrm{ABAC}_\alpha}$, $oavt \in \mathrm{OAVT}^{\mathrm{ABAC}_\alpha}$.
  5. ConstrObjModbySub$(s, o, oavt)$ for $s \in \mathrm{S}^{\mathrm{ABAC}_\alpha}$, $o \in \mathrm{O}^{\mathrm{ABAC}_\alpha}$, $oavt \in \mathrm{OAVT}^{\mathrm{ABAC}_\alpha}$.
- Entailment $\vdash$ specifies that given a state $\gamma \in \Gamma^{\mathrm{ABAC}_\alpha}$ and a query q $\in \mathrm{Q}^{\mathrm{ABAC}_\alpha}$, $\gamma \vdash$ q if and only if q returns true in state γ.

$\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ Scheme

An $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ scheme consists of $\langle \Gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}, \Psi^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}, Q^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}, \vdash^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \rangle$, as follows.

- $\Gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ is the set of all states. Where each state $\gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \in \Gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ is characterized by $\langle OS^\gamma_\Delta, UR, ATT, AVT, avtf \rangle$. Here $\mathrm{OS}^\gamma_\Delta$ is the object schema in state γ.

- $\Psi^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ is set of state transition rules which are the set of creating and non-creating commands of $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ defined in Tables 4.2 and 4.3 respectively.

- $\mathrm{Q}^{\mathrm{ABAC}_\alpha}$ is the set of queries and of following types:

  1. CheckPCNCR($uc_r, avtf(s), avtf(o), avt_3, avt_4$) for $uc_r \in$ UC, $r \in$ UR, s and o are $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ objects.

  2. CheckPCCR($uc_r, avtf(s), avt_2, avt_3$) for $uc_r \in$ UC, $r \in$ UR, s is an $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ object.

- Entailment $\vdash$ specifies that given a state $\gamma \in \Gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ and a query q $\in \mathrm{Q}^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$, $\gamma \vdash$ q if and only if q returns true in state γ.

Mapping from ABAC_α **to** $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ **(**$\sigma^{\mathrm{ABAC}_\alpha}$**)**

- Mapping of $\Gamma^{\mathrm{ABAC}_\alpha}$ to $\Gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$

  - Mapping of Object Schema(OS_Δ), ATT and UR is provided in Table 4.1

- Mapping of $\Psi^{\mathrm{ABAC}_\alpha}$ to $\Psi^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$

  - σ(Access$_p$) = Access$_r^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ where r = p.

  - σ(CreateSubjectbyUser($u, s, savt$)) = CreateSubjectbyUser_ij$_d$(s,o),

    i = uavtf(u) and j = $savt$.

  - σ(DeleteSubjectbyUser(u, s)) = DeleteSubjectbyUser$_d$(s,o).

  - σ(ModifySubjectAttbyUser($u, s, savt$)) = ModifySubjectAttbyUser_ijk$_d$(s,o),

    i = uavtf(u) and j = savtf(s) and k = $savt$.

  - σ(CreateObjectbySubject($s, o, oavt$)) = CreateObjectbySubject_ij$_d$(s,o),

    i = savtf(s) and j = $oavt$.

  - σ(ModifyObjectAttbySubject($s, o, oavt$)) = ModifyObjectAttbySubject_ijk$_d$(s,o),

    i = savtf(s) and j = oavtf(o) and k = $oavt$.

- Mapping of Q^{ABAC_α} to $Q^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ is provided below

  - σ(Authorization$_p(s, o)$) = CheckPCNCR(Access$_p$, avtf(s), avtf(o), avtf(s), avtf(o)).
  - σ(ConstrSubCreatebyUser($u, s, savt$)) = CheckPCCR(CreateSubjectbyUser_ij$_d$, avtf(s), avtf(s), $\langle$ subject, NULL, u, false, NULL,. . . , NULL, savt$_1$, . . . savt$_n$, NULL,. . . , NULL $\rangle$) where i = uavtf(u) and j = savt.
  - σ(ConstrSubModbyUser($u, s, savt$))= CheckPCNCR(ModifySubjectAttbyUser_ijk$_d$, avtf(s), avtf(o), avtf(s), $\langle$ savt$_1$, . . . savt$_n$ $\rangle$) where i = uavtf(u), j = savtf(s) and k = $savt$.
  - σ(ConstrObjCreatebySub($s, o, oavt$)) = CheckPCCR(CreateObjectbySubject_ij$_d$, avtf(s), o, avtf(s), $\langle$ object, NULL, NULL, NULL, NULL,. . . , NULL, NULL,. . . , NULL, oavt$_1$, . . . oavt$_p$ $\rangle$) where i = savtf(s) and j = oavt.
  - σ(ConstrObjModbySub($s, o, oavt$))= CheckPCNCR(ModifyObjectAttbySubject_ijk$_d$, avtf(s), avtf(o), avtf(s), $\langle$ oavt$_1$, . . . oavt$_p$ $\rangle$) where i = savtf(s), j = oavtf(o) and k = $oavt$.

Proof that $\sigma^{\mathrm{ABAC}_\alpha}$ is State-Matching

The proof that the mapping provided above is a state matching reduction is lengthy and tedious. Here we present an outline of the main argument.

Lemma 1. *$\sigma^{\mathrm{ABAC}_\alpha}$ satisfies assertion 1 of the state matching reduction of Definition 1.*

Proof. (Sketch): Assertion 1 requires that, for every $\gamma^{\mathrm{ABAC}_\alpha} \in \Gamma^{\mathrm{ABAC}_\alpha}$ and every $\psi^{\mathrm{ABAC}_\alpha} \in \Psi^{\mathrm{ABAC}_\alpha}$, $\langle\, \gamma^{\mathrm{ABAC}_\alpha}, \psi^{\mathrm{ABAC}_\alpha} \,\rangle = \sigma\ (\langle\, \gamma^{\mathrm{ABAC}_\alpha}, \psi^{\mathrm{ABAC}_\alpha} \,\rangle)$ has the following property:

For every $\gamma_1^{\mathrm{ABAC}_\alpha}$ in scheme ABAC_α such that
$\gamma^{\mathrm{ABAC}_\alpha} \xrightarrow{*}_{\psi^{\mathrm{ABAC}_\alpha}} \gamma_1^{\mathrm{ABAC}_\alpha}$,
there exists a state $\gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ such that

1. $\gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}(=\sigma(\gamma^{\mathrm{ABAC}_\alpha})) \xrightarrow{*}_{\psi^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}(=\sigma(\psi^{\mathrm{ABAC}_\alpha}))} \gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$.

2. for every query $q^{\mathrm{ABAC}_\alpha} \in Q^{\mathrm{ABAC}_\alpha}$, $\gamma_1^{\mathrm{ABAC}_\alpha} \vdash^{\mathrm{ABAC}_\alpha} q^{\mathrm{ABAC}_\alpha}$ if and only if $\gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \vdash^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \sigma(q^{\mathrm{ABAC}_\alpha})$. It can be decomposed into two directions:

   (a) The "if" direction:

   $\gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \vdash^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \sigma(q^{\mathrm{ABAC}_\alpha}) => \gamma_1^{\mathrm{ABAC}_\alpha} \vdash^{\mathrm{ABAC}_\alpha} q^{\mathrm{ABAC}_\alpha}$.

   (b) The "only if" direction:

   $\gamma_1^{\mathrm{ABAC}_\alpha} \vdash^{\mathrm{ABAC}_\alpha} q^{\mathrm{ABAC}_\alpha} => \gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \vdash^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \sigma(q^{\mathrm{ABAC}_\alpha})$.

The proof is by induction on number of steps n in $\gamma^{\mathrm{ABAC}_\alpha} \xrightarrow{*}_{\psi^{\mathrm{ABAC}_\alpha}} \gamma_1^{\mathrm{ABAC}_\alpha}$.

□

Lemma 2. *$\sigma^{\mathrm{ABAC}_\alpha}$ satisfies assertion 2 of the state matching reduction of definition 1.*

Proof. (Sketch): Assertion 2 requires that, for every $\gamma^{\mathrm{ABAC}_\alpha} \in \Gamma^{\mathrm{ABAC}_\alpha}$ and every $\psi^{\mathrm{ABAC}_\alpha} \in \Psi^{\mathrm{ABAC}_\alpha}$, $\langle\, \gamma^{\mathrm{ABAC}_\alpha}, \psi^{\mathrm{ABAC}_\alpha} \,\rangle = \sigma\,(\langle\, \gamma^{\mathrm{ABAC}_\alpha}, \psi^{\mathrm{ABAC}_\alpha} \rangle)$ has the following property:

For every $\gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ in scheme $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ such that $\gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}(=\sigma(\gamma^{\mathrm{ABAC}_\alpha})) \xrightarrow{*}_{\psi^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}(=\sigma(\psi^{\mathrm{ABAC}_\alpha}))} \gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$, there exists a state $\gamma_1^{\mathrm{ABAC}_\alpha}$ such that

1. $\gamma^{\mathrm{ABAC}_\alpha} \xrightarrow{*}_{\psi^{\mathrm{ABAC}_\alpha}} \gamma_1^{\mathrm{ABAC}_\alpha}$.

2. for every query $q^{\mathrm{ABAC}_\alpha} \in Q^{\mathrm{ABAC}_\alpha}$, $\gamma_1^{\mathrm{ABAC}_\alpha} \vdash^{\mathrm{ABAC}_\alpha} q^{\mathrm{ABAC}_\alpha}$

   if and only if $\gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \vdash^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \sigma(q^{\mathrm{ABAC}_\alpha})$.

   It can be decomposed into two directions:

   (a) The "if" direction:

   $\gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \vdash^{\mathrm{ABAC}_\alpha} \sigma(q^{\mathrm{ABAC}_\alpha}) => \gamma_1^{\mathrm{ABAC}_\alpha} \vdash^{\mathrm{ABAC}_\alpha} q^{\mathrm{ABAC}_\alpha}$.

   (b) The "only if" direction:

   $\gamma_1^{\mathrm{ABAC}_\alpha} \vdash^{\mathrm{ABAC}_\alpha} q^{\mathrm{ABAC}_\alpha} => \gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \vdash^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \sigma(q^{\mathrm{ABAC}_\alpha})$.

The proof is by induction on number of steps n in $\gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}(=\sigma(\gamma^{\mathrm{ABAC}_\alpha})) \xrightarrow{*}_{\psi^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}(=\sigma(\psi^{\mathrm{ABAC}_\alpha}))} \gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$.

□

Theorem 9. *$\sigma^{\mathrm{ABAC}_\alpha}$ is a state matching reduction.*

Proof. Lemma 3 shows that $\sigma^{\mathrm{ABAC}_\alpha}$ satisfies assertion 1 of Definition 1 and Lemma 4 shows that $\sigma^{\mathrm{ABAC}_\alpha}$ satisfies assertion 2 of Definition 1. Thereby, $\sigma^{\mathrm{ABAC}_\alpha}$ is a state matching reduction. □

Theorem 10. *Safety of ABAC_α is decidable.*

Proof. Safety of $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ is decidable [107]. Theorem 9 proved there exists a state matching reduction from ABAC_α to $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$. A state matching reduction preserves security properties including safety [131]. □

4.2 Safety and Expressive Power of a $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ Equivalent ABAC_α Enhancement

ABAC_α and $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ are both attribute based pre-authorization models which have finite attribute domain and unbounded creation of subjects and objects. However there are significant differences between them such as clear distinction between user and subject, subject capability to create and modify another subject or modify itself, attribute mutability, and strong coupling between authorization and update. This section proposes an enhancement of ABAC_α, viz., $\mathrm{ABAC}^{\mathrm{AM}}_\alpha$. It provides state-matching reductions from $\mathrm{ABAC}^{\mathrm{AM}}_\alpha$ to $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ and vice versa. Thereby, $\mathrm{ABAC}^{\mathrm{AM}}_\alpha$ and $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ are equivalent in expressive power and safety of $\mathrm{ABAC}^{\mathrm{AM}}_\alpha$ is also decidable.

4.2.1 ABAC$^{AM}_\alpha$ Model

In this subsection we define an extension of ABAC_α with subject attribute mutability during creation and modification of subjects, and capability for subjects to create and modify subjects (including self-modification). We name this extended model as $\mathrm{ABAC}^{\mathrm{AM}}_\alpha$. $\mathrm{ABAC}^{\mathrm{AM}}_\alpha$ has the same basic sets and functions as ABAC_α and we use the same notation for attribute value tuples. The main difference between ABAC_α and $\mathrm{ABAC}^{\mathrm{AM}}_\alpha$ are in their creation, modification and deletion policies, policy configuration points and operational functionalities. An ABAC_α subject can only

Table 4.4: $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ Formal Model

Basic Sets and Functions

U, S, O, UA, SA, OA, SCOPE, Range, UAVT, SAVT, OAVT, uavtf, savtf, oavtf are same as ABAC_{α}

Authorization Policy

- **Authorization on Object**
  Same as ABAC_{α}
- **Authorization on Subject**
  For each p$\in$ P, AuthorizationonSubject$_p$(s_1,s_2) returns true or false.
  Specified by LAuthorizationonSubject

Creation, Deletion and Modification Policy

Subject Creation Policy:

- **Subject Creation by User**
  Same as ABAC_{α}
- **Subject Creation by Subject**
  ConstrSubCreatebySub(s_1:S,s_2:S,savt$_1$:SAVT,savt$_2$:SAVT) returns true or false.
  Specified by LConstrSubCreatebySub

Subject Deletion Policy:

- **Subject Deletion by User**
  ConstrSubDelbyUser(u, s) returns true or false.
  Specified by LConstrSubDelbyUser
- **Subject Deletion by Subject**
  ConstrSubDelbySub(s_1, s_2, $savt$) returns true or false.
  Specified by LConstrSubDelbySubject

Subject Modification Policy:

- **Subject Modification by User (with mutability)**
  ConstrSubModbyUser(u:U,s:S, $uavt$:UAVT, $savt$:SAVT) returns true or false.
  Specified by LConstrSubModbyUser.
- **Subject Modification by Subject (with mutability)**
  ConstrSubModbySub(s_1:S,s_2:S,savt$_1$:SAVT, savt$_2$:SAVT) returns true or false.
  Specified by LConstrSubModbySub.

Object Creation Policy:

- **Object Creation by Subject**
  Same as ABAC_{α}

Object Modification Policy:

- **Object Modification by Subject**
  Same as ABAC_{α}

Policy Language

Each policy language is an instantiation of the Common Policy Language CPL (defined in Table 2.2) that varies only in the values it can compare. Table 4.5 specifies the set and $atomic$ instantiation of LAuthorizationonSubject, LConstrSubCreatebySub, LConstrSubDelbyUser, LConstrSubDelbySub, LConstrSubModbyUser, LConstrSubModbySub.

Functional Specification

$\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ operations are specified in Table 4.6

be created by a user and be modified only by the creating user. An $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ subject, in addition, can be created by another subject and can be modified by itself or another subject. The feature that a subject's own attributes are changed when that subject creates or modifies another subject, is called mutability. $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ introduces authorization on subject on top of ABAC_{α}'s authorization on objects. Table 4.4 provides the formal definition of $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$. In $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ formal model components U, S, O, UA, SA, OA, UAVT, SAVT, P, SubCreator have the same definition as ABAC_{α}. When a subject creates another subject, the creator of the creating subject is considered as the creator of new subject (which will be a user).

Authorization Policy

$\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ authorization policy consists of a single authorization policy on object for each permission and a single authorization policy on subject on each permission. Permissions and authorization policy on an object are same as ABAC_{α}. Each authorization policy on subject takes a permission and two subjects as input and returns true or false based on the boolean expression defined on the attributes of those subjects.

Creation, Modification and Deletion Policy

Subject creation, object creation and object modification policies are same as ABAC_{α} which has been defined in Chapter 2, Section 2.1.1. CPL is same as ABAC_{α} for all the languages defined in Table 2.2. Subject creation by subject, subject modification by user, and subject modify by subject policies, consider mutability with boolean expressions and defined using LConstrSubCreatebySub, LConstrSubModbyUser and LConstrSubModbySub respectively. LConstrSubCreatebySub is a CPL instantiation where set and $atomic$ refers to the set and atomic valued existing attribute value of creating subject and proposed attribute value for creating subject and subject to be created. LConstrSubModbyUser is a CPL instantiation where set and $atomic$ refers to the set and atomic valued existing and proposed attribute of concerned user and subject. LConstrSubModbySub is a CPL instantiation where set and $atomic$ refers to the set and atomic valued existing and proposed

Table 4.5: Definition of added Language for $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$

Language	*set*	*atomic*
LAuthorizationonSubject	setsa(s_1) | setsa(s_2)	atomicsa(s_1) | atomicsa(s_2)
LConstrSubCreatebySub	setsa(s_1) | setsa′(s_1) | setsa′(s_2)	atomicsa(s_1) | atomicsa′(s_1) | atomicsa′(s_2) |
LConstrSubDelbyUser	setua(u) | setsa(s)	atomicua(u) | atomicsa(s)
LConstrSubDelbySubject	setsa(s_1) | setsa′(s_1) | setsa(s_2)	atomicsa(s_1) | atomicsa′(s_1) | atomicsa(s_2)
LConstrSubModbyUser	setua(u) | setsa(s) | setua′(u) | setsa′(s)	atomicua(u) | atomicsa(s) | atomicua′(u) |atomicsa′(s)|
LConstrSubModbySub	setsa(s_1) | setsa(s_2)| setsa′(s_1) | setsa′(s_2)	atomicsa(s_1) | atomicsa(s_2)| atomicsa′(s_1) | atomicsa′(s_2)

Table 4.6: Functional Specification for $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$.

Operations	**Conditions**	**Updates**	**Change relative to** ABAC_{α}
Access$_p(s, o)$	same as ABAC_{α}	same as ABAC_{α}	same as ABAC_{α}
CreateSubjectbyUser			
CreateObjectbySubject			
ModifyObjectAttbySubject			
DeleteSubjectbyUser (u,s:NAME)	$s \in \mathrm{S} \wedge u \in \mathrm{U} \wedge$ SubCreator(s) = u $\wedge$ ConsSubDelbyUser(u,s)	Same as ABAC_{α}	Extended from ABAC_{α}
ModifySubjectAttbyUser (u,s,$uavt$:UAVT,$savt$:SAVT)	$s \in \mathrm{S} \wedge u \in \mathrm{U} \wedge$ SubCreator(s)=$u \wedge$ ConstrSubMutable(u,s,$uavt$,$savt$)	uavtf(u) = $uavt$ savtf(s) = $savt$	
AccessSubject$_p$(s_1,s_2)	AuthorizationOnSubject$_p$(s_1, s_2)		Newly added
CreateSubjectbySubject(s_1, s_2:NAME,$savt_1$:SAVT, $savt_2$:SAVT)	ConstrSubCreatebySub(s_1,s_2, $savt_1$, $savt_2$)$\wedge$ $s_1 \in \mathrm{S} \wedge s_2 \notin \mathrm{S}$	S′ = S $\cup$ s_2 SubCreator(s_2) = SubCreator(s_1) savtf(s_1) =$savt_1$ savtf(s_2) = $savt_2$	
DeleteSubjectbySubject (s_1,s_2:NAME,$savt$)	$s_1 \in \mathrm{S} \wedge s_2 \in \mathrm{S} \wedge$ SubCreator(s_1) = SubCreator(s_2) $\wedge$ ConsSubDelbySubject(s_1,s_2, $savt$)	S′ = S$\backslash\{s_2\}$ savtf(s_1)= $savt$	
ModifySubjectAttbySubject (s_1,s_2, $savt_1$:SAVT, $savt_2$:SAVT)	$s_1 \in \mathrm{S} \wedge s_2 \in \mathrm{S} \wedge$ ConstrSubModbySub(s_1,s_2, $savt_1$, $savt_2$)$\wedge$ SubCreator(s_1) = SubCreator(s_2)	savtf(s_1) = $savt_1$ savtf(s_2) = $savt_2$	

attribute of concerned subjects. Subject deletion by user and subject deletion by subject is defined using LConstrSubDelbyUser and LConstrSubDelbySub respectively. LConstrSubDelbyUser is a CPL instantiation where set and $atomic$ refers to the set and atomic valued existing attributes of user and subject. LConstrSubDelbySub is a CPL instantiation where set and $atomic$ refers to the set and atomic valued existing attributes of concerned subjects and set or atomic valued proposed attributes of deleting subject.

Table 4.5 shows the set and $atomic$ instantiation for LAuthorizationonSubject, LConstrSubModbyUser, LConstrSubCreateBySub, LConstrSubModbySub, LConstrSubDelbyUser, LConstrSubDelbySub.

Functional Specification

$\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ functional specifications has 10 operations: access a subject or an object by a subject, creation of subject by user or another subject, modification of subject attributes by user or another subject or by itself, deletion of subject by user or by another subject, creation of object and modification of object attributes by subject. Table 4.6 gives the functional specifications of $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$. Here access an object by a subject, creation of subject by user, creation or modification of object by subject are same as ABAC_{α}. Modification of subject by user, deletion of subject by user are extended from ABAC_{α}. Subject access, creation, modification and deletion by subject are newly added operations on top of ABAC_{α}.

4.2.2 Reductions

In this subsection we define two reductions: 1) $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ to $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ and 2) $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ to $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$.

Reduction from $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ to $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$:

This construction is similar to the construction provided in subsection 4.1.1. $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ PreConditions are command specific boolean functions while $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ policies are boolean expressions. For attribute value update $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ uses direct value from the range of the attribute while $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ uses value computing functions (see Table 2.5) to compute the value of the attributes. Here we need to introduce similar notations defined in 4.1.1 to relate the machinery of both the models. Here policy evaluation functions pre-evaluates the boolean expression of different $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ policies for a specific set of existing and proposed attribute value tuples and returns true or false. It then construct eligible sets of tuples for creation and modification of subjects and objects and deletion of subjects for which the policy evaluation function is true. For each element in the eligible set this construction defines a corresponding $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ command. It also defines PreCondition evaluation functions which pre-compute the PreConditions and attribute value computing functions (f_1, f_2) for a specific set of attribute value tuples and for a specific $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$

command and returns true or false. These additional notations and data structures enable us to relate the machinery of these two models.

Policy Evaluation Functions:

Each Policy evaluation function evaluates corresponding policy and returns true or false.

Authorization Policy Evaluation Functions:

- ChkAuth($savt$, $oavt$) checks for pre-computed policy subject access object and returns true or false.
- ChkAuthSub($savt_1$, $savt_2$) checks the for pre computed policy subject access subject and returns true or false.

Creation and Modification Policy Evaluation Functions:

- ChkConstrSubCreatebyUser($uavt$,$savt$) checks for pre-computed policy subject creation by user and returns true or false.
- ChkConstrSubCreatebySub($savt_1$, $savt_2$, $savt_3$) checks for pre-computed policy subject creation by subject and returns true or false.
- ChkConstrSubDelbyUser($uavt$, $savt$) checks for pre computed policy subject deletion by user and returns true or false.
- ChkConstrSubDelbySub($savt_1, savt_2, savt_1$) checks for pre computed policy subject deletion by subject and returns true or false.
- ChkConstrSubModbyUser($uavt_1$, $savt_1$, $uavt_2$, $savt_2$) checks for pre computed policy subject attribute modification by user and returns true or false.
- ChkConstrSubModbySub($savt_1$),$savt_2$), $savt_3$, $savt_4$)checks for pre computed policy subject attribute modification by subject and returns true or false.
- ChkConstrObjCreatebySub($savt$, $oavt$) checks for pre computed policy object creation by subject and returns true or false.

Table 4.7: Reduction from $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ to $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$

Object Schema($\mathbf{OS}_{\triangle}$):
[entity_type:{user, subject, object}, user_name: $\mathrm{U}^{\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}}$, SubCreator: $\mathrm{U}^{\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}}$, ua_1:Range(ua_1), . . . , ua_m:Range(ua_m), sa_1:Range(sa_1), . . . , sa_n:Range(sa_n), oa_1:Range (oa_1), . . . , oa_p: Range (oa_p)]
Attributes:
ATT = {entity_type, user_name, SubCreator} $\cup \mathrm{UA}^{\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}} \cup \mathrm{SA}^{\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}} \cup \mathrm{OA}^{\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}}$
Usage Rights:
UR= $\mathrm{P}^{\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}} \cup \{\mathrm{d}\}$
Commands:
$\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ commands are defined in Table 4.8, 4.9 and 4.10

- ChkConstrobjModbySub($savt_1, oavt_1, oavt_2$) checks for pre computed policy object attribute modification by subject and returns true or false.

Sets of Eligible Attribute Value Tuples

Using the policy evaluation functions for ABAC_{α} we define 4 eligible sets for attribute value tuples as follows.

Definition 12. Set of user-subject-creatable-tuples

UAVTCrSAVT $\subseteq$ UAVT $\times$ SAVT

UAVTCrSAVT ={ $\langle i, j \rangle$ | i $\in$ UAVT $\wedge$ j $\in$ SAVT

$\wedge$ ChkConstrSubCreatebyUser(i,j)}

Definition 13. Set of subject-subject-creatable-tuples

SAVTCrSAVT $\subseteq$ SAVT $\times$ SAVT $\times$ SAVT

SAVTCrSAVT ={ $\langle i, j, k \rangle$ | i $\in$ SAVT $\wedge$ j $\in$ SAVT

$\wedge$ k $\in$ SAVT $\wedge$ ChkConstrSubCreatebySub(i,j,k)}

Definition 14. Set of user-subject-deletable-tuples

UAVTDelSAVT $\subseteq$ UAVT $\times$ SAVT

UAVTDelSAVT ={ $\langle i, j \rangle$ | i $\in$ UAVT $\wedge$ j $\in$ SAVT

$\wedge$ ChkConstrSubDelbyUser(i,j)}

Definition 15. Set of subject-subject-deletable-tuples

SAVTDelSAVT $\subseteq$ SAVT $\times$ SAVT $\times$ SAVT

SAVTDelSAVT =$\{\langle i, j, k\rangle$ | i $\in$ SAVT $\wedge$ j $\in$ SAVT

$\wedge$ k $\in$ SAVT $\wedge$ ChkConstrSubDelbySub(i,j,k)$\}$

Definition 16. Set of user-subject-modifiable-tuples

UAVTModSAVT $\subseteq$ UAVT $\times$ UAVT $\times$ SAVT $\times$ SAVT

UAVTModSAVT=$\{\langle i, j, k, l\rangle$ | i $\in$ UAVT $\wedge$ j$\in$ UAVT

$\wedge$ k $\in$ SAVT $\wedge$ l $\in$ SAVT

$\wedge$ ChkConstrSubModbyUser$(i, j, k, l)\}$

Definition 17. Set of subject-subject-modifiable-tuples

SAVTModSAVT $\subseteq$ SAVT $\times$ SAVT $\times$ SAVT $\times$ SAVT

SAVTModSAVT=$\{\langle i, j, k, l\rangle$ | i $\in$ SAVT $\wedge$ j$\in$ SAVT

$\wedge$ k $\in$ SAVT $\wedge$ l $\in$ SAVT

$\wedge$ChkConstrSubModBySub$(i, j, k, l)\}$

Definition 18. Set of subject-object-creatable-tuples

SAVTCrOAVT $\subseteq$ SAVT $\times$ OAVT

SAVTCrOAVT =$\{\langle i, j\rangle$ | i $\in$ SAVT $\wedge$ j $\in$ OAVT

$\wedge$ ChkConstrObjCreatebySub(i, j) $\}$

Definition 19. Set of subject-object-modifiable-tuples

SAVTModOAVT$\subseteq$ SAVT $\times$ OAVT $\times$ OAVT

SAVTModOAVT =$\{\langle i, j, k\rangle$ | i $\in$ SAVT $\wedge$ j $\in$ OAVT

$\wedge$ k $\in$ OAVT $\wedge$ ChkConstrObjModbySub(i,j,k)$\}$

PreCondition Evaluation Functions for $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$

- **CheckPCNCR**$(uc_r, avt_1, avt_2, avt_3, avt_4)$ checks non creating command and returns true or false.

- **CheckPCCR**$(uc_r, avt_1, avt_2, avt_3)$ checks creating command and returns true or false.

- **CheckPCDel(**$uc_r, avtf(s), avtf(o), avt$**)** checks deleting command and returns true or false.

Object Schema of $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$**:** Every $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ entity (user, subject, object) is represented as a $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ object and the attribute entity_type specifies whether a particular $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ object is $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ user, subject or object. User, subject and object attributes of $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ are represented as $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ object attributes. There is no user creation in $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$, so $\mathrm{U}^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$ is a finite set. $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ function SubCreator is configured here with a mandatory $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ object attribute whose domain would be finite set of users ($\mathrm{U}^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$). To determine which user is the creator of an $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ subject, $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ object needs to have another mandatory attribute user_name whose range is also finite set of users ($\mathrm{U}^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$). We consider “NULL" as a special attribute value for any atomic or set valued attribute. It is assigned to an attribute which is not appropriate for a particular entity. We need to add “NULL" in the range of UA, SA and OA for this reduction. $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ attribute set ATT = {entity_type, user_name, SubCreator} $\cup$ $\mathrm{UA}^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$ $\cup$ $\mathrm{SA}^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$ $\cup$ $\mathrm{OA}^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$.

$\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ **usage rights UR:** In this reduction each $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ permission is considered as a usage right in $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ and additionally a dummy right d is introduced. Each $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ command associates with a right. We use dummy right d for association with the commands which are defined to configure $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ operations. Usage Right $\mathrm{UR}^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ = $\mathrm{P}^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$ $\cup$ $\{d\}$.

$\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ **commands:** $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ operations are reduced to specific $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ commands. We use the sets of eligible attribute value tuples to define $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ commands. It defines a creating command for each element of UAVTCrSAVT, SAVTCrSAVT, SAVTCrOAVT , a non-creating command for each element of UAVTModSAVT, SAVTModSAVT and SAVTModOAVT and a deleting command for each UAVTDelSAVT, SAVTDelSAVT. For example consider an $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ subject creation policy where a user u with attribute value tuple $uavt$ is allowed to create a subject s with attribute value tuple $savt$, so by definition $\langle uavt, savt\rangle \in$ UAVTCrSAVT. For each element $\langle i, j\rangle \in$ UAVTCrSAVT this reduction has a command named CreateSubject_ij(s, o) which creates an object o with entity_type = subject. This is shown in top left quadrant of Table 4.8. The top right quadrant of Table 4.8 shows the reduction of $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ operation $\mathrm{CreateSubjectbySubject}^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$

Table 4.8: $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ Creating Commands

For each $\langle i, j \rangle \in$ UAVTCrSAVT **CreateSubjectbyUser_ij$_d$(s, o)** PreCondition: s.entity_type = user $\wedge\ s.ua_1 = i_1 \wedge \ldots \wedge s.ua_m = i_m$ PreUpdate: create o o.entity_type = subject o.user_name = NULL o.SubCreator = s.user_name o.ua$_1$ = NULL $\vdots$ o.ua$_m$ = NULL o.sa$_1$ = j$_1$ $\vdots$ o.sa$_n$ = j$_n$ o.oa$_1$ = NULL $\vdots$ o.oa$_p$ = NULL	For each $\langle i, j, k \rangle \in$ SAVTCrSAVT **CreateSubjectbySubject_ijk$_d$(s, o)** PreCondition: s.entity_type = subject $\wedge\ s.sa_1 = i_1 \wedge \ldots \wedge s.sa_n = i_n$ PreUpdate: create o o.entity_type = subject o.user_name = NULL o.SubCreator = s.SubCreator o.ua$_1$ = NULL $\vdots$ o.ua$_m$ = NULL o.sa$_1$ = k$_1$ $\vdots$ o.sa$_n$ = k$_n$ o.oa$_1$ = NULL $\vdots$ o.oa$_p$ = NULL s.sa$_1$ = j$_1$ $\vdots$ s.sa$_n$ = j$_n$
For each $\langle i, j \rangle \in$ SAVTCrOAVT **CreateObjectbySub_ij$_d$(s,o)** PreCondition: s.entity_type = subject $\wedge\ s.sa_1 = i_1 \wedge \ldots \wedge s.sa_n = i_n$ PreUpdate: create o o.entity_type = object o.user_name = NULL o.SubCreator = NULL o.ua$_1$ = NULL $\vdots$ o.ua$_m$ = NULL o.sa$_1$ = NULL $\vdots$ o.sa$_n$ = NULL o.oa$_1$ = j$_1$ $\vdots$ o.oa$_p$ = j$_p$	

Table 4.9: $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ Non-Creating Commands

for each r ∈ $\mathrm{UR}^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \setminus \{\mathrm{d}\}$ **Access**$_r(s,o)$ PreCondition: s.entity_type = subject ∧ o.entity_type = object ∧ ChkAuth(r,avtf(s),avtf(o)) PreUpdate: N/A	for each r ∈ $\mathrm{UR}^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \setminus \{\mathrm{d}\}$ **AccessSubject**$_r(s,o)$ PreCondition: s.entity_type = subject ∧ o.entity_type = subject ∧ ChkAuthSubject(r,avtf(s),avtf(o)) PreUpdate: N/A
For each $\langle i,j,k,l\rangle \in$ UAVTModSAVT **ModifySubjectAttbyUser_ijkl**$_d$**(s,o)** PreCondition: s.entity_type = user ∧ o.entity_type = subject $\wedge$ o.SubCreator = s.user_name $\wedge\ s.ua_1 = i_1 \wedge \ldots \wedge s.ua_m = i_m$ $\wedge\ o.sa_1 = k_1 \wedge \ldots \wedge o.sa_n = k_n$ PreUpdate: s.ua$_1$ = j$_1$ ⋮ s.ua$_m$ = j$_m$ o.sa$_1$ = l$_1$ ⋮ o.sa$_n$ = l$_n$	For each $\langle i,j,k,l\rangle \in$ SAVTModSAVT **ModifySubjectAttbySub_ijkl**$_d$**(s,o)** PreCondition: s.entity_type = subject ∧ o.entity_type = subject s.SubCreator = o.SubCreator $\wedge\ s.sa_1 = i_1 \wedge \ldots \wedge s.sa_n = i_n$ $\wedge\ o.sa_1 = k_1 \wedge \ldots \wedge s.sa_n = k_n$ PreUpdate: s.sa$_1$ = j$_1$ ⋮ s.sa$_n$ = j$_n$ o.sa$_1$ = l$_1$ ⋮ o.sa$_n$ = l$_n$
For each $\langle i,j,k\rangle \in$ SAVTModOAVT **ModifyObjectAttbySub_ijk**$_d$**(s,o)** PreCondition: s.entity_type = subject ∧ o.entity_type = object $\wedge\ s.sa_1 = i_1 \wedge \ldots\ s.sa_n = i_n$ $\wedge\ o.oa_1 = j_1 \wedge \ldots\ s.oa_p = j_p$ PreUpdate: o.oa$_1$ = k$_1$ ⋮ o.oa$_p$ = k$_p$	

$(s_1, s_2, savt_1, savt_2)$ by $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ commands CreateSubjectbySubject_ijk$_d(s,o)$. The bottom left quadrant shows the reduction of $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ operation create object by subject. Each Access$_r^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}(s,o)$ configures Access$_p^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}(s,o)$ where r = p. Here Access$_r^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ is a non-creating command with PreCondition part only and PreCondition checks the authorization evaluation function of $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$. This is shown on the top left hand side of Table 4.9. There is no PreUpdate in these commands. The top right hand side of Table 4.9 similarly shows the $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ commands to simulate the $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ command AccessSubject$_p^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}(s_1, s_2)$ for r = p. The middle row left column of Table 4.9 the simulation of the ModifySubjectAttbyUser$^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}(u, s, uavt, savt)$ by ModifySubjectAttbyUser_ijkl$_d^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}(s,o)$. The middle row right column of Table 4.9 shows the reduction of the $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ modify subject attribute by subject while bottom row left column of same Table shows the reduction of $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ modify object attribute by subject. Table 4.10 shows

the reduction of $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ deleting commands by user (left) and by subject (right).

Table 4.10: $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ Deleting Commands

For each $\langle i, j \rangle \in$ UAVTDelSAVT **DeleteSubbyUser_ij$_d$(s,o)** PreCondition: s.entity_type = user $\wedge$ o.entity_type = subject $\wedge\ s.ua_1 = i_1 \wedge \ldots \wedge s.ua_m = i_m$ $\wedge\ o.sa_1 = j_1 \wedge \ldots \wedge o.sa_n = j_n$ PreUpdate: delete o	For each $\langle i, j, k \rangle \in$ SAVTDelSAVT **DeleteSubbySub_ijk$_d$(s,o)** PreCondition: s.entity_type = subject $\wedge$ o.entity_type = subject $\wedge\ s.ua_1 = i_1 \wedge \ldots \wedge s.ua_m = i_n$ $\wedge\ o.sa_1 = j_1 \wedge \ldots \wedge o.sa_n = j_n$ PreUpdate: delete o s.sa$_1$ = k$_1$ $\vdots$ s.sa$_n$ = k$_n$

Reduction from $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ to $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$

In this subsection we provide a reduction from $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ to $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$. Before going to detail about the construction here we give the high light of how this construction works.

Construction Outline

- This construction only does sequential simulation. No simultaneous access is possible.
- To ensure the sequential access there is a single user in the system who sets a subject token and its own token before attempting the access and takes back that token after the access or after denial of access.
- This construction has only one user and no objects. Set of subjects should be initialized with $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ objects.
- Subject creation, deletion by user and object creation, modification, access by subject is not allowed. The policies of those operations are set as false for this construction.
- The only allowed operations are subject modification by user, subject creation, modification, deletion and access by subject.
- A single $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ command is configured with a sequence of $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ operations. Each operation changes the state. To track the change of state we are using one user attribute (uTo-

ken) and 8 subject attributes ($sToken$, $comm$, $commType$, $newVal_1$, $newVal_2$, $isAuth_z$, $isPassed$, $isRlsdToken$). State tracking attributes are additional subject attributes on top of the $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ object attributes.

- $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ preconditions ($f_b(s,o)$, $f_b(s)$)are command specific boolean functions which use existing attribute value of subjects. On the other hand $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ policies are configurable boolean expressions which can use existing and proposed attribute values of concerned entities. $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ uses function $f_1(s,o)$/ $f_1(s)$ and $f_2(s,o)$/ $f_2(s)$ to compute the proposed value for the subjects and/ target subject. While $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ supports attribute value to be supplied directly from the range of a specific attribute. To convert the command specific boolean functions and $f_1(s,o)$/ $f_1(s)$ and $f_2(s,o)$/ $f_2(s)$ into a configurable boolean expression this construction takes the following steps:

  1. Pre-computes preconditions and f_1 and f_2 for all the commands and for all possible existing attribute value tuples and proposed attribute value tuples for source and target subjects and stores in a truth table. So there are three truth tables for three types of command non-creating, creating and deletion.
  2. Constructs a conjunctive normal form (CNF) for every true rows of the truth table and make a boolean expression by disjunction of all the CNFs. This boolean expression is used in configuring $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ policies for corresponding operations.

Construction Detail

For this configuration we have only one user u_1, objects of $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ are mapped as subjects of $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ and there are no objects in $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$. Rights of $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ are mapped as permissions of $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$. There is only one user attribute $uToken$ and no object attributes. Subjects have eight additional attributes ($sToken$, $comm$, $commType$, $newVal_1$, $newVal_2$, $isAuth_z$, $isPassed$, $isRlsdToken$) along with all the $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ object attributes. Table 4.11 shows the reduction of basic sets and functions from $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ to $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$. $uToken$ and $sToken$ specify the user token and subject token respectively. $comm$ and $commType$ specify the command name and

Table 4.11: Basic Sets and Functions Reduction from $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ to $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$

UA = {uToken}
SA = {sToken, comm, commType, newVal$_1$, newVal$_2$, isAuth$_z$, isPassed, isRlsdToken}
$\cup$ ATT$^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$
Where ATT$^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ ={ a$_1$, a$_2$, ... a$_n$ }
OA = $\emptyset$
Range($uToken$) = Range($sToken$) = {0,1}
Range($comm$) = UC$^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$
Range($commType$) = {Create, Delete, Modify}
Range($newVal_1$) = Range($newVal_2$) =$\sigma(a_1) \times \ldots \times \sigma(a_n)$
Range($isAuth_z$) = {0,1,2}
Range($isPassed$) = {0,1,2}
Range($isRlsdToken$) = {0,1,2}
Range(a_i) = $\sigma(a_i)$, i = 1 to n
UAVT = Range (uToken)
SAVT = Range (sToken) $\times$ Range (comm) $\times$ Range (commType) $\times$ Range (isAuth$_z$) $\times$ Range (isPassed) $\times$ isRlsdToken $\times$ $\sigma(a_1) \times \ldots$ $\sigma(a_n)$
OAVT = $\emptyset$
P$^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$ = UC$^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$

command type (create, modify, delete) respectively, $newVal_1$ and $newVal_2$ specify the proposed new value tuple for source and target subject, $isAuth_z$ keeps track of whether a particular command is authorized, $isPassed$ keeps track of whether the pre update and/ create/ delete and access has completed, $isRlsdToken$ specifies whether target has released its token to source. The initial state of the system is configured as follows: U $^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$ = {u}

S $^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$ = O $^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$

O $^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$ = $\emptyset$

uToken(u)= 0

For each subject s $\in$ S $^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$,

where o$^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ $\mapsto$ s$^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$

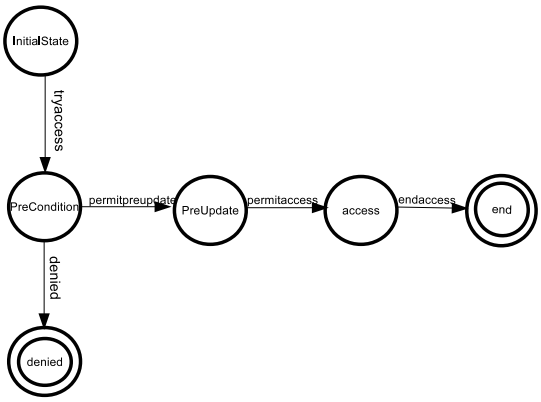
- SubCreator(s) = u
- sToken(s) = 0
- comm= null
- commType= null

- newVal$_1$= null
- newVal$_2$=null
- isAuth$_z$=2
- isPassed= 2
- isRlsdToken = 2
- a_1= o.a_1
- $\vdots$
- a_n=o.a_n

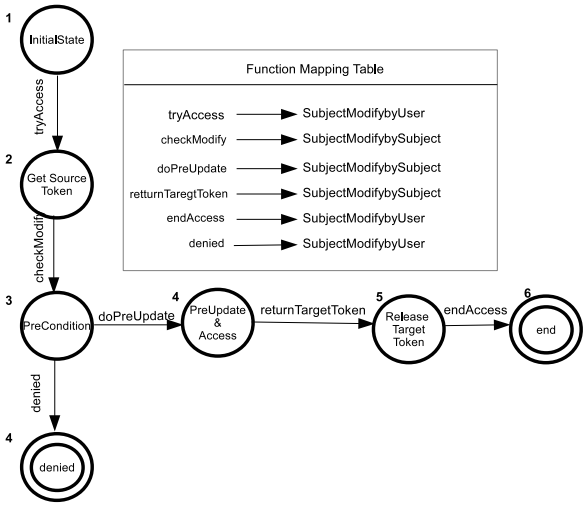
Table 4.12 specifies the $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ policies to configure for the reduction, the details about the policies are provided in the Table 4.14, Table 4.15,Table 4.16 and Table 4.17. Figure 4.2, Figure 4.3 and Figure 4.4 shows the state transition actions of $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ configuration $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ non-creating, creating and deleting commands respectively. Analyzing these figures we understand that a single type $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ operations need to handle different state transitions which are shown in Figure 4.2(b), Figure 4.3(b) and Figure 4.4 (b).

- SubjectModifybyUser handles tryaccess, endAccess and denied
- SubjectModifybySubject handles checkModify, checkCreate, checkDelete, doPreUpdate and returnTargetToken
- SubjectCreatebySubject handles doCreate
- SubjectDeletebySubject handles doDelete

Table 4.13 shows an example how a sequence of $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ operations configure a $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ non-creating command uc$_r$(s,o).

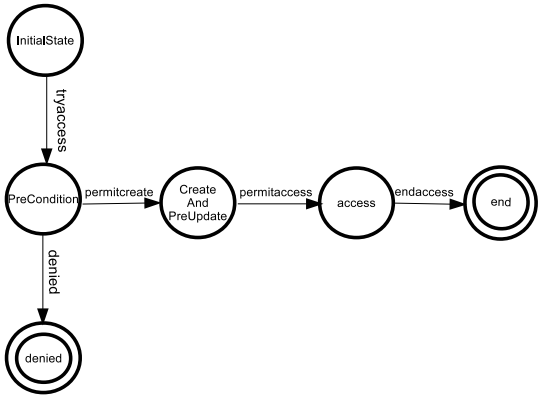


(a) State transition actions with a Non Creating Command for a single $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ access.

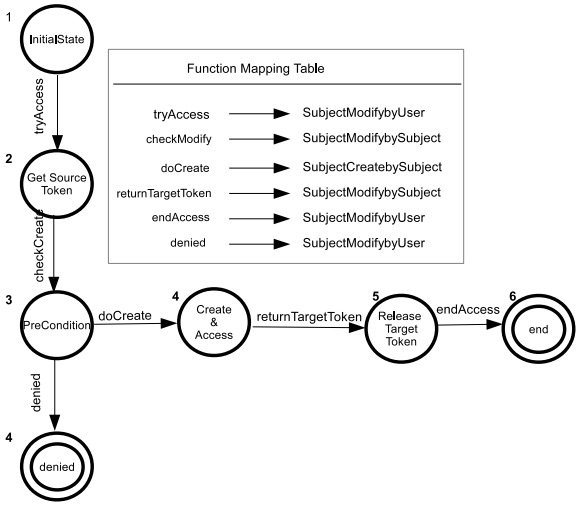


(b) Mapped $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ State transition actions with a Non Creating Command for a single $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ access.

Figure 4.2: Mapping of $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ Non-Creating Command



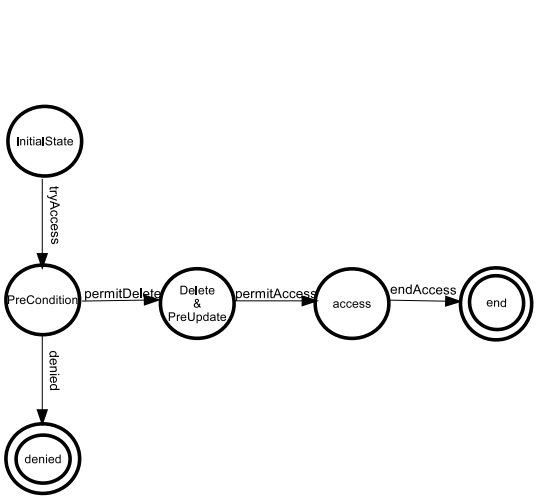
(a) State transition actions with a Creating Command for a single $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ access.



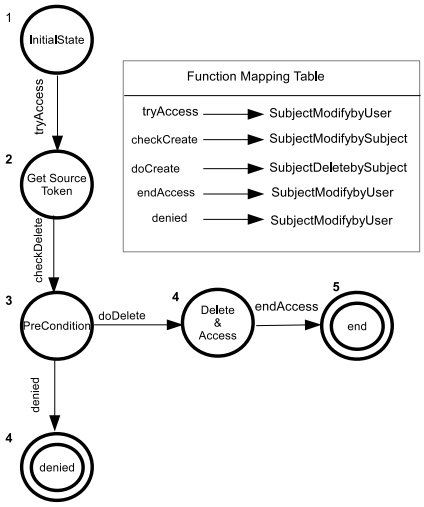
(b) Mapped $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ State transition actions with a Creating Command for a single $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ access.

Figure 4.3: Mapping of $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ Creating Command

For this configuration $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ uses only four operations : 1)SubjectModifybyUser, 2)SubjectModifybySubject, 3)SubjectCreatebySubject and 4)SubjectDeletebySubject). Other $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ operations such as SubjectCreatebyUser, SubjectDeletebyUser, ObjectCreatebySubject and ObjectModifybySubject are not permitted for this construction. So the policy for these operations are configured as false. To handle different state transitions using a single $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ operation, its pol-



(a) State transition actions with a Deleting Command for a single $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ access.



(b) Mapped $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ State transition actions with a Deleting Command for a single $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ access.

Figure 4.4: Mapping of $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ Deleting Command

icy should have several disjunctive policies where we have to make sure only one disjunctive policy would be true for a specific state transition function. Table 4.12 shows the disjunctive policy configuration. The detail constructions of these policies are provided in Table 4.14, 4.15, 4.16, 4.17. Table 4.14 shows the configuration of constraints for 1)giving user token to source subject, 2)return target token to source after authorized access and 3) return user token after authorized access and 4) return user token after denial of access. Table 4.15 shows the configuration of constrains for 1) checking whether the modification is authorized, 2) checking whether the modification is not authorized, 3)constraint for modification and access. Similarly Table 4.16 and Table 4.17 show the configuration for constraints of creation and deletion respectively. Here NCPC, CPC and DelPC are actually the boolean expression constructed from the disjuction of the CNF of truth tables for non-creating, creating and deleting commands respectively.

Here is a brief description of the constraint provided in Table 4.14.

- **ConstrTryAccess(**$u, s, uavt, savt$**)** checks whether user token and subject token is 0, that means no access is currently executing and it also checks whether uavt and savt contains proposed attribute value for tokens are 1 and other information regarding the commands.

- **ConstrRtrnTrgtToken(**s_1**,**s_2**,**$savt_1$**,**$savt_2$**)** checks whether the authorized access is already

Table 4.12: $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ Policy Configuration

$\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ **Authorization Policy**
Authorization$_p$(s,o) $\equiv$ False AuthorizationonSubject$_p$(s$_1$,s$_2$) $\equiv$ True
$\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ **Creation, Deletion and Modification Policy**
• ConstrSubCreatebyUser($u, s, savt$) $\equiv$ False • ConstrSubDelbyUser(u, s)$\equiv$ False • ConstrObjCreatebySub($s, o, oavt$)$\equiv$ False • ConstrObjModbySub($s, o, oavt$)$\equiv$ False • ConstrSubCreatebysub($s_1, s_2, savt_1, savt_2$) $\equiv$ ConstrPermitCreate($s_1, s_2, savt_1, savt_2$) • ConstrSubDelbySub($s_1, s_2, savt$) $\equiv$ ConstrPermitDelete($s_1, s_2, savt$) • ConstrSubModbyUser($u, s, uavt, savt$) $\equiv$ ConstrTryAccess(u,s,uavt,savt) $\vee$ ConstrEndAccess(u,s,uavt,savt) $\vee$ ConstrDenied(u,s,uavt,savt) • ConstrSubModBySub($s_1, s_2, savt_1, savt_2$)$\equiv$ ConstrCheckModifyPos($s_1, s_2, savt_1, savt_2$) $\vee$ ConstrCheckModifyNeg($s_1, s_2, savt_1, savt_2$) $\vee$ ConstrCheckCreatePos($s_1, s_2, savt_1, savt_2$) $\vee$ ConstrCheckCreateNeg($s_1, s_2, savt_1, savt_2$) $\vee$ ConstrCheckDeletePos($s_1, s_2, savt_1, savt_2$) $\vee$ ConstrCheckDeleteNeg($s_1, s_2, savt_1, savt_2$) $\vee$ ConstrPermitPreUpdate($s_1, s_2, savt_1, savt_2$) $\vee$ ConstrRtrnTrgtToken($s_1, s_2, savt_1, savt_2$)

passed ((s_1's and s_2's isAuthorized= 1 and isPassed =1) and also checks whether $savt_1$ contains isRlsdToken = 1 and $savt_2$ contains sToken = 0, isAuthorized = 2 , isPassed = 2, isRlsdToken = 2 and nullify the other information regarding the command.

- **ConstrEndAccess(**$u, s, uavt, savt$**)** checks whether the authorized access is already passed and target token is returned ((s_1's and s_2's isAuthorized= 1, isPassed =1 and isRlsdToken = 1) also checks uavt and savt has proposed token 0 and nullify the other information regarding the command.

- **ConstrDenied(**$u, s, uavt, savt$**)** checks whether access is not authorized and user and sub-

Table 4.13: A Sequence of Actions in $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ to Configure the $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ Non-Creating Command uc_r(s, o)

State Transition Actions	$\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ Operations
tryaccess(Give Source Token)	ModifySubjectAttbyUser(u,s, $\langle uToken = 1\rangle$, $\langle sToken = 1, comm = uc_r, commType = Modify, newVal_1 = \langle f_{1,a_1}^{uc_r}(s), \ldots f_{1,a_n}^{uc_r}(s)\rangle, newVal_2 = \langle f_{2,a_1}^{uc_r}(s), \ldots f_{2,a_n}^{uc_r}(s)\rangle, isAuth_z = 2, isPassed = 2, isRlsdToken = 2\rangle$)
checkModify(Check Pre-Condition if Authorized)	ModifySubjectAttBySubject(s_1, s_2, $\langle sToken = 1, comm = comma(s_1), commType = Modify, newVal_1 = newVal_1(s_1), newVal_2 = newVal_2(s_1), isAuth_z = 1, isPassed = 0, isRlsdToken = 0, a_1 = a_1(s_1), \ldots, a_n = a_n(s_1)\rangle$, $\langle 1, comm(s_1), commType = Modify, newVal_1 = newVal_1(s_1), newVal_2 = newVal_2(s_1), isAuth_z = 1, isPassed = 0, isRlsdToken = 0, a_1 = a_1(s_2), \ldots, a_n = a_n(s_2)\rangle$)
checkModify(Check Pre-Condition if not Authorized)	ModifySubjectAttBySubject(s_1, s_2, $\langle sToken = 1, comm = command(s_1), commType = Modify, newVal_1 = newVal_1(s_1), newVal_2 = newVal_2(s_1), isAuth_z = 0, isPassed = 2, isRlsdToken = 2, a_1 = a_1(s_1), \ldots, a_n = a_n(s_1)\rangle$, $\langle 1, comm(s_1), commType = Modify, newVal_1 = newVal_1(s_1), newVal_2 = newVal_2(s_1), isAuth_z = 2, isPassed = 2, isRlsdToken = 2, a_1 = a_1(s_2), \ldots, a_n = a_n(s_2)\rangle$)
doPreUpdate (PreUpdate and Access if Authorized)	ModifySubjectAttBySubject(s_1, s_2, $\langle sToken = 1, comm = comm(s_1), commType = Modify, newVal_1 = newVal_1(s_1), newVal_2 = newVal_2(s_1), isAuth_z = 1, isPassed = 1, isRlsdToken = 0, a_1 = avt_1.a_1, \ldots.avt_1.a_n\rangle$, $\langle sToken = 1, comm = comm(s_1), commType = Modify, newVal_1 = newVal_1(s_1), newVal_2 = newVal_2(s_1), isAuth_z = 1, isPassed = 1, isRlsdToken = 0, a_1 = avt_2.a_1, \ldots.avt_2.a_n\rangle$)
returnTargetToken(Release Target Token)	ModifySubjectAttBySubject(s_1, s_2,$\langle sToken = 1, comm = comm(s_1), commType = commType(s_1), newVal_1 = newVal_1(s_1), newVal_2 = newVal_2(s_1), isAuth_z = isAuth_z(s_1), isPassed = 1, isRlsdToken = 0, a_1 = a_1(s_1), \ldots, a_n = a_n(s_1)\rangle$, $\langle sToken = 0, comm = null, commType = null, newVal_1 = null, newVal_2 = null, isAuth_z = 2, isPassed = 2, isRlsdToken = 2, a_1 = a_1(s_2), \ldots, a_n = a_n(s_2)\rangle$)
denied/endAccess(Release Source Token)	ModifySubjectAttbyUser(u,s, $\langle uToken = 0\rangle$, $\langle sToken = 0, comm = null, commType = null, newVal_1 = null, newVal_2 = null, isAuth_z = 2, isPassed = 2, isRlsdToken = 2, a_1 = a_1(s), \ldots, a_n = a_n(s)\rangle$)

ject still has active token(sToken =1, isAuthorized= 0, isPassed =2 and isRlsdToken=2) and nullify the user and subject token and all the command specific information the subject contains.

Here is the brief description of the constraints provided in Table 4.15, 4.16 and 4.17

- **ConstrCheckModifyPos(**s_1**,**s_2**,**$savt_1$**,**$savt_2$**)** Checks whether the non-creating command comm (s_1) is authorized. Here NCPC is the boolean expression constructed from the truth table of non-creating commands.

- **ConstrCheckModifyNeg(**s_1**,**s_2**,**$savt_1$**,**$savt_2$**)** Checks whether the non-creating command comm(s_1) is not authorized. Here NCPC is the boolean expression constructed from the truth table of non-creating commands.

Table 4.14: Configuration of Constraint: Give and Back User Token and Subject Token

Constraint for Giving User Token to Source Subject
ConstrTryAccess(u,s,uavt,savt) $\equiv$ uToken(u) =0 $\wedge$ sToken(s)=0 $\wedge$ comm(s)= null $\wedge$ commType(s)= null $\wedge$ isAuth$_z$ (s) 2 $\wedge$ isPassed(s)= 2 $\wedge$ isRlsdToken (s)= 2 uToken$'$(u) =1 $\wedge$ sToken$'$(s)=1 $\wedge$ comm$'$(s)$\in$ UC$^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ $\wedge$ commType$'$(s)$\in$ Range (commType) $\wedge$ isAuth$_z{}'$(s)= 2 $\wedge$ isPassed$'$(s)= 2 $\wedge$ isRlsdToken$'$(s)= 2 $\wedge$ a$'_1$(s)= a$_1$(s) $\wedge$ . . . $\wedge$ a$'_n$(s)=a$_n$(s)
Return target token to source after authorized access
ConstrRtrnTrgtToken($s_1, s_2, savt_1, savt_2$) $\equiv$ sToken(s_1) =1 $\wedge$ comm(s_1)= comm(s_1) $\wedge$ commType(s_1)= commType(s_1) $\wedge$ isAuth$_z$(s_1) = 1 $\wedge$ isPassed(s_1)= 1 $\wedge$ isRlsdToken(s_1) = 0 $\wedge$ sToken(s_2)=1 $\wedge$ comm(s_2) = comm(s_1) $\wedge$ commType(s_2)= commType(s_1) $\wedge$ isAuth$_z$(s_2) = 1 $\wedge$ isPassed(s_2)= 1 $\wedge$ isRlsdToken(s_2) = 0 $\wedge$ sToken$'$(s$_1$)=1 $\wedge$ comm$'$(s$_1$)=comm(s_1)$\wedge$ commType$'$(s$_1$)= commType(s_1) $\wedge$ isAuth$_z{}'$(s$_1$) = 1 $\wedge$ isPassed$'$(s$_1$)= 1 $\wedge$ isRlsdToken$'$(s$_1$) = 1 $\wedge$ a$_1{}'$(s$_1$) = a$_1$(s$_1$)$\wedge$ . . . $\wedge$ a$_n{}'$(s$_1$) = a$_n$(s$_1$) $\wedge$ sToken$'$(s$_2$)=0 $\wedge$ comm$'$(s$_2$)=null $\wedge$ commType$'$(s$_2$)= null $\wedge$ isAuth$_z{}'$(s$_2$) = 2 $\wedge$ isPassed$'$(s$_2$)= 2 $\wedge$ isRlsdToken$'$(s$_2$) = 2 $\wedge$ a$_1{}'$(s$_2$) = a$_1$(s$_2$) $\wedge$ . . . $\wedge$ a$_n{}'$(s$_2$) = a$_n$(s$_2$)
Return user token after authorized access
ConstrEndAccess($u, s, uavt, savt$) $\equiv$ uToken(u) = 1 $\wedge$ sToken(s)=1 $\wedge$ comm(s)= comm(s) $\wedge$ commType(s)= commType(s) $\wedge$ isAuth$_z$(s) = 1 $\wedge$ isPassed(s)= 1 $\wedge$ isRlsdToken(s) = 1 $\wedge$ uToken(u) = 0$\wedge$ sToken$'$(s) =0 $\wedge$ comm$'$(s)= null $\wedge$ commType$'$(s)= null $\wedge$ isAuth$_z{}'$(s) = 2 $\wedge$ isPassed$'$(s)= 2 $\wedge$ isRlsdToken$'$(s) = 2 $\wedge$ a$_1{}'$(s)=a$_1$(s) $\wedge$ . . . $\wedge$ a$_n{}'$(s)= a$_n$(s))
Return User Token After Denial
ConstrDeniedAccess($u, s, uavt, savt$) uToken(u) = 1$\wedge$ sToken(s)=1 $\wedge$ comm(s)= comm(s) $\wedge$ commType(s)= commType(s)$\wedge$ isAuth$_z$(s) = 0 $\wedge$ isPassed(s)= 2 $\wedge$ isRlsdToken(s) = 2 $\wedge$ uToken(u) = 0$\wedge$ sToken$'$(s) =0 $\wedge$ comm$'$(s)= null $\wedge$ commType$'$(s)= null $\wedge$ isAuth$_z{}'$(s) = 2 $\wedge$ isPassed$'$(s)= 2 $\wedge$ isRlsdToken$'$(s) = 2 $\wedge$ a$_1{}'$(s)=a$_1$(s) $\wedge$ . . . $\wedge$ a$_n{}'$(s)= a$_n$(s))

- **ConstrCheckCreatePos(**s_1**,**s_2**,**$savt_1$**,**$savt_2$**)** Checks whether the creating command comm (s_1) is authorized. Here CPC is the boolean expression constructed from the truth table of creating commands.

- **ConstrCheckCreateNeg(**s_1**,**s_2**,**$savt_1$**,**$savt_2$**)** Checks whether the creating command comm (s_1) is not authorized. Here CPC is the boolean expression constructed from the truth table of creating commands.

- **ConstrDeleteCreatePos(**s_1**,**s_2**,**$savt_1$**,**$savt_2$**)** Checks whether the deleting command comm(s_1) is authorized. Here DelPC is the boolean expression constructed from the truth table of deleting

Table 4.15: Configuration of Constraints: for Modify(Checking authorization, modify)

Check Whether Modification is Authorized
ConstrCheckModifyPos($s_1, s_2, savt_1, savt_2$) ≡ sToken(s_1)=1 ∧ comm(s_1)=comm(s_1)∧ commType(s_1)= Modify∧ isAuth$_z$(s_1) = 2∧ isPassed= 2∧ isRlsdToken(s_1) = 2) ∧ sToken(s_2)=0 ∧ comm(s_2) = null ∧ commType(s_2)= null ∧ isAuth$_z$(s_2) = 2 ∧ isPassed(s_2)= 2 ∧ isRlsdToken(s_2)= 2) ∧ NCPC ∧ sToken′(s_1)=1 comm′(s_1)=comm(s_1), commType′(s_1)= Modify, isAuth$_z$′(s_1) = 1 ∧ isPassed′(s_1)= 0
Check Whether modification is not authorized
ConstrCheckModifyNeg($s_1, s_2, savt_1, savt_2$)≡ sToken(s_1)=1 ∧ comm(s_1)=comm(s_1) ∧ commType(s_1)= Modify ∧ isAuth$_z$(s_1) = 2∧ isPassed(s_1)= 2 ∧ isRlsdToken(s_1) = 2 ∧ sToken(s_2)=0 ∧ comm(s_2) = null ∧ commType(s_2)= null ∧ isAuth$_z$(s_2) = 2 ∧ isPassed(s_2)= 2 ∧ isRlsdToken = 2 ∧ !(NCPC∧ NCf$_1$ ∧ NCf$_2$) ∧ sToken′(s_1)=1 ∧ comm′(s_1)=comm(s_1)∧ commType′(s_1)= Modify ∧ isAuth$_z$′(s_1) = 0 ∧ isPassed′(s_1)= 2 ∧ isRlsdToken′(s_1) = 2∧ a$_1$′(s_1)=a$_1$(s$_1$) ∧ . . . ∧ a$_n$′(s_1)=a$_n$(s$_1$) ∧ sToken′(s_2)=0∧ comm′(s_2)=null∧ commType′(s_2)=null∧ isAuth$_z$ ′(s_2)= 2∧ isPassed′(s_2)= 2 ∧ isRlsdToken′(s_2) = 2∧ a$_1$′(s_2)=a$_1$(s$_2$)∧ . . . ∧ a$_n$′(s_2)=a$_n$(s$_2$)
Do PreUpdate
ConstrPermitModify($s_1, s_2, savt_1, savt_2$) ≡ sToken(s$_1$)=1 ∧ comm(s$_1$)=comm(s_1)∧ commType(s$_1$)= Modify∧ isAuth$_z$(s$_1$) = 1∧ isPassed(s$_1$)= 0∧ isRlsdToken(s$_1$) = 0 ∧ sToken(s$_2$)=1∧ comm(s$_2$) = comm(s_1)∧ commType(s$_2$)= Modify ∧ isAuth$_z$(s$_2$) = 1∧ isPassed(s$_2$)= 0 ∧ isRlsdToken(s$_2$) = 0 ∧ sToken′(s$_1$)=1 ∧ comm′(s$_1$)=comm(s_1) ∧ commType′(s$_1$)= Modify ∧ isAuth$_z$′(s$_1$) = 1 ∧ isPassed′(s$_1$)= 1 ∧ isRlsdToken′(s$_1$) = 0 ∧ sToken′(s$_2$)=0 ∧ comm′(s$_2$)=comm(s_1)∧ commType′(s$_2$)= Modify ∧ isAuth$_z$′(s$_2$) = 1 ∧ isPassed′(s$_2$)= 1 ∧ isRlsdToken′(s$_2$) = 0

- **ConstrDeleteCreateNeg(**s_1**,**s_2**,**$savt_1$**,**$savt_2$**)** Checks whether the deleting command comm(s_1) is not authorized. Here DelPC is the boolean expression constructed from the truth table of deleting commands.

- **ConstrPermitModify(**s_1**,**s_2**,**$savt_1$**,**$savt_2$**)** Check the state attribute of an authorized non-creating command and permit access and do the update (isAuthorized= 1, isPassed=1, isRlsdToken = 0)

- **ConstrPermitCreate(**s_1**,**s_2**,**$savt_1$**,**$savt_2$**)** Check the state attribute of an authorized creating command and permit access and do the create (isAuthorized= 1, isPassed=1, isRlsdToken = 0)

Table 4.16: Configuration of Constraints: for Create(Checking authorization, Create)

Check Whether creation is authorized
ConstrCheckCreatePos($s_1, s_2, savt_1, savt_2$) ≡ sToken(s_1)=1 ∧ comm(s_1)=comm(s_1)∧ commType(s_1)= Create∧ isAuth$_z$(s_1) = 2∧ isPassed= 2∧ isRlsdToken(s_1) = 2) ∧ sToken(s_2)=0 ∧ comm(s_2) = null ∧ commType(s_2)= null ∧ isAuth$_z$(s_2) = 2 ∧ isPassed(s_2)= 2 ∧ isRlsdToken(s_2)= 2) ∧ (CPC) ∧ sToken′(s_1)=1 comm′(s_1)=comm(s_1), commType′(s_1)= Modify, isAuth$_z$′(s_1) = 1 ∧ isPassed′(s_1)= 0 ∧ isRlsdToken′(s_1) = 0 ∧ a$_1$′(s_1)=a$_1$(s$_1$) ∧ . . . a′$_n$(s_1)=a$_n$(s$_1$) ∧ sToken′(s_2)=1∧ comm′(s_2)=comm(s_1)∧ commType′(s_2)= Create∧ isAuth$_z$′(s_2) = 1∧ isPassed′(s_2)= 0 ∧isRlsdToken′(s_2) = 0∧ a$_1$′(s_2)=a$_1$(s$_2$)∧ . . . ∧ a$_n$′(s_2)=a$_n$(s$_2$)
Check Whether creation is not authorized
ConstrCheckCreateNeg($s_1, s_2, savt_1, savt_2$)≡ sToken(s_1)=1 ∧ comm(s_1)=comm(s_1) ∧ commType(s_1)= Create ∧ isAuth$_z$(s_1) = 2∧ isPassed(s_1)= 2 ∧ isRlsdToken(s_1) = 2 ∧ sToken(s_2)=0 ∧ comm(s_2) = null ∧ commType(s_2)= null ∧ isAuth$_z$(s_2) = 2 ∧ isPassed(s_2)= 2 ∧ isRlsdToken = 2 ∧ !(CPC) ∧ sToken′(s_1)=1 ∧ comm′(s_1)=comm(s_1)∧ commType′(s_1)= Create ∧ isAuth$_z$′(s_1) = 0 ∧ isPassed′(s_1)= 2 ∧ isRlsdToken′(s_1) = 2∧ a$_1$′(s_1)=a$_1$(s$_1$) ∧ . . . ∧ a$_n$′(s_1)=a$_n$(s$_1$) ∧ sToken′(s_2)=0∧ comm′(s_2)=null∧ commType′(s_2)=null∧ isAuth$_z$ ′(s_2)= 2∧ isPassed′(s_2)= 2 ∧ isRlsdToken′(s_2) = 2∧ a$_1$′(s_2)=a$_1$(s$_2$)∧ . . . ∧ a$_n$′(s_2)=a$_n$(s$_2$)
Do Create
ConstrPermitCreate($s_1, s_2, savt_1, savt_2$) ≡ sToken(s$_1$)=1 ∧ comm(s$_1$)=comm(s_1)∧ commType(s$_1$)= Create∧ isAuth$_z$(s$_1$) = 1∧ isPassed(s$_1$)= 0∧ isRlsdToken(s$_1$) = 0 ∧ sToken′(s$_1$)=1 ∧ comm′(s$_1$)=comm(s_1) ∧ commType′(s$_1$)= Modify ∧ isAuth$_z$′(s$_1$) = 1 ∧ isPassed′(s$_1$)= 1 ∧ isRlsdToken′(s$_1$) = 0 ∧ sToken′(s$_2$)=1 ∧ comm′(s$_2$)=comm(s_1)∧ commType′(s$_2$)= Create ∧ isAuth$_z$′(s$_2$) = 1 ∧ isPassed′(s$_2$)= 1 ∧ isRlsdToken′(s$_2$) = 0

- **ConstrPermitDelete(**s_1**,**s_2**,**$savt_1$**)**Check the state attribute of an authorized deleting command and permit access and do delete (isAuthorized= 1, isPassed=1, isRlsdToken = 0)

4.2.3 Safety and Expressive Power

$\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ Scheme

An $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ scheme consists of $\langle \Gamma^{\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}}, \Psi^{\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}}, Q^{\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}}, \vdash^{\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}} \rangle$. Where

- $\Gamma^{\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}}$ is the set of all states. Where each state $\gamma^{\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}} \in \Gamma^{\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}}$ is characterized by $\langle$ U$_\gamma$, S$_\gamma$, O$_\gamma$, UA, SA, OA, uavtf, savtf, oavtf, P, SubCreator$\rangle$ where U$_\gamma$, S$_\gamma$, O$_\gamma$ are set of users, subjects objects respectively in state γ.

Table 4.17: Configuration of Constraints: for Delete(Checking authorization, Delete)

Check whether Deletion is authorized
ConstrCheckDelPos($s_1, s_2, savt_1, savt_2$) ≡ sToken(s_1)=1 ∧ comm(s_1)=comm(s_1)∧ commType(s_1)= Delete∧ isAuth$_z$(s_1) = 2∧ isPassed= 2∧ isRlsdToken(s_1) = 2) ∧ sToken(s_2)=0 ∧ comm(s_2) = null ∧ commType(s_2)= null ∧ isAuth$_z$(s_2) = 2 ∧ isPassed(s_2)= 2 ∧ isRlsdToken(s_2)= 2) ∧ (DelPC) ∧ sToken′(s_1)=1 comm′(s_1)=comm(s_1), commType′(s_1)= Delete, isAuth$_z$′(s_1) = 1 ∧ isPassed′(s_1)= 0 ∧ isRlsdToken′(s_1) = 0 ∧ a$_1$′(s_1)=a$_1$(s$_1$) ∧ . . . a′$_n$(s_1)=a$_n$(s$_1$) ∧ sToken′(s_2)=1∧ comm′(s_2)=comm(s_1)∧ commType′(s_2)= Delete∧ isAuth$_z$′(s_2) = 1∧ isPassed′(s_2)= 0 ∧isRlsdToken′(s_2) = 0∧ a$_1$′(s_2)=a$_1$(s$_2$)∧ . . . ∧ a$_n$′(s_2)=a$_n$(s$_2$)
Check Whether Deletion is not authorized
ConstrCheckDelNeg($s_1, s_2, savt_1, savt_2$)≡ sToken(s_1)=1 ∧ comm(s_1)=comm(s_1) ∧ commType(s_1)= Delete ∧ isAuth$_z$(s_1) = 2∧ isPassed(s_1)= 2 ∧ isRlsdToken(s_1) = 2 ∧ sToken(s_2)=0 ∧ comm(s_2) = null ∧ commType(s_2)= null ∧ isAuth$_z$(s_2) = 2 ∧ isPassed(s_2)= 2 ∧ isRlsdToken = 2 ∧ !(DelPC) ∧ sToken′(s_1)=1 ∧ comm′(s_1)=comm(s_1)∧ commType′(s_1)= Delete ∧ isAuth$_z$′(s_1) = 0 ∧ isPassed′(s_1)= 2 ∧ isRlsdToken′(s_1) = 2∧ a$_1$′(s_1)=a$_1$(s$_1$) ∧ . . . ∧ a$_n$′(s_1)=a$_n$(s$_1$) ∧ sToken′(s_2)=0∧ comm′(s_2)=null∧ commType′(s_2)=null∧ isAuth$_z$ ′(s_2)= 2∧ isPassed′(s_2)= 2 ∧ isRlsdToken′(s_2) = 2∧ a$_1$′(s_2)=a$_1$(s$_2$)∧ . . . ∧ a$_n$′(s_2)=a$_n$(s$_2$)
Do Delete
ConstrPermitDelete($s_1, s_2, savt_1$) ≡ sToken(s$_1$)=1 ∧ comm(s$_1$)=comm(s_1)∧ commType(s$_1$)= Delete∧ isAuth$_z$(s$_1$) = 1∧ isPassed(s$_1$)= 0∧ isRlsdToken(s$_1$) = 0 ∧ sToken(s$_2$)=1∧ comm(s$_2$) = comm(s_1)∧ commType(s$_2$)= Delete ∧ isAuth$_z$(s$_2$) = 1∧ isPassed(s$_2$)= 0 ∧ isRlsdToken(s$_2$) = 0 ∧ sToken′(s$_1$)=1 ∧ comm′(s$_1$)=comm(s_1) ∧ commType′(s$_1$)= Modify ∧ isAuth$_z$′(s$_1$) = 1 ∧ isPassed′(s$_1$)= 1 ∧ isRlsdToken′(s$_1$) = 0

- $\Psi^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$ is the set of state transition rules which are all $\mathrm{ABAC}_\alpha^{\mathrm{AM}}$ operations defined in Table 4.6.

- $\mathrm{Q}^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$ is the set of queries of type:

  1. Authorization$_p$(s, o)

     for p∈ $\mathrm{P}^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$, $s \in \mathrm{S}^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$, $o \in \mathrm{O}^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$.

  2. AuthorizationonSubject$_p$(s_1, s_2)

     for p∈ $\mathrm{P}^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$, $s_1 \in \mathrm{S}^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$, $s_2 \in \mathrm{S}^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$.

  3. ConstrSubCreatebyUser($u, s, savt$)

for $u \in \text{U}^{\text{ABAC}_\alpha^{\text{AM}}}$, $s \notin \text{S}^{\text{ABAC}_\alpha^{\text{AM}}}$, $savt \in \text{SAVT}^{\text{ABAC}_\alpha^{\text{AM}}}$.

4. ConstrSubCreatebySub(s_1, s_2, $savt_1$, $savt_2$)

   for $s_1 \in \text{S}^{\text{ABAC}_\alpha^{\text{AM}}}$, $s_2 \notin \text{S}^{\text{ABAC}_\alpha^{\text{AM}}}$, $savt_1 \in \text{SAVT}^{\text{ABAC}_\alpha^{\text{AM}}}$, $savt_2 \in \text{SAVT}^{\text{ABAC}_\alpha^{\text{AM}}}$.

5. ConstrSubModbyUser(u,s,$uavt$, $savt$)

   for $u \in \text{U}^{\text{ABAC}_\alpha^{\text{AM}}}$, $s \in \text{S}^{\text{ABAC}_\alpha^{\text{AM}}}$, $uavt \in \text{UAVT}^{\text{ABAC}_\alpha^{\text{AM}}}$, $savt \in$ SA- $\text{VT}^{\text{ABAC}_\alpha^{\text{AM}}}$.

6. ConstrSubModbySub(s_1,s_2,$savt_1$, $savt_2$)

   for $s_1 \in \text{S}^{\text{ABAC}_\alpha^{\text{AM}}}$, $s_2 \in \text{S}^{\text{ABAC}_\alpha^{\text{AM}}}$, $savt_1 \in \text{SAVT}^{\text{ABAC}_\alpha^{\text{AM}}}$, $savt_2 \in \text{SAVT}^{\text{ABAC}_\alpha^{\text{AM}}}$.

7. ConstrObjCreatebySub($s, o, oavt$)

   for $s \in \text{S}^{\text{ABAC}_\alpha^{\text{AM}}}$, $o \notin \text{O}^{\text{ABAC}_\alpha^{\text{AM}}}$, $oavt \in \text{OAVT}^{\text{ABAC}_\alpha^{\text{AM}}}$.

8. ConstrObjModbySub($s, o, oavt$)

   for $s \in \text{S}^{\text{ABAC}_\alpha^{\text{AM}}}$, $o \in \text{O}^{\text{ABAC}_\alpha^{\text{AM}}}$, $oavt \in \text{OAVT}^{\text{ABAC}_\alpha^{\text{AM}}}$.

- Entailment $\vdash$ specifies that given a state $\gamma \in \Gamma^{\text{ABAC}_\alpha^{\text{AM}}}$ and a query q $\in \text{Q}^{\text{ABAC}_\alpha^{\text{AM}}}$, $\gamma \vdash$ q if and only if q returns true in state γ.

$\text{UCON}_{\text{preA}}^{\text{finite}}$ Scheme

An $\text{UCON}_{\text{preA}}^{\text{finite}}$ scheme consists of $\langle \Gamma^{\text{UCON}_{\text{preA}}^{\text{finite}}}, \Psi^{\text{UCON}_{\text{preA}}^{\text{finite}}}, Q^{\text{UCON}_{\text{preA}}^{\text{finite}}}, \vdash^{\text{UCON}_{\text{preA}}^{\text{finite}}} \rangle$, as follows.

- $\Gamma^{\text{UCON}_{\text{preA}}^{\text{finite}}}$ is the set of all states. Where each state $\gamma^{\text{UCON}_{\text{preA}}^{\text{finite}}} \in \Gamma^{\text{UCON}_{\text{preA}}^{\text{finite}}}$ is characterized by $\langle OS_\Delta^\gamma, UR, ATT, AVT, avtf \rangle$. Here OS_Δ^γ is the object schema in state γ.

- $\Psi^{\text{UCON}_{\text{preA}}^{\text{finite}}}$ is set of state transition rules which are the set of creating, non-creating and deleting commands of $\text{UCON}_{\text{preA}}^{\text{finite}}$ defined in Table 4.8, 4.9 and 4.10 respectively.

- $\text{Q}^{\text{UCON}_{\text{preA}}^{\text{finite}}}$ is the set of queries and of following types:

  1. CheckPCNCR(Access$_p$, avtf(s), avtf(o), avtf(s), avtf(o)) for $uc_r \in$ UC, $r \in$ UR, s and o are $\text{UCON}_{\text{preA}}^{\text{finite}}$ objects.

2. CheckPCNCR(AccessSubject$_p$, avtf(s), avtf(o), avtf(s), avtf(o)) for $uc_r \in$ UC, $r \in$ UR, s and o are $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ objects.
3. CheckPCNCR(ModifySubjectAttbyUser_ijkl$_d$, avtf(s), avtf(o),$\langle$ user, s.user_name, NULL, NULL, uavt$_1$, ... uavt$_n$,NULL,...,NULL, NULL,..., NULL $\rangle$, $\langle$ savt$_1$, ... savt$_n$ $\rangle$) where i = uavtf(u), j = savtf(s), k = $uavt$, l = $savt$.
4. CheckPCNCR(ModifySubjectAttbySub_ijkl$_d$, avtf(s), avtf(o),$\langle$ subject, NULL, s.SubCreator, savt$_{1_1}$, ... savt$_{1_n}$,NULL,...,NULL, NULL,..., NULL $\rangle$, $\langle$ savt$_1$, ... savt$_n$ $\rangle$) where i = uavtf(u), j = savtf(s), k = $uavt$, l = $savt$.
5. CheckPCNCR(ModifyObjectAttbySubject_ijk$_d$, avtf(s), avtf(o), avtf(s), $\langle$ oavt$_1$, ... oavt$_p$ $\rangle$) where i = savtf(s), j = oavtf(o) and k = $oavt$.
6. CheckPCCR(CreateSubjectbyUser_ij$_d$, avtf(s), o, avtf(s), $\langle$ subject, NULL, u, NULL,..., NULL, savt$_1$, ... savt$_n$, NULL,..., NULL $\rangle$) where i = uavtf(u) and j = savt.
7. CheckPCCR(CreateSubjectbySubject_ijk$_d$, avtf(s), o, $\langle$ subject, NULL, u, savt$_{1_1}$, ... savt$_{1_n}$, NULL,..., NULL $\rangle$, $\langle$ subject, NULL, u, NULL,..., NULL, savt$_{2_1}$, ... savt$_{2_n}$, NULL,..., NULL $\rangle$) where i = uavtf(u), j = savt$_1$ and k = savt$_2$.
8. CheckPCCR(CreateObjectbySubject_ij$_d$, avtf(s), o, avtf(s), $\langle$ object, NULL, NULL, NULL, NULL,..., NULL, NULL,..., NULL, oavt$_1$, ... oavt$_p$ $\rangle$) where i = savtf(s) and j = oavt.
9. CheckPCDelCR(DeleteSubbyUser_ij$_d$, avtf(s), avtf(o)) where i = uavtf(u), j = savtf(s).
10. CheckPCDelCR(DeleteSubbySub_ijk$_d$, avtf(s), avtf(o),$\langle savt_1, \ldots, savt_n \rangle$) where i = uavtf(u), j = savtf(s) and k = savt.

- Entailment $\vdash$ specifies that given a state $\gamma \in \Gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ and a query q $\in \mathrm{Q}^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$, $\gamma \vdash$ q if and only if q returns true in state γ.

Mapping from $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ **to** $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ **(**$\sigma^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$**)**

- Mapping of $\Gamma^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$ to $\Gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$

- Mapping of Object Schema(OS_{Δ}), ATT and UR is provided in Table 4.7

- Mapping of $\Psi^{\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}}$ to $\Psi^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}}$
  - σ(Access$_p$) = Access$_r^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}}$ where r = p.
  - σ(AccessSubject$_p$) = AccessSubject$_r^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}}$ where r = p.
  - σ(CreateSubjectbyUser($u, s, savt$)) = CreateSubjectbyUser_ij$_d$(s,o), i = uavtf(u) and j = $savt$.
  - σ(CreateSubjectbySubject($s_1, s_2, savt_1, savt_2$)) = CreateSubjectbySubject_ijk$_d$(s,o), i = savtf(s_1), j = $savt_1$ and k = $savt_2$.
  - σ(DeleteSubjectbyUser(u, s)) = DeleteSubbyUser_ij$_d$(s, o), i = uavtf(u) and j = savtf(s).
  - σ(DeleteSubjectbySubject($s_1, s_2, savt$)) = DeleteSubbySub_ijk$_d$(s,o), i = savtf(s_1), j = savtf(s_2), k = savt.
  - σ(ModifySubjectAttbyUser($u, s, uavt, savt$)) = ModifySubjectAttbyUser_ijkl$_d$(s,o), i = uavtf(u) and j = savtf(s), k = $uavt$, l =$savt$.
  - σ(ModifySubjectAttbySubject($s_1, s_2, savt_1, savt_2$)) = ModifySubjectAttbySubject_ijkl$_d$(s,o), i = savtf(S_1) and j = savtf(s_2), k = $savt_1$, l =$savt_2$.
  - σ(CreateObjectbySubject($s, o, oavt$)) = CreateObject_ij$_d$(s,o), i = savtf(s) and j = $oavt$.
  - σ(ModifyObjectAttbySubject($s, o, oavt$)) = ModifyObjectAtt_ijk$_d$(s,o), i = savtf(s) and j = oavtf(o) and k = $oavt$.

- Mapping of $Q^{\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}}$ to $Q^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}}$ is provided below
  - σ(Authorization$_p$(s, o)) = CheckPCNCR(Access$_p$, avtf(s), avtf(o), avtf(s), avtf(o)).
  - σ(AuthorizationonSubject$_p$(s, o)) = CheckPCNCR(AccessSubject$_p$, avtf(s), avtf(o), avtf(s), avtf(o)).
  - σ(ConstrSubCreatebyUser($u, s, savt$)) = CheckPCCR(CreateSubjectbyUser_ij$_d$, avtf(s), o, avtf(s), $\langle$ subject, NULL, u, NULL,. . . , NULL, savt$_1$, . . . savt$_n$, NULL,. . . , NULL $\rangle$) where i = uavtf(u) and j = savt.

- σ(ConstrSubCreatebySub($s_1, s_2, savt_1$,)) = CheckPCCR(CreateSubjectbySubject_ijk$_d$, avtf(s), o, $\langle$ subject, NULL, u, savt$_{1_1}$, ... savt$_{1_n}$, NULL,..., NULL $\rangle$, $\langle$ subject, NULL, u, NULL,..., NULL, savt$_{2_1}$, ... savt$_{2_n}$, NULL,..., NULL $\rangle$) where i = uavtf(u), j = savt$_1$ and k = savt$_2$.
- σ(ConstrSubDelbyUser(u, s))= CheckPCDelCR(DeleteSubbyUser_ij$_d$, avtf(s), avtf(o)) where i = uavtf(u), j = savtf(s).
- σ(ConstrSubDelbySub($s_1, s_2, savt$))= CheckPCDelCR(DeleteSubbySub_ijk$_d$, avtf(s), avtf(o),$\langle savt_1, \ldots, savt_n \rangle$) where i = uavtf(u), j = savtf(s) and k = savt.
- σ(ConstrSubModbyUser($u, s, uavt, savt$))= CheckPCNCR(ModifySubjectAttbyUser-_ijkl$_d$, avtf(s), avtf(o),$\langle$ user, s.user_name, NULL, NULL, uavt$_1$, ... uavt$_n$,NULL, ..., NULL, NULL, ..., NULL $\rangle$, $\langle$ savt$_1$, ... savt$_n$ $\rangle$) where i = uavtf(u), j = savtf(s), k = $uavt$, l = $savt$.
- σ(ConstrSubModbySub($s_1, s_2, savt_1, savt_2$))= CheckPCNCR(ModifySubjectAttbySub-_ijkl$_d$, avtf(s), avtf(o),$\langle$ subject, NULL, s.SubCreator, savt$_{1_1}$, ... savt$_{1_n}$,NULL,...,NULL, NULL,..., NULL $\rangle$, $\langle$ savt$_1$, ... savt$_n$ $\rangle$) where i = uavtf(u), j = savtf(s), k = $uavt$, l = $savt$.
- σ(ConstrObjCreatebySub($s, o, oavt$)) = CheckPCCR(CreateObjectbySubject_ij$_d$, avtf(s), o, avtf(s), $\langle$ object, NULL, NULL, NULL, NULL,..., NULL, NULL,..., NULL, oavt$_1$, ... oavt$_p$ $\rangle$) where i = savtf(s) and j = oavt.
- σ(ConstrObjModbySub($s, o, oavt$))= CheckPCNCR(ModifyObjectAttbySubject_ijk$_d$, avtf(s), avtf(o), avtf(s), $\langle$ oavt$_1$, ... oavt$_p$ $\rangle$) where i = savtf(s), j = oavtf(o) and k = $oavt$.

Mapping from $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ **to** $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ **(**$\sigma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$**)**

- Mapping of $\Gamma^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$ to $\Gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$
  - Mapping of UA, SA, OA, UAVT, SAVT, OAVT and P defined in Table 4.11.

Table 4.18: A Sequence of Actions of $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ to Configure the $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ Creating Command uc_r(s, o).

State Transition Actions	$\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ Operations
tryaccess(Give Source Token)	ModifySubjectAttbyUser(u,s, $\langle uToken = 1\rangle$, $\langle sToken = 1, comm = uc_r, commType = Create, newVal_1 = \langle f_{1,a_1}^{uc_r}(s), \ldots f_{1,a_n}^{uc_r}(s)\rangle, newVal_2 = \langle f_{2,a_1}^{uc_r}(s), \ldots f_{2,a_n}^{uc_r}(s)\rangle, isAuth_z = 2, isPassed = 2, isRlsdToken = 2\rangle$).
checkCreate(Check PreCondition if Authorized)	ModifySubjectAttBySubject(s_1, s_2, $\langle sToken = 1, comm = comma(s_1), commType = Create, newVal_1 = newVal_1(s_1), newVal_2 = newVal_2(s_1), isAuth_z = 1, isPassed = 0, isRlsdToken = 0, a_1 = a_1(s_1), \ldots, a_n = a_n(s_1)\rangle$, $\langle 1, comm(s_1), commType = Create, newVal_1 = newVal_1(s_1), newVal_2 = newVal_2(s_1), isAuth_z = 1, isPassed = 0, isRlsdToken = 0, a_1 = a_1(s_2), \ldots, a_n = a_n(s_2)\rangle$)
checkCreate(Check PreCondition if not Authorized)	ModifySubjectAttBySubject(s_1, s_2, $\langle sToken = 1, comm = command(s_1), commType = Create, newVal_1 = newVal_1(s_1), newVal_2 = newVal_2(s_1), isAuth_z = 0, isPassed = 2, isRlsdToken = 2, a_1 = a_1(s_1), \ldots, a_n = a_n(s_1)\rangle$, $\langle 1, comm(s_1), commType = Create, newVal_1 = newVal_1(s_1), newVal_2 = newVal_2(s_1), isAuth_z = 2, isPassed = 2, isRlsdToken = 2, a_1 = a_1(s_2), \ldots, a_n = a_n(s_2)\rangle$)
doCreate (Create and Access if Authorized)	CreateSubjectBySubject(s_1, s_2, $\langle sToken = 1, comm = comm(s_1), commType = Create, newVal_1 = newVal_1(s_1), newVal_2 = newVal_2(s_1), isAuth_z = 1, isPassed = 1, isRlsdToken = 0, a_1 = avt_1.a_1, \ldots .avt_1.a_n\rangle$, $\langle sToken = 1, comm = comm(s_1), commType = Create, newVal_1 = newVal_1(s_1), newVal_2 = newVal_2(s_1), isAuth_z = 1, isPassed = 1, isRlsdToken = 0, a_1 = avt_2.a_1, \ldots .avt_2.a_n\rangle$)
returnTargetToken(Release Target Token)	ModifySubjectAttBySubject(s_1, s_2,$\langle sToken = 1, comm = comm(s_1), commType = commType(s_1), newVal_1 = newVal_1(s_1), newVal_2 = newVal_2(s_1), isAuth_z = isAuth_z(s_1), isPassed = 1, isRlsdToken = 0, a_1 = a_1(s_1), \ldots, a_n = a_n(s_1)\rangle$, $\langle sToken = 0, comm = null, commType = null, newVal_1 = null, newVal_2 = null, isAuth_z = 2, isPassed = 2, isRlsdToken = 2, a_1 = a_1(s_2), \ldots, a_n = a_n(s_2)\rangle$)
denied/endAccess(Release Source Token)	ModifySubjectAttbyUser(u,s, $\langle uToken = 0\rangle$, $\langle sToken = 0, comm = null, commType = null, newVal_1 = null, newVal_2 = null, isAuth_z = 2, isPassed = 2, isRlsdToken = 2, a_1 = a_1(s), \ldots, a_n = a_n(s)\rangle$)

  - Mapping of U, S, O and the initial state γ_0 is defined in Section 4.2.2

- Mapping of $\Psi^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}}$ to $\Psi^{\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}}$

  - $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ non-creating command uc_r(s,o) is same as shown in Table 4.13.
  - Sequence of $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ operations that map $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ creating command uc_r(s,o) provided in Table 4.19.
  - Sequence of $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ operations that map $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ deleting command uc_r(s,o) provided in Table 4.18.

- Mapping of $Q^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}}$ to $Q^{\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}}$ is provided below

Table 4.19: A Sequence of Actions in $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ to Configure the $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ Deleting Command uc_r(s, o).

tryaccess(Give Source Token)	ModifySubjectAttbyUser(u,s, $\langle uToken = 1\rangle$, $\langle sToken = 1, comm = uc_r, commType = Delete, newVal_1 = \langle f_{1,a_1}^{uc_r}(s,o), \ldots f_{1,a_n}^{uc_r}(s,o)\rangle, newVal_2 = null, isAuth_z = 2, isPassed = 2, isRlsdToken = 2\rangle$)
checkDelete(Check PreCondition if Authorized)	ModifySubjectAttBySubject(s_1, s_2, $\langle sToken = 1, comm = comma(s_1), commType = Delete, newVal_1 = newVal_1(s_1), newVal_2 = newVal_2(s_1), isAuth_z = 1, isPassed = 0, isRlsdToken = 0, a_1 = a_1(s_1), \ldots, a_n = a_n(s_1)\rangle$, $\langle 1, comm(s_1), commType = Delete, newVal_1 = newVal_1(s_1), newVal_2 = newVal_2(s_1), isAuth_z = 1, isPassed = 0, isRlsdToken = 0, a_1 = a_1(s_2), \ldots, a_n = a_n(s_2)\rangle$)
checkDelete(Check PreCondition if not Authorized)	ModifySubjectAttBySubject(s_1, s_2, $\langle sToken = 1, comm = command(s_1), commType = Delete, newVal_1 = newVal_1(s_1), newVal_2 = newVal_2(s_1), isAuth_z = 0, isPassed = 2, isRlsdToken = 2, a_1 = a_1(s_1), \ldots, a_n = a_n(s_1)\rangle$, $\langle 1, comm(s_1), commType = Delete, newVal_1 = newVal_1(s_1), newVal_2 = newVal_2(s_1), isAuth_z = 2, isPassed = 2, isRlsdToken = 2, a_1 = a_1(s_2), \ldots, a_n = a_n(s_2)\rangle$)
doDelete (Delete and Access if Authorized)	DeleteSubjectBySubject(s_1, s_2, $\langle sToken = 1, comm = comm(s_1), commType = Delete, newVal_1 = newVal_1(s_1), newVal_2 = newVal_2(s_1), isAuth_z = 1, isPassed = 1, isRlsdToken = 0, a_1 = avt_1.a_1, \ldots .avt_1.a_n\rangle$)
returnTargetToken(Release Target Token)	ModifySubjectAttBySubject(s_1, s_2,$\langle sToken = 1, comm = comm(s_1), commType = commType(s_1), newVal_1 = newVal_1(s_1), newVal_2 = newVal_2(s_1), isAuth_z = isAuth_z(s_1), isPassed = 1, isRlsdToken = 0, a_1 = a_1(s_1), \ldots, a_n = a_n(s_1)\rangle$, $\langle sToken = 0, comm = null, commType = null, newVal_1 = null, newVal_2 = null, isAuth_z = 2, isPassed = 2, isRlsdToken = 2, a_1 = a_1(s_2), \ldots, a_n = a_n(s_2)\rangle$)
denied/endAccess(Release Source Token)	ModifySubjectAttbyUser(u,s, $\langle uToken = 0\rangle$, $\langle sToken = 0, comm = null, commType = null, newVal_1 = null, newVal_2 = null, isAuth_z = 2, isPassed = 2, isRlsdToken = 2, a_1 = a_1(s), \ldots, a_n = a_n(s)\rangle$)

- σ(CheckPCNCR(Access$_p$, avtf(s), avtf(o), avtf(s), avtf(o))) = Authorization$_p(s, o)$ where $\langle a_1(s), \ldots a_n(s)\rangle$ = avtf(s) and $\langle a_1(s), \ldots a_n(s)\rangle$ = avtf(o) .
- σ (CheckPCNCR(AccessSubject$_p$, avtf(s), avtf(o), avtf(s), avtf(o))) = AuthorizationonSubject$_p(s_1, s_2)$ where $\langle a_1(s_1), \ldots a_n(s_1)\rangle$ = avtf(s) and $\langle a_1(s_2), \ldots a_n(s_2)\rangle$ = avtf(o
- σ(CheckPCCR(CreateSubjectbyUser_ij$_d$, avtf(s), o, avtf(s), $\langle$ subject, NULL, u, NULL,. . . , NULL, savt$_1$, . . . savt$_n$, NULL,. . . , NULL $\rangle$))= ConstrSubCreatebyUser($u, s, savt$)) where i = uavtf(u)= i and savt= j.
- σ(CheckPCCR(CreateSubjectbySubject_ijk$_d$, avtf(s), o, $\langle$ subject, NULL, u, savt$_{1_1}$, . . . savt$_{1_n}$, NULL,. . . , NULL $\rangle$, $\langle$ subject, NULL, u, NULL,. . . , NULL, savt$_{2_1}$, . . . savt$_{2_n}$, NULL,. . . , NULL $\rangle$))= ConstrSubCreatebySub($s_1, s_2, savt_1$,)), where uavtf(u) = i, savt$_1$ = j and savt$_2$ = k.

- σ(CheckPCDelCR(DeleteSubbyUser_ij$_d$, avtf(s), avtf(o))) = ConstrSubDelbyUser(u, s)), where uavtf(u) = i, savtf(s) = j.
- σ(CheckPCDelCR(DeleteSubbySub_ijk$_d$, avtf(s), avtf(o),$\langle savt_1, \ldots, savt_n \rangle$)) = ConstrSubDelbySub($s_1, s_2, savt$)), where i = uavtf(u), j = savtf(s) and k = savt.
- σ(CheckPCNCR(ModifySubjectAttbyUser_ijkl$_d$, avtf(s), avtf(o),$\langle$ user, s.user_name, NULL, NULL, uavt$_1$, ... uavt$_n$,NULL,...,NULL, NULL,..., NULL $\rangle$, $\langle$ savt$_1$, ... savt$_n$ $\rangle$)) = ConstrSubModbyUser($u, s, uavt, savt$)), where uavtf(u) = i, savtf(s) = j, $uavt$ = k, $savt$ = l.
- σ(CheckPCNCR(ModifySubjectAttbySubject_ijkl$_d$, avtf(s), avtf(o),$\langle$ subject, NULL, s.SubCreator, savt$_{1_1}$, ... savt$_{1_n}$,NULL,...,NULL, NULL,..., NULL $\rangle$, $\langle$ savt$_1$, ... savt$_n$ $\rangle$)) = ConstrSubModbySub($s_1, s_2, savt_1, savt_2$)), where uavtf(u) = i, savtf(s) = j, $uavt$ = k, $savt$ = l.
- σ(CheckPCCR(CreateObjectbySubject_ij$_d$, avtf(s), o, avtf(s), $\langle$ object, NULL, NULL, NULL, NULL,..., NULL, NULL,..., NULL, oavt$_1$, ... oavt$_p$ $\rangle$)) = ConstrObjCreatebySub($s, o, oavt$)), where savtf(s) = i and oavt = j.
- σ(CheckPCNCR(ModifyObjectAttbySubject_ijk$_d$, avtf(s), avtf(o), avtf(s), $\langle$ oavt$_1$, ... oavt$_p$ $\rangle$)) = ConstrObjModbySub($s, o, oavt$)), where savtf(s) = i, oavtf(o) = j and $oavt$ = k .

Expressive Power Equivalence and Safety Decidability

The proof that the mappings provided in Section 4.2.3 and Section 4.2.3 are state matching reductions is lengthy and tedious. Here we present an outline of the main argument.

Lemma 3. *$\sigma^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$ satisfies assertion 1 of the state matching reduction of Definition 1.*

Proof. (Sketch): Assertion 1 requires that, for every $\gamma^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} \in \Gamma^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$ and every $\psi^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} \in \Psi^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$, $\langle\, \gamma^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}, \psi^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} \,\rangle = \sigma\,(\langle\, \gamma^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}, \psi^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} \,\rangle)$ has the following property:

For every $\gamma_1^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$ in scheme $\mathrm{ABAC}_\alpha^{\mathrm{AM}}$ such that

$\gamma^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} \xrightarrow{*}_{\psi^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}} \gamma_1^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$,

there exists a state $\gamma_1^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}}$ such that

1. $\gamma^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}}(=\sigma(\gamma^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}})) \xrightarrow{*}_{\psi^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}(=\sigma(\psi^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}))}} \gamma_1^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}}$.
2. for every query $q^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} \in Q^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$, $\gamma_1^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} \vdash^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} q^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$ if and only if $\gamma_1^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}} \vdash^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}} \sigma(q^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}})$. It can be decomposed into two directions:
   (a) The "if" direction:
   $\gamma_1^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}} \vdash^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}} \sigma(q^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}) => \gamma_1^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} \vdash^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} q^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$.
   (b) The "only if" direction:
   $\gamma_1^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} \vdash^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} q^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} => \gamma_1^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}} \vdash^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}} \sigma(q^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}})$.

The proof is by induction on number of steps n in

$\gamma^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} \xrightarrow{*}_{\psi^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}} \gamma_1^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$.

□

Lemma 4. *$\sigma^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$ satisfies assertion 2 of the state matching reduction of Definition 1.*

Proof. (Sketch): Assertion 2 requires that, for every $\gamma^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} \in \Gamma^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$ and every $\psi^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} \in \Psi^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$, $\langle \gamma^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}, \psi^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} \rangle = \sigma(\langle \gamma^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}, \psi^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} \rangle)$ has the following property:

For every $\gamma_1^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}}$ in scheme $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ such that

$\gamma^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}}(=\sigma(\gamma^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}})) \xrightarrow{*}_{\psi^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}(=\sigma(\psi^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}))}} \gamma_1^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}}$,

there exists a state $\gamma_1^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$ such that

1. $\gamma^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} \xrightarrow{*}_{\psi^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}} \gamma_1^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$.
2. for every query $q^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} \in Q^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$, $\gamma_1^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} \vdash^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}} q^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}}$
   if and only if $\gamma_1^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}} \vdash^{\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}} \sigma(q^{\mathrm{ABAC}_\alpha^{\mathrm{AM}}})$.
   It can be decomposed into two directions:

(a) The “if" direction:

$$\gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \vdash_{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}} \sigma(q^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}) => \gamma_1^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}} \vdash_{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}} q^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}.$$

(b) The “only if" direction:

$$\gamma_1^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}} \vdash_{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}} q^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}} => \gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \vdash_{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \sigma(q^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}).$$

The proof is by induction on number of steps n in

$$\gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} (=\sigma(\gamma^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}})) \xrightarrow[\psi^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}(=\sigma(\psi^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}))]{*} \gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}.$$

□

Theorem 11. $\sigma^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$ *is a state matching reduction.*

Proof. Lemma 3 shows that $\sigma^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$ satisfies assertion 1 of Definition 1 and lemma 4 shows that $\sigma^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$ satisfies assertion 2 of Definition 1. Thereby $\sigma^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$ is a state matching reduction. □

Lemma 5. $\sigma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ *satisfies assertion 1 of the state matching reduction of Definition 1.*

Proof. (Sketch): Assertion 1 requires that, for every $\gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \in \Gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ and every $\psi^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \in \Psi^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$, $\langle\, \gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}, \psi^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \,\rangle = \sigma\,(\langle\, \gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}, \psi^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \,\rangle)$ has the following property:

For every $\gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ in scheme $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ such that
$\gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \xrightarrow[\psi^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}]{*} \gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$,
there exists a state $\gamma_1^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$ such that

1. $\gamma^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}(=\sigma(\gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}})) \xrightarrow[\psi^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}(=\sigma(\psi^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}))]{*} \gamma_1^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$.
2. for every query $q^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \in Q^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$, $\gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \vdash_{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} q^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ if and only if $\gamma_1^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}} \vdash_{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}} \sigma(q^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}})$. It can be decomposed into two directions:

   (a) The “if" direction:

   $$\gamma_1^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}} \vdash_{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}} \sigma(q^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}) => \gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \vdash_{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} q^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}.$$

   (b) The “only if" direction:

   $$\gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \vdash_{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} q^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} => \gamma_1^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}} \vdash_{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}} \sigma(q^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}).$$

The proof is by induction on number of steps n in

$\gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \xrightarrow[\psi^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}]{*} \gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$.

□

Lemma 6. *$\sigma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ satisfies assertion 2 of the state matching reduction of Definition 1.*

Proof. (Sketch): Assertion 2 requires that, for every $\gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \in \Gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ and every $\psi^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ $\in \Psi^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$, $\langle\, \gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}, \psi^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \,\rangle = \sigma\,(\langle\, \gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}, \psi^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \rangle)$ has the following property:

For every $\gamma_1^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$ in scheme $\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}$ such that
$\gamma^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}(=\sigma(\gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}})) \xrightarrow[\psi^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}(=\sigma(\psi^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}))]{*} \gamma_1^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$,
there exists a state $\gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ such that

1. $\gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \xrightarrow[\psi^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}]{*} \gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$.
2. for every query $q^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \in Q^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$, $\gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \vdash^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} q^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$
   if and only if $\gamma_1^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}} \vdash^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}} \sigma(q^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}})$.

   It can be decomposed into two directions:

   (a) The "if" direction:
   $\gamma_1^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}} \vdash^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \sigma(q^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}) => \gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \vdash^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} q^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$.

   (b) The "only if" direction:
   $\gamma_1^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} \vdash^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} q^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}} => \gamma_1^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}} \vdash^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}} \sigma(q^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}})$.

The proof is by induction on number of steps n in

$\gamma^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}(=\sigma(\gamma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}})) \xrightarrow[\psi^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}(=\sigma(\psi^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}))]{*} \gamma_1^{\mathrm{ABAC}^{\mathrm{AM}}_{\alpha}}$.

□

Theorem 12. *$\sigma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ is a state matching reduction.*

Proof. Lemma 5 shows that $\sigma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ satisfies assertion 1 of Definition 1 and lemma 6 shows that $\sigma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ satisfies assertion 2 of Definition 1. Thereby $\sigma^{\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}}$ is a state matching reduction.

□

Theorem 13. $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ *and* $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ *are equivalent in expressive power*

Proof. Theorem 11 proves that there exists a state matching reduction from $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ to $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ and Theorem 12 proves that there exists a state matching reduction from $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ to $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$. Thereby $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ and $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ are equivalent in expressive power. □

Theorem 14. *Safety of* $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ *is decidable.*

Proof. Safety of $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ is decidable [107]. Theorem 13 proves that $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ is equivalent in to $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ in expressive power. So safety of $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ is also decidable. □

4.3 A Safety Undecidable ABAC_{α} Enhancement

4.3.1 Extension of ABAC_{α} beyond decidability

$\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ is a $\mathrm{UCON}_{\mathrm{preA}}^{\mathrm{finite}}$ equivalent extension of ABAC_{α} which is still decidable. To analyze the decidability boundary we furhter extend $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ to get an undecidable model. In this section we allow infinite domain entity attribute in $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ and we name the resulting model $\mathrm{ABAC}_{\alpha}^{\mathrm{MI}}$.

Safety Analysis of $\mathrm{ABAC}_{\alpha}^{\mathrm{MI}}$

Here we provide a detail construction of a general Turing Machine with one dimensional single tape and show that the safety problem of $\mathrm{ABAC}_{\alpha}^{\mathrm{MI}}$ can be reduced to the well known undecidable problem of whether a Turing machine would reach to its accept state starting from an initial state and finally prove that the safety problem of $\mathrm{ABAC}_{\alpha}^{\mathrm{MI}}$ is undecidable.

4.3.2 Turing Machine

A general Turing machine with one dimensional single tape [96] $\mathcal{M}$ is a 6 tuple:{Q,Σ,δ,q_0, q_{accept}, q_{reject} }, where:

- Q is a finite set of states,
- Σ is a finite set, alphabet with *blank*

- $\delta : Q \times \Sigma \longrightarrow Q \times \Sigma \times \{L, R\}$ is the transition function,
- q_0, q_{accept}, $q_{reject} \in Q$ are the start state, accept state, and reject state, respectively, where $q_{accept} \neq q_{reject}$

The movement of the head in the tape in described as below:

- $\delta(q, x)$ = (p,y,L) in state q the tape head searching for the cell containing x and the head write y on that cell moves one cell to the left on the tape and the new state should be named as p. If the tape head is at the left end no movement will occur. Left transition can be of two types:
  1. Left transition when head pointing to the left end as it is a one way tape no creation will occur resulting this transition. Only cell content and state will change
  2. Left transition when head not pointing to the leftend, modifies the current cell, and move the head to immediate left cell and put a new state value for that cell.
- $\delta(q, x)$ = (p,y,R) same as above only moves right.

  Right transition can be of two types:
  1. Right transition when head pointing to the right end new cell should be created to move the head right.
  2. Right transition when head not pointing to the right end modifies the current cell, and move the head to immediate right cell and put a new state value for that cell.

4.3.3 Configuration of Turing Machine with $\mathrm{ABAC}_{\alpha}^{\mathrm{MI}}$

We construct an $\mathrm{ABAC}_{\alpha}^{\mathrm{MI}}$ system that simulate Turing Machine $\mathcal{M}$ defined above. Our construction follows the same technique provided in [140] for construction of Turing Machine with $\mathrm{UCON}_{\mathrm{preA}}$. Table 4.20 gives the configuration of basic sets, functions and policies to configure the construction. There are 4 subject attributes: $\{state, cell, right, left\}$ and no user or object attributes for this construction. The value of attribute $state$ for a subject is either null or the state

of $\mathcal{M}$ if its head is positioned on this cell, the value of $cell$ is the content in the $cell$ that the head is scanning. $right$ and $left$ are the entity attributes where the value is the identity of another subject representing the right side cell or left side cell of the concerned subject. For the subject representing the rightmost cell of the tape $right$ is null and for the subject representing the leftmost cell in the tape the value of attribute $left$ would be null. Initial state of the system contains a single user (u) and a single subject (s_1) and the attribute value assignment for the subject is defined below:

- U = $\{u\}$
- S = $\{s_1\}$
- O = $\{\}$
- state(s_1)= q_0
- cell(s_1) = blank
- right(s_1) = null
- left(s_1) = null

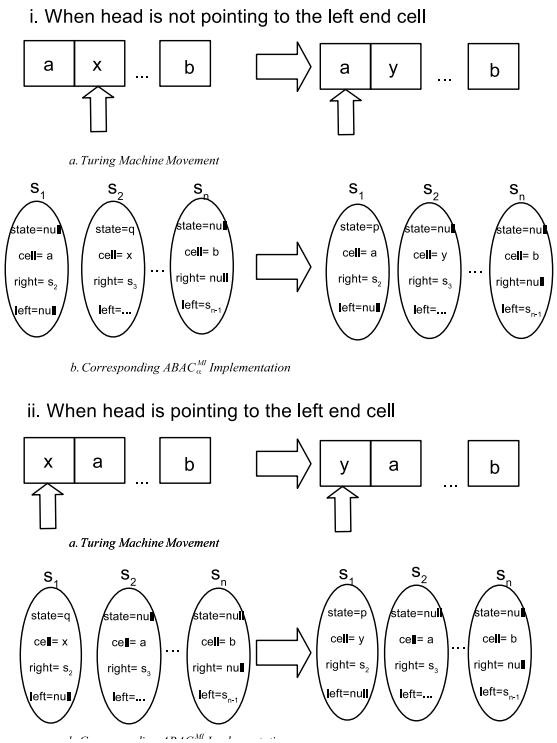
Configuration of Movement:

The movement of the head in the tape is configured with $\mathrm{ABAC}_{\alpha}^{\mathrm{MI}}$ operations. The policy configuration is shown in Table 4.20. Here is the example of how

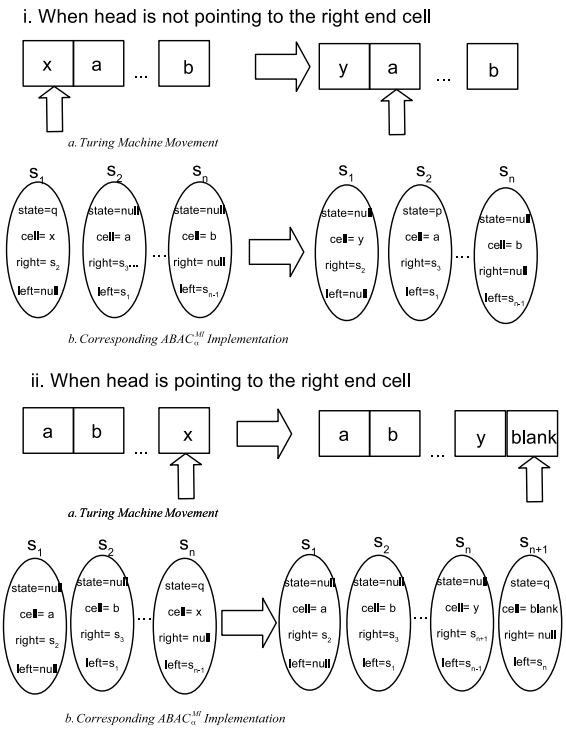
- Left Movement $\delta(q, x)$ = (p,y,L):
  - Head not pointing to the left end cell: policy needs to check that head is not pointing to the leftmost cell, that means to check $state(s_1)$ = q, $cell(s_1)$ = x and $left(s_1)$!=null. It also needs to check s_2 positioned on immediate left of s_1 which means to check $left(s_1)$ = s_2. Then it simulates the movement using the subject attribute modification operation ModifySubjectAttbySubject(s_1, s_2, $\langle\ null$, y, $right(s_1)$, $left(s_1)\ \rangle$, $\langle\ p$, $cell(s_2)$, $right(s_2)$, $left(s_2)\ \rangle$).

Table 4.20: Turing Machine ($\mathcal{M}$) with $\mathrm{ABAC}_{\alpha}^{\mathrm{MI}}$

UA = { }, SA= {state, cell, left, right}, OA = { }
attType(state) = attType (cell) = attType(left) = attType(right) = atomic
Range (state) = Q, Range (cell) = Γ , Range (left) = Range (right)= S
P = Q $\cup$ {leftMove, rightMove, create}
Authorization Policy:
Authorization$_p(u, s) \equiv$ false
Authorization$_p(u, s, savt) \equiv$ false
Subject Creation Policy:
ConstrSubCreatebyUser$(u, s, savt) \equiv$ false
/* right movement $\delta(q, x)$ = (p,y,R)*, head in rightmost cell*/
ConstrSubCreatebySub$(s_1, s_2, savt_1, savt_2) \equiv$ (right(s_1) = null $\wedge$ state(s_1)= q $\wedge$ cell(s_1) = x $\wedge$ right$'(s_1)$
= s_2 $\wedge$ state(s_1)= null $\wedge$ cell(s_1) = y $\wedge$
right$'(s_2)$ = null $\wedge$ state(s_2)= p $\wedge$ cell(s_2) = *blank*)
$\vee$
$\vdots$
Subject Attribute Modification Policy:
ConstrSubModbyUser$(u, s, uavt, savt) \equiv$ false
ConstrSubModbySub$(s_1, s_2, savt_1, savt_2) \equiv$
/* right movement $\delta(q, x)$ = (p,y,R)*, head not in rightmost cell*/
(right(s_1)= s_2 $\wedge$ state(s_1) = q $\wedge$ cell(s_1) = x $\wedge$
left(s_2) = s_1 $\wedge$ state$'(s_1)$ = null $\wedge$ state$'(s_2)$ = p$\wedge$ cell$'(s_1)$ = y)
$\vee$
$\vdots$
$\vee$
/*left movement $\delta(q, x)$ = (p,y,L), head not in leftmost cell*/
(left(s_1)= s_2 $\wedge$ state(s_1) = q $\wedge$ cell(s_1) = x $\wedge$
right(s_2) = s_1 $\wedge$ state$'(s_1)$ = null $\wedge$ state$'(s_2)$ = p$\wedge$ cell$'(s_1)$ = y)
$\vee$
$\vdots$
$\vee$
/*left movement $\delta(q, x)$ = (p,y,L), head in leftmost cell*/
(left(s_1)= null $\wedge$ state(s_1) = q $\wedge$ cell(s_1) = x $\wedge$
state$'(s_1)$ = null $\wedge$ state$'(s_1)$ = p $\wedge$ cell$'(s_1)$ = y)
$\vee$
$\vdots$
$\vee$
Subject Deletion Policy:
ConstrSubDelbyUser$(u, s) \equiv$ false
ConstrSubDelbySub$(u, s, savt) \equiv$ false
Object Creation Policy:
ConstrObjCreatebySub$(s, o, oavt) \equiv$ false
Object Attribute Modification Policy
ConstrObjModAttrbySub$(s, o, oavt) \equiv$ false



(a) Left Movement Simulation of Turing Machine with $\mathrm{ABAC}_\alpha^{\mathrm{MI}}$



(b) Left Movement Simulation of Turing Machine with $\mathrm{ABAC}_\alpha^{\mathrm{MI}}$

Figure 4.5: Simulation of Turing Machine Movement with $\mathrm{ABAC}_\alpha^{\mathrm{MI}}$

– Head pointing to the right end cell: policy needs to check that head is pointing to the leftmost cell, that means to check $state(s_1)$ = q, $cell(s_1)$ = x and $left(s_1)$=null. simulate the movement using the subject attribute modification operation ModifySubjectAttbySubject (s_1, s_1, $\langle\ p, y, right(s_1), left(s_1)\ \rangle$, $\langle\ p, y, right(s_1), left(s_1)\ \rangle$)

- Right Movement $\delta(q, x)$ = (p,y,R):

  – Head not pointing to the right end cell: policy needs to check that head is not pointing to the rightmost cell, that means to check $state(s_1)$ = q, $cell(s_1)$ = x and $right(s_1)$!=null.It also needs to check s_2 positioned on immediate right of s_1 which means to check $right(s_1) = s_2$. Then it simulates the movement using the subject attribute modification operation ModifySubjectAttbySubject(s_1, s_2, $\langle\ null, y, right(s_1), left(s_1)\ \rangle$, $\langle\ p, cell(s_2), right(s_2), left(s_2)\ \rangle$).

– Head pointing to the right end cell: policy needs to check that head is pointing to the rightmost cell, that means to check $state(s_1)$ = q, $cell(s_1)$ = x and $right(s_1)$=null. Then it simulates the movement using the subject creation operation CreateSubjectbySubject(s_1, s_2, $\langle\ null, y, right(s_1), left(s_1)\ \rangle$, $\langle\ p, blank, null, s_1\ \rangle$).

Fig. 4.5 shows examples of right and left movement simulation with $\mathrm{ABAC}_{\alpha}^{\mathrm{MI}}$ in different scenarios. 1)Fig. 4.5(a)i. shows the left movement simulation with $\mathrm{ABAC}_{\alpha}^{\mathrm{MI}}$ when head is not pointing to the leftmost cell, 2)Fig. 4.5(a)ii. shows the left movement simulation with $\mathrm{ABAC}_{\alpha}^{\mathrm{MI}}$ when head is pointing to the leftmost cell, 3)Fig. 4.5(b)i. shows the right movement simulation with $\mathrm{ABAC}_{\alpha}^{\mathrm{MI}}$ when head is not pointing to the rightmost cell and 4)Fig. 4.5(b)ii. shows the left movement simulation with $\mathrm{ABAC}_{\alpha}^{\mathrm{MI}}$ when head is pointing to the leftmost cell.

In a particular state of $\mathrm{ABAC}_{\alpha}^{\mathrm{MI}}$ system, only one of the 2 operations (Modify Subject attribute(left movement and for right movement if the head is not on the rightmost cell) or Create subject (right movement if the head is on the rightmost cell))is authorized according to the policy defined above. To distinguish between different condition of attribute modification the policy also checks the position of the head and direction of movement. So actually in any state one of the 4 operations :1) modify subject attribute for left movement when head is not pointing to the leftmost cell , 2) modify subject attribute for left movement when head is pointing to the leftmost cell, 3) modify subject attribute for right movement when head is not pointing to the rightmost cell and 4) create subject for right movement when head is pointing to the rightmost cell, since the state attribute is nonnull only for one subject. Each operation assigns a non-null value to a subject‘s state, and sets another one to null. The $left$ attribute is only null for leftmost cell and $right$ attribute is only null for rightmost cell. Therefore, this $\mathrm{ABAC}_{\alpha}^{\mathrm{MI}}$ system with the above policy configuration can simulate the operations of $\mathcal{M}$.

4.3.4 Safety and Expressive Power

Theorem 15. *Safety of* $\mathrm{ABAC}_{\alpha}^{\mathrm{MI}}$ *is undecidable.*

Proof. (Proof Sketch): For a Turing machine, it is undecidable to check if the state q_{accept} can be

reached from the initial state. Therefore, with the scheme of $\mathrm{ABAC}_{\alpha}^{\mathrm{MI}}$, it is undecidable whether the $state$ attribute of a subject can have the value q_{accept}. This completes our undecidability proof.

□

Theorem 16. $\mathrm{ABAC}_{\alpha}^{\mathrm{MI}}$ *is more expressive than* $\mathrm{UCON}_{\mathrm{preA}}$ *and* $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$

Proof. Proof Sketch: By definition $\mathrm{ABAC}_{\alpha}^{\mathrm{MI}}$ allows infinite domain entity attribute on top of all $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ features. On the other hand $\mathrm{UCON}_{\mathrm{preA}}$ and $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ only supports finite domain attributes. It is trivial to proof that $\mathrm{ABAC}_{\alpha}^{\mathrm{MI}}$ is more expressive than $\mathrm{ABAC}_{\alpha}^{\mathrm{AM}}$ and $\mathrm{UCON}_{\mathrm{preA}}$.

□

Chapter 5: OBJECT-TO-OBJECT RELATIONSHIP BASED ACCESS CONTROL

This chapter proposes a novel relationship based access control model using object-to-object relationships. It also describes a proof-of-concept implementation of this model for relationship based resource sharing in multicloud environment for Openstack object storage Swift.

5.1 OOReBAC Model

In the OSN context, ReBAC typically expresses authorization policy in terms of interpersonal relationship between users. OSN-inspired ReBAC models primarily focus on user-to-user relationships, although some have also considered user-to-resource and resource-to-resource relationships. An OSN has very specific type of resources (photos, comments, notes etc.) which are closely related to users, so it is natural to consider resource relationships in OSNs as occurring through users. However user-independent resource-to-resource (or object-to-object) relationships have been around for decades in information systems. For instance, object-oriented systems maintain inheritance, composition and association relationships among objects, version control systems use derived-from relationships between different versions, and digital content management systems use similar relationships between different media files. To our knowledge no existing ReBAC model considers user-independent generic relationships between objects, as a useful means to express authorization policies. In this chapter we propose a novel Object-to-Object ReBAC model (OOReBAC) which uses object relationships for controlling access to objects. We build a proof-of-concept implementation of OOReBAC using the open source OpenStack cloud platform and specifically its Swift object storage service.

5.1.1 Object-to-Object Relationship-Based Access Control Model Characteristics

In this subsection we discuss the general characteristics of an object-to-object relationship model for access control. To our knowledge this is a first step towards this direction. Hence we will

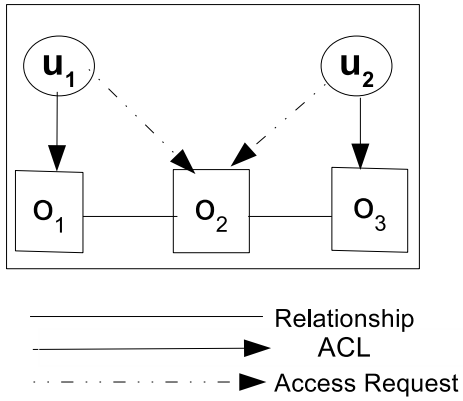


Figure 5.1: Object-to-Object Relationship Based Access Control.

keep our model simple, raising the question as to what are the minimum requirements to realize such a model. A typical access request in any access control model arises when a user (or subject) tries to perform an action on a resource or object. So a set of users, a set of objects and a set of actions are mandatory components for any access control model. Our main focus is on expressing authorization policy considering object relationships, so the model obviously needs a set of possible (binary) relationship types and a data structure (preferably a relationship graph) to store relationships between objects. To keep the model definition simple we will consider only one type of symmetric relationship.

We need a special direct access from a user to object which can be maintained by a system function or access control list (ACL), starting from where additional related objects can be accessed. We propose to limit, in an object specific and action specific manner, the number of relationship links (or hopcount) that can be traversed to access a related object from a given starting point. For example if the system specifies the relationship level of a particular object is 0 for write and 1 for read that means the object is not allowed to be accessed through relationship chain for write, however it allows 1 level relationship chain for read. A system function would specify the relationship level consideration for authorization of a particular object for a particular action.

Figure 5.1 shows how the model relationship and access would work. The system has two users u_1 and u_2, and 3 objects o_1, o_2, o_3. The relationships are $\{\{o_1,o_2\}, \{o_2, o_3\}\}$. The system function ACL takes an object as input and returns a list of users. Here ACL(o_1) = $\{u_1\}$, ACL(o_2) = $\{\}$ and

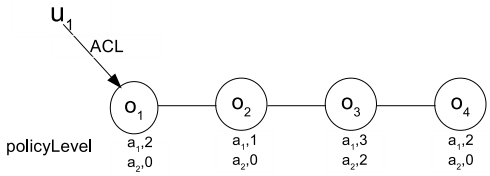


Figure 5.2: Policy Level Example.

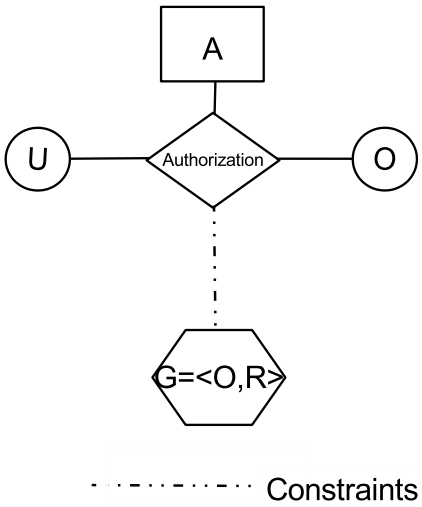


Figure 5.3: OOReBAC Model.

$ACL(o_3) = \{u_2\}$. When user u_1 tries to access o_1 he can directly do that without using relationships. When u_1 tries to access o_2 or o_3 the access control system needs to consider relationship between $\{o_1, o_2\}$ and $\{\{o_1, o_2\}, \{o_2,o_3\}\}$ respectively.

Figure 5.2 shows the policy level specification of objects. Here $ACL(o_1) = \{u_1\}$, $ACL(o_2) = \{\}$, $ACL(o_3) = \{\}$, and $ACL(o_4) = \{\}$. There are two actions in the system, a_1 and a_2. We have the following values of policy level as listed in Figure 5.2.

$$policyLevel(a_1,o_1) = 2,\ policyLevel(a_2,o_1) = 0$$

$$policyLevel(a_1,o_2) = 1,\ policyLevel(a_2,o_2) = 0$$

$$policyLevel(a_1,o_3) = 3,\ policyLevel(a_2,o_3) = 2$$

$$policyLevel(a_1,o_4) = 2,\ policyLevel(a_2,o_4) = 0$$

When u_1 tries to do an action a_1 or a_2 on o_1 the access request would be granted as u_1 is in ACL

Table 5.1: OOReBAC Model

- U is a set of users
- O is a set of objects
- $R \subseteq \{z \mid z \subset O \wedge \mid z \mid = 2\}$
- G=$\langle O, R \rangle$ is an undirected relationship graph with vertices O and edges R
- A is a set of actions
- $P^i(o_1)$ = { o_2 | there exists a simple path of length p in graph G from o_1 to o_2 }
- policyLevel: $O \times A \rightarrow \mathbb{N}$
- ACL: $O \rightarrow 2^U$ is the access control list for each object.
- There is a single policy configuration point.
  Authorization Policy:
  for each action a ∈ A, $Authz_a$(u:U,o:O) is a boolean function which returns true or false where u and o are formal parameters.
- Authorization Policy Language:
  Each action "a" has a single authorization policy $Authz_a$(u:U,o:O) specified using the following language.
  ϕ := u ∈ $PATH_i$
  $PATH_i$:= $ACL(P^0(o)) \cup \ldots \cup ACL(P^i(o))$ where i = min(| O | - 1, policyLevel(a,o))
  where for any set X, $ACL(X) = \bigcup_{x \in X} ACL(x)$

of o_1. When u_1 tries to do action a_1 on o_2 the access would be granted even though u_1 is not in o_2's ACL, since o_2 allows upto 1 level of relationship chaining for authorizing action a_1 and there is a 1 level relationship of o_2 with o_1 as well as u_1 is in o_1's ACL. When u_1 tries to do a_2 on o_2 the authorization would be denied as u_1 is not in o_2's ACL and o_2 allows 0 level relationship chaining for action a_2. When u_1 tries to do a_1 or a_2 on o_3 both of the actions would be granted. On the other hand when u_1 tries to do a_1 or a_2 on o_4 both the actions will be denied.

5.1.2 OOReBAC: Model Definition

In this section we define a model OOReBAC which considers object to object relationships in authorization policy. The model components are as follows: **U** is a set of **users**. A user is a human being who performs action on objects. **O** is a set of **objects**. Objects are resources in the system which need to be protected. **R** is a set of symmetric **relationships** between objects. **G =** $\langle$ **O,**

R $\rangle$ is the relationship graph where objects are nodes and relationship between objects are edges. There is a system function **ACL** which takes an object as input and returns a set of users as output. There is another system function **policyLevel** which takes an object and an action as input and returns a natural number indicating the relationship level that object would allow for authorization of that particular action. **A** is a set of actions. Each action a $\in$ **A** has a single authorization policy **Authz**$_a$**(u:**U**, o:**O**)** which takes u and o as inputs and returns true or false. Here u and o are formal parameters. The authorization policy is a boolean function which considers object relationships, ACL and policyLevel. If Authz$_a$ (u,o) returns true then u is authorized to do action a on object o. On the other hand if Authz$_a$(u,o) returns false then u is not authorized to do action a on o.

Figure 5.3 shows the model components. Table 5.1 shows the formal representation of the model definition and the language for authorization policy. OOReBAC is an operational model. Create/delete users or objects, add/update relationships between objects, configure/update ACL or policy levels are administrative operations and out of scope of OOReBAC model. These would be specified in an administrative model, of which there could be many possibilities.

An instantiation of authorization policy for OOReBAC is given below.

- A = {read, write}
- Authz$_{read}$(u:U,o:O) $\equiv$ u $\in$ P$^{policyLevel(read,o)}$
- Authz$_{write}$(u:U,o:O) $\equiv$ u $\in$ P$^{policyLevel(write,o)}$

An example configuration of OOReBAC and an instantiation of OOReBAC policy is given below.

- U = {u$_1$, u$_2$, u$_3$}
- O = { o$_1$, o$_2$, o$_3$, o$_4$}
- R = {{o$_1$, o$_2$}, {o$_2$, o$_3$}, {o$_3$, o$_4$}}
- ACL(o$_1$) = {u$_1$}

  ACL(o$_2$) = {u$_3$}

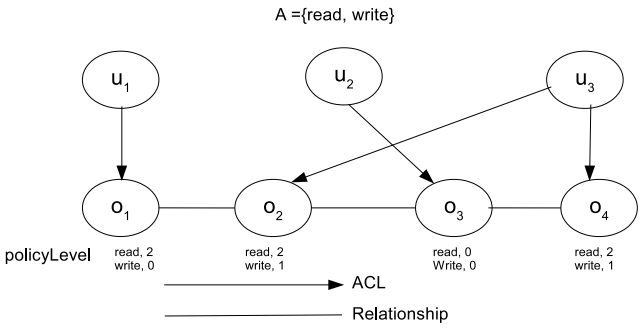


Figure 5.4: An Example of OOReBAC State I_1.

$ACL(o_3) = \{u_2\}$

$ACL(o_4) = \{u_3\}$

- $policyLevel(read, o_1) = 2$

  $policyLevel(write, o_1) = 0$

  $policyLevel(read, o_2) = 2$

  $policyLevel(write, o_2) = 1$

  $policyLevel(read, o_3) = 0$

  $policyLevel(write, o_3) = 0$

  $policyLevel(read, o_4) = 2$

  $policyLevel(write, o_4) = 1$

Figure 5.4 shows an example state I_1 of this system. The following are some actions that different users try in state I_1 and their outcome.

- $read(u_1, o_3)$, $write(u_1, o_3)$ are denied
- $read(u_2, o_1)$ is allowed, $write(u_2, o_1)$ is denied
- $read(u_1, o_4$, $write(u_1, o_4)$ are denied

5.1.3 OOReBAC:Applications

Application of OOReBAC model is restricted to systems where a single symmetric relationship is used, e.g., document co-citation, document clustering, or related medical record. Consider an

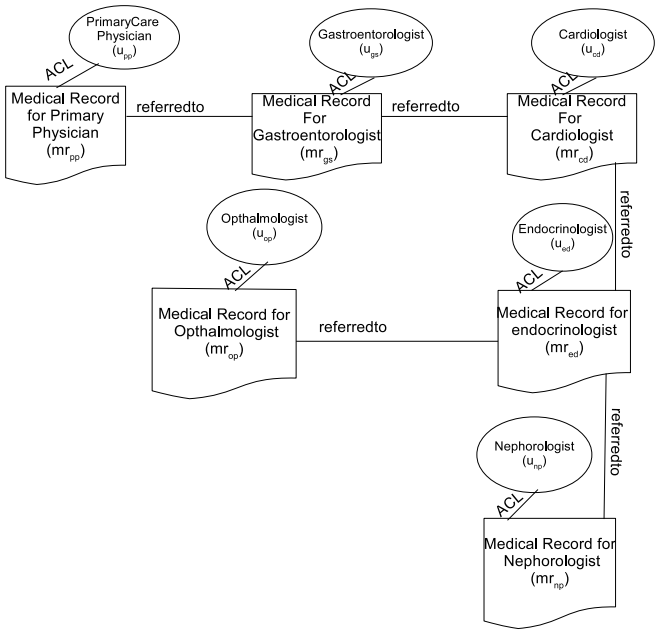


Figure 5.5: Object Relationship in Medical Record.

example of a patient's health records in different specialities shown in Figure 5.5. Here a patient went to his primary care physician with certain symptoms such as chest pain. The primary care physician created a record of his symptoms and medications he was taking at that time and referred him to a gastroenterologist. The gastroenterologist created a record of his symptoms and investigations and based upon the results referred him to a cardiologist. The cardiologist then referred him to an endocrinologist who also referred him to an ophthalmologist and a nephrologist. In every stage of his treatment a new document is created considering the speciality of the doctor treating him, and a relationship between consecutive documents is established. The doctor who creates a particular document has a direct access to that document. Every time a specific doctor treats the patient, he needs to look at his medical history and current treatments by other specialists using the relationship between the records. Figure 5.5 shows the treatment scenario of the patient. For example, if the nephrologist needs to see the records of the gastroenterologist for that patient, he can use the relationship between records to do so.

Consider the policy that every specialist is able to write only on a document for which he is assigned in the document's ACL. Reading a document is allowed through the document relationships. To specify this policy in our OOReBAC model we need to first express the OOReBAC

instantiation of the scenario as follows.

- U = { u_{pp}, u_{gs}, u_{cd}, u_{op}, u_{ed}, u_{np} }
- O = { mr_{pp}, mr_{gs}, mr_{cd}, mr_{op}, mr_{ed}, mr_{np} }
- R = {{mr_{pp}, mr_{gs}}, {mr_{gs}, mr_{cd}}, {mr_{cd}, mr_{ed}}, {mr_{op}, mr_{ed}},{mr_{np}, mr_{ed}}}}
- ACL(mr_{pp}) = {u_{pp}},

  ACL(mr_{gs}) = {u_{gs}},

  ACL(mr_{cd}) = {u_{cd}},

  ACL(mr_{op}) = {u_{op}},

  ACL(mr_{ed}) = {u_{ed}},

  ACL(mr_{np}) = {u_{np}}
- Action ={read, write}
- policyLevel(read,mr_{pp})=∞, policyLevel(write,mr_{pp})=0,

  policyLevel(read,mr_{gs})=∞, policyLevel(write,mr_{gs})=0,

  policyLevel(read,mr_{cd})=∞, policyLevel(write,mr_{cd})=0,

  policyLevel(read,mr_{op})=∞, policyLevel(write,mr_{op})=0,

  policyLevel(read,mr_{ed})=∞, policyLevel(write,mr_{ed})=0,

  policyLevel(read,mr_{np})=∞, policyLevel(write,mr_{np})=0
- Authorization policy:

  $Authz_{read}(u,o) \equiv u \in P^{policyLevel(read,o)}$

  $Authz_{write}(u,o) \equiv u \in P^{policyLevel(write,o)}$

Some sample operations and their outcomes are given below.

1. read(u_{np}, mr_{pp}) : authorized
2. read(u_{cd}, mr_{np}) : authorized

3. write(u_{np}, mr_{np}) : authorized

4. write(u_{np}, mr_{pp}) : denied

5. write(u_{np}, mr_{pp}) : denied

5.2 Implementation of OOReBAC in Openstack Object Storage Swift

Using object-to-object relationship brings in a new dimension when we consider relationship between objects which can originate in different environments. Organizations often use multicloud environment for independent and parallel work, including reducing reliance on any single vendor, increasing flexibility through choice, and mitigating against disasters, etc. This is similar to the use of best-of-breed applications from multiple developers on a personal computer, rather than the defaults offered by the operating system vendor. Using multiple infrastructure providers for different workloads, deploying a single workload load balanced across multiple providers (active-active), or deploying a single workload on one provider, with a backup on another (active-passive) [6], are common multicloud applications. Sharing resources between multiple clouds IaaS is very important in today's multicloud world. Using object-to-object relationship can be one way to share our objects between different clouds.

For a proof-of-concept implementation we use homogeneous multicloud using the open source IaaS OpenStack platform [9] for all clouds in the system. In particular we build upon the OpenStack object storage Swift. In this section we provide a brief description of the implementation of OOReBAC. We first review OpenStack object storage Swift and its original authorization module.

Our proposed model for Swift authorization can be named as relationship based resource sharing for OpenStack object storage Swift. It enables the following features beyond those natively provided in Swift.

- Object specific ACL.

- Allow users to access objects through relationship along with ACL.

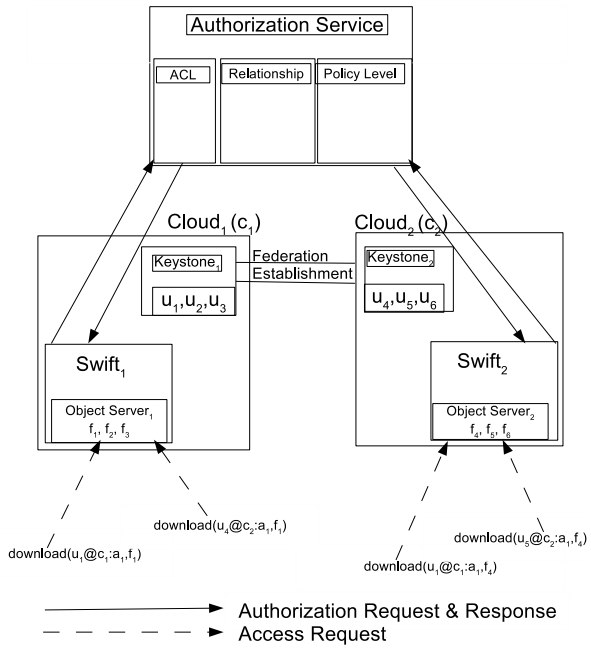


Figure 5.6: MultiCloud Implementation of OOReBAC Model.

Algorithm 5.1 authorize(u,f,G)

```
if u in ACL(f) then
    return true
else
    policyLevel = policyLevel(f)
    for depth limited search upto min(policyLevel, |O| - 1) do
        if if any of the file's ACL contains u then
            return true
    return false
```

- Allow users outside projects/accounts to access an object through relationship.
- Overall this proposed model would be able to work in multicloud environment.

To enable these features we are proposing an authorization service for Swift access control.

5.2.1 Proposed Authorization Service for Swift

An authorization service for Swift would take care of the authorization of objects. We would store all the container level ACL and relationship between files in authorization service. The collaboration between different clouds is done through federation. Once federation is established every file can be accessed by two types of user, local user and federated user. Swift operations

Table 5.2: Functional Specification.

Functions	Conditions	Updates
Administrative Actions		
CreateRelationship (u,filename$_1$,filename$_2$)	admin $\in$ role(u) $\wedge$ cloud(filename$_1$) = cloud(u) $\wedge$ filename$_1$ $\notin$ RelationshipSet(filename$_2$) $\wedge$ filename$_2$ $\notin$ RelationshipSet(filename$_1$)	RelationshipSet(filename$_1$) $\cup$= {filename$_2$} RelationshipSet(filename$_2$) $\cup$= {filename$_1$}
DeleteRelationship (u,filename$_1$,filename$_2$)	admin $\in$ role(u) $\wedge$ cloud(filename$_1$) = cloud(u) filename$_1$ $\in$ RelationshipSet(filename$_2$) $\wedge$ filename$_2$ $\in$ RelationshipSet(filename$_1$)	RelationshipSet(filename$_1$) $\setminus$= {filename$_2$} RelationshipSet(filename$_2$) $\setminus$= {filename$_1$}
IncludeAUserinACL (u,filename$_1$,username$_1$)	Role(u) $\in$ Admin$\wedge$ cloud(filename$_1$) = cloud(u)$\wedge$ username$_1$ $\notin$ ACLSet(filename$_1$)	ACLSet(filename$_1$) $\cup$= {username$_1$}
ExcludeAUserFromACL (u,filename$_1$,username$_1$)	Role(u) $\in$ Admin$\wedge$ cloud(filename$_1$) = cloud(u)$\wedge$ username$_1$ $\in$ ACLSet(filename$_2$)	ACLSet(filename$_1$) $\setminus$= {username$_1$}
ConfigurePolicyLevel (u,filename,num)	Role(u)$\in$ Admin $\wedge$ cloud(filename$_1$) = cloud(u) num $\leq$ | O |	PolicyLevel(filename)= num
Operational Command		
download (u,filename$_1$)	u$\in$ U $\wedge$ authorize(u,filename$_1$,G)	allow user u to download file filename$_1$

are of two types: **Administrative Operations** and **User Operations**. Creating ACL entry for a particular object, updating ACL, creating relationship between objects, updating relationship, configuring policy levels and updating policy levels are **Administrative Operations**.

The proposed OOReBAC theoretical model is defined for operational authorization and does not include an administrative model. Therefore, for our implementation we have defined a simple administrative model for Swift authorization service. This administrative model allows an admin user from any of the collaborating clouds to configure and update relationships, ACLs and policy levels. To configure and update relationship admin user and at least one file for which relationship is being configured should be from same cloud. To configure and update ACL and policyLevel admin user and the corresponding file should be from same cloud. Admin user can directly issue a RESTAPI command from Swift to the authorization service database to create relationships, update relationships, create an ACL, update an ACL, create policy level and update policy level. In Swift **User Operations** are uploading a file and downloading a file. Only the creator of the container can upload a file. In our implementation the upload operation is kept as it is. The authorization of downloading a file is done through authorization service.

Figure 5.6 shows the implementation detail of the model. In this figure we are considering two

Table 5.3: Relationship.

SourceFileName	TargetFileList
f_1@$cloud_1$:$account_1$:$container_1$	{f_2@$cloud_1$:$account_1$:$container_1$, f_3@$cloud_2$:$account_1$:$container_1$}
...	...

Table 5.4: ACL.

Filename	UserList
f_1@$cloud_1$:$account_1$:$container_1$	{u_1@$cloud_1$:$account_1$, u_2@$cloud_1$:$account_1$}
...	...

clouds c_1 and c_2. First we need to establish federation between these two clouds. The authorization service would contain all the ACL information of every file, along with relationship information and policy level information. To configure our OOReBAC model for this implementation platform user identification will comprise of cloud and current account information along with user name. Files or objects also need to contain filename along with cloud name, account name and container name. Each user is identified as username@cloudname:accountname, while each file is identified as filename@cloudname:accountname:containername.

When a download request comes from a user for a local file, the user's request triggers a RESTAPI call to the authorization service. The authorization service looks up the ACL table to determine if this user has direct access to the file. If so it returns true, else it goes to the policyLevel table to find out how many levels of relationship the file allows. Then it looks up to the policy level depth in relationship table whether any of the file up to that depth has an ACL authorizing the accessing user. If it finds any it returns true, otherwise it returns false.

Algorithm 5.1 shows the pseudocode of the algorithm in the authorization service to evaluate access authorization. Here we have used depth limited search upto a fix depth considering the policy level of a particular object for a particular action. Depth limited search searches upto a fix limited depth for all possible paths. Depth first search is a special case of depth limited search where limit is ∞. The overall time complexity of the algorithm is $O(\mid O \mid^{\mid O \mid})$, although with small policy limits the performance will be considerably better.

Table 5.2 specifies the administrative commands and operational commands of the imple-

Table 5.5: Policy Level

FileName	Policy Level
f_3@$cloud_1$:$account_2$:$container_3$	(download,2)
...	...

mented model for the authorization service. Administrative function **CreateRelationship** creates relationship between two files by a cloud admin. It takes a user and two filename as input. It checks whether the cloud admin and the first file are from same cloud and that no relationship exists between the two files. Administrative function **DeleteRelationship** deletes an existing relationship between two files, **IncludeAUserinACL** includes a user in the ACL list of a file by the cloud admin. **ExludeAUserinACL** excludes a user in the ACL list of a file by the cloud admin, and **ConfigurePolicyLevel** configures the policy level of a file by the cloud admin.The only user operation is **download**. It takes a user and a file as input, and checks whether the user is an existing user and using the **authorize** algorithm from authorization service it returns true or false.

Tables 5.3, 5.4, and 5.5 shows the structure of the Relationship, ACL and policyLevel table in the authorization service. In Relationship table the graph is stored in adjacency list format. In ACL table ACL information is stored as file specific userlist and in Policy Level table file specific policylevel is stored.

Chapter 6: CONCLUSION

The following sections summarize contributions of this dissertation and discuss some future research directions that can be further investigated.

6.1 Summary of Contributions

This dissertation makes fundamental contributions towards consensus study on characteristics and comparison of Attribute and Relationship Based Access control models.

First it develops a rigorous comparison between ABAC and ReBAC. We do this by classifying ABAC and ReBAC models based on salient aspects that are relevant to their comparison and proposes two semi-formal families of models for both ReBAC and ABAC and compare them. The main purpose of these classifications is to enable comparison. The classifications are not intended to be a complete characterization of ABAC models or ReBAC models. They are only a partial classification but sufficient to draw out the essential relationships between ABAC and ReBAC.

Second it analyzes the safety of the existing ABAC_α model and proposes two enhanced versions of ABAC_α so as to analyze the decidability boundary and comparative expressive power for the extensions. One keeps the same safety decidability result while other extension has undecidable safety. The first extension is shown to be equivalent in expressive power to the existing safety decidable $\mathrm{UCON}^{\mathrm{finite}}_{\mathrm{preA}}$ model.

Third it defines a novel form of ReBAC model (OOReBAC) considering object-to-object relationship independent of users to control access of resources and an implementation of it for multicloud resource sharing in OpenStack object storage Swift.

6.2 Future Work

There are several opportunities to extend the comparative study and analysis of Attribute and Relationship Based Access Control Models presented in this dissertation.

The relationship between ABAC and ReBAC can be define more concretely through config-

uration of more sophisticated ReBAC we have these days. More significantly metrics beyond theoretical equivalence need to be brought into consideration to better understand the relative advantages and disadvantages of these two approaches. Performance is one such metric but others such as maintainability, robustness, and agility, also need to be studied. The families of model we have presented are semi formal. To study formal comparison we need to consider formal representation of sophisticated ReBAC models with multiple asymmetric relationships between different entities (such as users, resources etc.), policy individualization, incoming and outgoing actions and then application of this framework in the general computing system.

The OOReBAC model presented here is motivated by object-to-object relationship in object-oriented systems and version control systems while it is more influenced by ReBAC in social context. It only considers one type of symmetric relationship whereas object-oriented systems contain different types of asymmetric relationship (inheritance, composition, and association). Version control system considers one type of relationship “derived from" however the graph is a directed acyclic graph (DAG) and the relationship is asymmetric. It would be interesting future work to develop a model evolved from OOReBAC, which can instantiate access control for an already existing object relationship application such as object-oriented systems and version control systems.

BIBLIOGRAPHY

[1] Alloy Language and Tool. http://alloy.mit.edu/alloy/. [Online; Accessed: 10/5/2017].

[2] Digital Asset Management. http://www.adobepress.com/articles/article.asp?p=2129363. [Online; Accessed: 10/05/2017].

[3] Digital Library. https://en.wikiversity.org/wiki/Digital_Libraries/Recommender_systems#Recommender_Systems. [Online; Accessed: 10/05/2017].

[4] Facebook Help Community Question. https://www.facebook.com/help/community/question/?id=10151800679568529. [Online; Accessed: 10/10/2017].

[5] Function Composition. https://en.wikipedia.org/wiki/Function_composition. [Online; Accessed: 10/05/2017].

[6] Multi-Cloud. https://en.wikipedia.org/wiki/Multicloud. [Online; Accessed: 10/5/2017].

[7] Neo4j. http://www.neo4j.org/. [Online; Accessed: 10/05/2017].

[8] Nested Function. http://mathworld.wolfram.com/NestedFunction.html. [Online; Accessed: 10/05/2017].

[9] Openstack. http://www.openstack.org/. [Online, Accessed: 10/05/2017].

[10] Panama Paper Leaks. http://info.neo4j.com/05262016---ICIJ-and-Panama-Papers-OnDemand_LP-Video.html?aliId=38013278. [Online; Accessed: 10/05/2017].

[11] Singlevalue multivalue. https://msdn.microsoft.com/en-us/library/aa746488(v=vs.85).aspx. [Online; Accessed: 10/4/2017].

[12] Structured Attribute. https://docops.ca.com/ca-identity-manager/12-6-5/EN/configuring/user-console-design/configuring-profile-tabs-and-screens/field-styles/structured-attribute-display. [Online; Accessed: 10/4/2017].

[13] Swift. http://docs.openstack.org/developer/swift/. [Online; Accessed: 10/05/2017].

[14] Swift API. http://docs.openstack.org/developer/swift/api/object_api_v1_overview.html. [Online; Accessed: 10/05/2017].

[15] Swift Authorization. http://docs.openstack.org/developer/swift/overview_auth.html. [Online; Accessed: 10/05/2017].

[16] Version Control. http://web.mit.edu/6.005/www/sp16/classes/05-version-control/. [Online; Accessed: 10/5/2017].

[17] number-of-monthly-active-facebook-users-worldwide. https://zephoria.com/top-15-valuable-facebook-statistics/, 2017. [Online; Accessed: 10/05/2017].

[18] Ali E. Abdallah and Etienne J. Khayat. A formal model for parameterized role-based access control. In *Formal Aspects in Security and Trust*, 2004.

[19] Tahmina Ahmed, Farhan Patwa, and Ravi Sandhu. Object-to-object relationship based access control: model and multi-cloud demonstration. In *IEEE Conference on Information Reuse and Integration (IRI)*, pages 297–304. IEEE, 2016.

[20] Tahmina Ahmed and Ravi Sandhu. Safety of ABAC_{α} is decidable. In *International Conference on Network and System Security (NSS)*. Springer, 2017.

[21] Tahmina Ahmed, Ravi Sandhu, and Jaehong Park. Classifying and comparing attribute-based and relationship-based access control. In *ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 59–70. ACM, 2017.

[22] Evangelos Aktoudianakis, Jason Crampton, Scott Schneider, Helen Treharne, and Adrian Waller. Policy templates for relationship-based access control. In *Eleventh Annual International Conference on Privacy, Security and Trust (PST)*, pages 221–228. IEEE, 2013.

[23] Mohammad A. Al-Kahtani and Ravi S. Sandhu. A model for attribute-based user-role assignment. In *Annual Computer Security Applications Conference*, 2002.

[24] Asma Alshehri and Ravi Sandhu. On the relationship between finite domain ABAM and PreUCON$_A$. In *International Conference on Network and System Security (NSS)*, pages 333–346. Springer, 2016.

[25] Paul Ammann, Ravi S Sandhu, and Richard Lipton. The expressive power of multi-parent creation in monotonic access control models. *Journal of Computer Security*, 4(2-3):149–165, 1996.

[26] Eric Andonoff, Gilles Hubert, Annig Le Parc, and Gilles Zurfluh. Modelling inheritance, composition and relationship links between objects, object versions and class versions. In *Advanced Information Systems Engineering*, pages 96–111. Springer, 1995.

[27] Malcolm P Atkinson, Francois Bancilhon, David J DeWitt, Klaus R Dittrich, David Maier, and Stanley B Zdonik. The object-oriented database system manifesto. In *DOOD*, volume 89, pages 40–57, 1989.

[28] John Barkley, Konstantin Beznosov, and Jinny Uppal. Supporting relationships in access control using role based access control. In *Proceedings of the Fourth ACM Workshop on Role-based Access Control*, pages 55–65. ACM, 1999.

[29] Michael Barr and Charles Wells. Category theory for computing science. In *Prentice Hall*, page 6, 1998.

[30] Fabrício Benevenuto, Tiago Rodrigues, Meeyoung Cha, and Virgílio Almeida. Characterizing user behavior in online social networks. In *ACM SIGCOMM Internet Measurement Conference (IMC)*, pages 49–62. ACM, 2009.

[31] Phillipa Bennett, Indrakshi Ray, and Robert France. Analysis of a relationship based access control model. In *Proceedings of the Eighth International C* Conference on Computer Science & Software Engineering*, pages 1–8. ACM, 2015.

[32] Elisa Bertino, Barbara Catania, Elena Ferrari, and Paolo Perlasca. A logical framework for reasoning about access control models. Proceedings of 6th ACM Symposium on Access Control Models and Technologies (SACMAT), 2001.

[33] Elisa Bertino, Pierangela Samarati, and Sushil Jajodia. An extended authorization model for relational databases. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 9(1):85–101, 1997.

[34] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy (S & P)*, pages 321–334. IEEE, 2007.

[35] Prosunjit Biswas, Farhan Patwa, and Ravi Sandhu. Content Level Access Control for OpenStack Swift Storage. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 123–126. ACM, 2015.

[36] Piero A. Bonatti and P. Samarati. Regulating service access and information release on the web. In *ACM Conference on Computer and Communications Security (CCS)*, 2000.

[37] Piero A. Bonatti and P. Samarati. A uniform framework for regulating service access and information release on the web. *Journal of Computer Security*, 2002.

[38] Joël Brunet. Modeling the world with semantic objects. In *IFIP - WG 8.1 Conf. on "The Object Oriented Approach in Information Systems"*, page 1, 1991.

[39] Glenn Bruns, Philip WL Fong, Ida Siahaan, and Michael Huth. Relationship-based access control: its expression and enforcement through hybrid logic. In *ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 117–124, 2012.

[40] Edward Caprin, Yan Zhang, and Khaled M. Khan. Social access control language (SocACL). In *Proceedings of the 6th International Conference on Security of Information and Networks*, pages 261–265. ACM, 2013.

[41] Barbara Carminati, Elena Ferrari, Raymond Heatherly, Murat Kantarcioglu, and Bhavani Thuraisingham. A semantic web based framework for social network access control. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 177–186. ACM, 2009.

[42] Barbara Carminati, Elena Ferrari, and Andrea Perego. Rule-based access control for social networks. In *OTM Confederated International Conferences: On the Move to Meaningful Internet Systems*, pages 1734–1744. Springer, 2006.

[43] Barbara Carminati, Elena Ferrari, and Andrea Perego. Enforcing access control in web-based social networks. *ACM Transactions on Information and System Security (TISSEC)*, 13(1):6, 2009.

[44] Barbara Carminati, Elena Ferrari, and Andrea Perego. A decentralized security framework for web-based social networks. *Pervasive Information Security and Privacy Developments: Trends and Advancements: Trends and Advancements*, page 356, 2010.

[45] D. Chadwick. *Understanding X.500: The Directory*. Chapman & Hall, Ltd., 1994.

[46] David W. Chadwick, Alexander Otenko, and Edward Ball. Role-based access control with X.509 attribute certificates. *IEEE Internet Computing*, 2003.

[47] Ajay Chander, John C Mitchell, and Drew Dean. A state-transition model of trust management and access control. In *csfw*, volume 1, pages 27–43, 2001.

[48] Melissa Chase. Multi-authority attribute based encryption. In *Theory of Cryptography Conference*, pages 515–534. Springer, 2007.

[49] Peter Pin-Shan Chen. The entity-relationship model toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976.

[50] Yuan Cheng, Khalid Bijon, and Ravi Sandhu. Extended ReBAC administrative models with cascading revocation and provenance support. In *Proceedings of the 21st ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 161–170. ACM, 2016.

[51] Yuan Cheng, Jaehong Park, and Ravi Sandhu. Relationship-based access control for online social networks: Beyond user-to-user relationships. In *International Conference on Privacy, Security, Risk and Trust (PASSAT)*, pages 646–655. IEEE, 2012.

[52] Yuan Cheng, Jaehong Park, and Ravi Sandhu. A user-to-user relationship-based access control model for online social networks. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 8–24. Springer, 2012.

[53] Yuan Cheng, Jaehong Park, and Ravi Sandhu. Attribute-aware relationship-based access control for online social networks. In *IFIP Annual Conference on Data and Applications Security and Privacy(DBSEC)*, pages 292–306. Springer, 2014.

[54] Michael J. Covington, Wende Long, Srividhya Srinivasan, Anind K. Dey, Mustaque Ahamad, and Gregory D. Abowd. Securing context-aware applications using environment roles. In *Proceedings of 6th ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2001.

[55] Jason Crampton and James Sellwood. Caching and auditing in the RPPM model. In *International Workshop on Security and Trust Management (STM)*, pages 49–64. Springer, 2014.

[56] Jason Crampton and James Sellwood. Path conditions and principal matching: a new approach to access control. In *Proceedings of the 19th ACM Symposium on Access Control Models and Technologies(SACMAT)*, pages 187–198. ACM, 2014.

[57] Jason Crampton and James Sellwood. ARPPM: Administration in the RPPM model. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 219–230. ACM, 2016.

[58] Jason Crampton and James Sellwood. Inter-ReBAC: inter-operation of relationship-based access control model instances. In *IFIP Annual Conference on Data and Applications Security and Privacy (DBSEC)*, pages 96–105. Springer, 2016.

[59] E. Damiani, S.D.C. di Vimercati, and P. Samarati. New paradigms for access control in open environments. *International Symposium on Signal Processing and Information Technology (ISSPIT)*, 2005.

[60] Gillian Dobbie, Xiaoying Wu, Tok Wang Ling, and Mong Li Lee. Ora-ss: An object-relationship-attribute model for semi-structured data. *TR21/00, Department of Computer Science, National University of Singapore*, 2000.

[61] Mark Evered. Supporting parameterised roles with object-based access control. In *HICSS*, 2003.

[62] Ronald Fagin. On an authorization mechanism. *ACM Transactions on Database Systems (TODS)*, 3(3):310–319, 1978.

[63] David F Ferraiolo, Ravi Sandhu, Serban Gavrila, D Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, 2001.

[64] M. Fire, L. Tenenboim, O. Lesser, R. Puzis, L. Rokach, and Y. Elovici. Link prediction in social networks using computationally efficient topological features. In *IEEE Third International Confernece on Social Computing (SocialCom)*, pages 73–80. IEEE, 2011.

[65] M. Fire, R. Tenenboim, R. Puzis, O. Lesser, L. Rokach, and Y. Elovici. Computationally efficient link prediction in variety of social networks. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(1), 2013.

[66] Jeffrey Fischer, Daniel Marino, Rupak Majumdar, and Todd D. Millstein. Fine-grained access control with object-sensitive roles. In *European Conference on Object Oriented Programming (ECOOP)*, 2009.

[67] Philip WL Fong. Relationship-based access control: protection model and policy language. In *Proceedings of the first ACM conference on Data and Application Security and Privacy(CODASPY)*, pages 191–202. ACM, 2011.

[68] Philip WL Fong, Mohd Anwar, and Zhen Zhao. A privacy preservation model for facebook-style social network systems. In *European Symposium of Research in Computer Security (ESORICS)*, pages 303–320. Springer, 2009.

[69] Philip WL Fong and Ida Siahaan. Relationship-based access control policies and their policy languages. In *Proceedings of the 16th ACM Symposium on Access Control Models and Technologies(SACMAT)*, pages 51–60. ACM, 2011.

[70] L. Fuchs, G. Pernul, and R. Sandhu. Roles in information security: A survey and classification of the research area. *Computers and Security*, 2011.

[71] Jean Gallier. Discrete mathematics. In *PWS Publishing*, page 118. Springer, 2011.

[72] Carrie Gates. Access control requirements for web 2.0 security and privacy. *IEEE Web*, 2(0), 2007.

[73] Mei Ge and Sylvia L. Osborn. A design for parameterized roles. In *Data and Application Security and Privacy (DBSEC)*, 2004.

[74] Luigi Giuri and Pietro Iglio. Role templates for content-based access control. In *ACM Workshop on RBAC*, 1997.

[75] Patricia P Griffiths and Bradford W Wade. An authorization mechanism for a relational database system. *ACM Transactions on Database Systems (TODS)*, 1(3):242–255, 1976.

[76] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Communications of the ACM(CACM)*, 19(8):461–471, 1976.

[77] Vincent C. Hu, David Ferrariolo, Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Scarfone Karen. Guide to attribute based access control (ABAC) definitions and considerations. *NIST Special Publication*, 800(162), 2014.

[78] John Hughes and Eve Maler. Security Assertion Markup Language (SAML), v2.0 technical overview. *OASIS SSTC Working Draft sstc-saml-tech-overview-2.0-draft-08*, pages 29–38, 2005.

[79] Mohammad Jafari and Mohammad Fathian. Management advantages of object classification in role-based access control (RBAC). In *Asian computing science conference on Advances in computer science: computer and network security(ASIAN)*, 2007.

[80] Sushil Jajodia, P. Samarati, Maria Luisa Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems (TODS)*, 2001.

[81] Xin Jin. *Attribute-Based Access Control Models And Implementation In Cloud Infrastructure As A Service*. PhD dissertation, University of Texas at San Antonio, 2014.

[82] Xin Jin, Ram Krishnan, and Ravi Sandhu. A unified attribute-based access control model covering DAC, MAC and RBAC. In *IFIP Annual Conference on Data and Applications Security and Privacy (DBSEC)*, pages 41–55. Springer, 2012.

[83] Anas Abou El Kalam, Salem Benferhat, Alexandre Miège, Rania El Baida, Frédéric Cuppens, Claire Saurel, Philippe Balbiani, Yves Deswarte, and Gilles Trouessin. Organization based access control. In *POLICY*, 2003.

[84] S. Kandala, R. Sandhu, and V. Bhamidipati. An attribute based framework for risk-adaptive access control models. In *International Conference on Availability, Reliability and Security (ARES)*, 2011.

[85] Won Kim, Elisa Bertino, and Jorge F. Garza. Composite objects revisited. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 337–347. ACM, 1989.

[86] Jan Kolter, Rolf Schillinger, and Günther Pernul. A privacy-enhanced attribute-based access control system. In *IFIP Annual Conference on Data and Applications Security and Privacy (DBSEC)*, pages 129–143. Springer, 2007.

[87] D. Richard Kuhn, Edward J. Coyne, and Timothy R. Weil. Adding attributes to role-based access control. *IEEE Computer*, 2010.

[88] Arun Kumar, Neeran M. Karnik, and Girish Chafle. Context sensitivity in role-based access control. *Operating Systems Review*, 2002.

[89] Bo Lang, Ian T. Foster, Frank Siebenlist, Rachana Ananthakrishnan, and Timothy Freeman. A flexible attribute based access control method for grid computing. *Journal of Grid Computing*, 2009.

[90] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, 2014.

[91] Ming Li, Shucheng Yu, Yao Zheng, Kui Ren, and Wenjing Lou. Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 24(1):131–143, 2013.

[92] Ninghui Li, John C Mitchell, and William H Winsborough. Design of a role-based trust-management framework. In *IEEE Symposium on Security and Privacy (S&P)*, pages 114–130. IEEE, 2002.

[93] Ninghui Li and William H Winsborough. Beyond proof-of-compliance: Safety and availability analysis in trust management. In *IEEE Symposium on Security and Privacy (S&P)*, pages 123–139. IEEE, 2003.

[94] Alex X. Liu, Fei Chen, JeeHyun Hwang, and Tao Xie. Xengine: a fast and scalable XACML policy evaluation engine. In *SIGMETRICS*, 2008.

[95] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, et al. The open provenance model core specification (v1. 1). *Future Generation Computer Systems*, 27(6):743–756, 2011.

[96] M.Sipser. *Introduction to the Theory of Computation*. PWS Publishing, 1997.

[97] Dang Nguyen, Jaehong Park, and Ravi Sandhu. A provenance-based access control model for dynamic separation of duties. In *Eleventh Annual International Conference on Privacy, Security and Trust (PST)*, pages 247–256. IEEE, 2013.

[98] OASIS. Extensible Access Control Markup Language (XACML), v2.0 (2005). 2005.

[99] Sylvia Osborn, Ravi Sandhu, and Qamar Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(2):85–106, 2000.

[100] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In *ACM Conference on Computer and Communications Security (CCS)*, pages 195–203. ACM, 2007.

[101] Jun Pang and Yang Zhang. A new access control scheme for facebook-style social networks. *Computers & Security*, 54:44–59, 2015.

[102] Jaehong Park, Dang Nguyen, and Ravi Sandhu. A provenance-based access control model. In *International Conference on Privacy, Security and Trust (PST)*, pages 137–144. IEEE, 2012.

[103] Jaehong Park and Ravi Sandhu. The UCONabc usage control model. *ACM Transactions on Information System Security (TISSEC)*, 2004.

[104] Torsten Priebe, Wolfgang Dobmeier, and Nora Kamprath. Supporting attribute-based access control with ontologies. In *International Conference on Availability, Reliability and Security (ARES)*, 2006.

[105] Navid Pustchi and Ravi Sandhu. MT-ABAC: A multi-tenant attribute-based access control model with tenant trust. In *International Conference on Network and System Security (NSS)*, pages 206–220. Springer, 2015.

[106] Bo Qin, Hua Deng, Qianhong Wu, Josep Domingo-Ferrer, David Naccache, and Yunya Zhou. Flexible attribute-based encryption applicable to secure e-healthcare records. *International Journal of Information Security*, 14(6):499–511, 2015.

[107] PV Rajkumar and Ravi Sandhu. Safety decidability for pre-authorization usage control with finite attribute domains. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 13(5):582–590, 2016.

[108] Prathima Rao, Dan Lin, Elisa Bertino, Ninghui Li, and Jorge Lobo. An algebra for fine-grained integration of XACML policies. In *Proceedings of 14th ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2009.

[109] Syed Zain R Rizvi and Philip WL Fong. Interoperability of relationship-and role-based access control. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 231–242. ACM, 2016.

[110] Syed Zain R Rizvi, Philip WL Fong, Jason Crampton, and James Sellwood. Relationship-based access control for an open-source medical records system. In *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 113–124. ACM, 2015.

[111] Marko A Rodriguez and Peter Neubauer. Constructions from dots and lines. *Bulletin of the American Society for Information Science and Technology*, 36(6):35–41, 2010.

[112] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques(EUROCRYPT)*, pages 457–473. Springer, 2005.

[113] Ravi Sandhu. Engineering authority and trust in cyberspace: The OM-AM and RBAC way. In *ACM Workshop on Role-Based Access Control*, pages 111–119. ACM, 2000.

[114] Ravi Sandhu. The PEI framework for application-centric security. In *International Workshop on Security and Communication Networks (IWSCN)*, pages 1–6. IEEE, 2009.

[115] Ravi Sandhu, Venkata Bhamidipati, and Qamar Munawer. The ARBAC97 model for role-based administration of roles. *Transactions on Information and System Security (TISSEC)*, 1999.

[116] Ravi Sandhu and Fang Chen. The multilevel relational (MLR) data model. *ACM Transactions on Information System Security (TISSEC)*, 1(1):93–132, 1998.

[117] Ravi Sandhu and Qamar Munawer. How to do discretionary access control using roles. In *Proceedings of the third ACM workshop on Role-based access control*, pages 47–54. ACM, 1998.

[118] Ravi Sandhu, Kumar Ranganathan, and Xinwen Zhang. Secure information sharing enabled by trusted computing and PEI models. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 2–12. ACM, 2006.

[119] Ravi S. Sandhu. Expressive power of the schematic protection model. *Journal of Computer Security*, 1(1):59–98, 1992.

[120] Ravi S. Sandhu. Lattice-based access control models. *IEEE Computer*, 26(11):9–19, 1993.

[121] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 1996.

[122] Ravi S Sandhu and Srinivas Ganta. On testing for absence of rights in access control models. In *Computer Security Foundations Workshop VI, 1993. Proceedings*, pages 109–118. IEEE, 1993.

[123] Ravi S. Sandhu and P. Samarati. Access control: Principles and practice. *Comunication Magazine, IEEE*, 1994.

[124] Christian Schläger, Manuel Sojer, Björn Muschall, and Günther Pernul. Attribute-based authentication and authorisation infrastructures for e-commerce providers. In *EC-Web*, 2006.

[125] Haibo Shen. A semantic-aware attribute-based access control model for web services. In *International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP)*, pages 693–703. Springer, 2009.

[126] Henry Small. Co-citation in the scientific literature: A new measure of the relationship between two documents. *Journal of the American Society for information Science*, 24(4):265–269, 1973.

[127] Scott D Stoller. An administrative model for relationship-based access control. In *IFIP Annual Conference on Data and Applications Security and Privacy (DBSEC)*, pages 53–68. Springer, 2015.

[128] Zhong Su, Qiang Yang, Hongjiang Zhang, Xiaowei Xu, and Yuhen Hu. Correlation-based document clustering using web logs. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS)*, pages 7–pp. IEEE, 2001.

[129] Bo Tang, Qi Li, and Ravi Sandhu. A multi-tenant RBAC model for collaborative cloud services. In *International Conference on Privacy, Security and Trust (PST)*, pages 229–238. IEEE, 2013.

[130] Bo Tang and Ravi Sandhu. Extending Openstack access control with domain trust. In *International Conference on Network and System Security (NSS)*, pages 54–69. Springer, 2014.

[131] Mahesh V Tripunitara and Ninghui Li. A theory for comparing the expressive power of access control models. *Journal of Computer Security*, 15(2):231–272, 2007.

[132] Mahesh V Tripunitara and Ninghui Li. The foundational work of Harrison-Ruzzo-Ullman revisited. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 10(1):28–39, 2013.

[133] Lingyu Wang, Duminda Wijesekera, and Sushil Jajodia. A logic-based framework for attribute based access control. In *In 2nd ACM Workshop on FMSE*, 2004.

[134] Christo Wilson, Bryce Boe, Alessandra Sala, Krishna PN Puttaswamy, and Ben Y Zhao. User interactions in social networks and their implications. In *Proceedings of the 4th ACM European Conference on Computer systems (EuroSys)*, pages 205–218. ACM, 2009.

[135] Jianming Yong, Elisa Bertino, Mark Toleman, and Dave Roberts. Extended RBAC with role attributes. In *10th Pacific Asia Conference on Information System (PACIS)*, 2006.

[136] Ting Yu, Xiaosong Ma, and Marianne Winslett. Prunes: an efficient and complete strategy for automated trust negotiation over the internet. In *ACM Conference on Computer and Communications Security (CCS)*, 2000.

[137] Ting Yu, Marianne Winslett, and Kent E. Seamons. Interoperable strategies in automated trust negotiation. In *ACM Conference on Computer and Communications Security (CCS)*, 2001.

[138] Ting Yu, Marianne Winslett, and Kent E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information System Security (TISSEC)*, 2003.

[139] Eric Yuan and Jin Tong. Attributed based access control (ABAC) for web services. In *Proceedings of the IEEE International Conference on Web Services(ICWS)*, pages 561–569. IEEE Computer Society, 2005.

[140] Xinwen Zhang, Ravi Sandhu, and Francesco Parisi-Presicce. Safety analysis of usage control authorization models. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, volume 6, pages 243–254, 2006.

VITA

Tahmina Ahmed was born and grew up in Dhaka, Bangladesh. Following her graduation from Shaheed Bir Uttam Lt. Anwar Girls' College and Viqarunnisa Noon School and College, Tahmina received her Bachelor of Science in Engineering degree with a major in Computer Science and Engineering from Bangladesh University of Engineering and Technology (BUET), Bangladesh in 2004. After completion of her BSc. Eng. she worked in software and telecommunication industry for 6 years. In 2011, she joined University of Texas at San Antonio (UTSA) to pursue her doctoral degree. She joined the Institute for Cyber Security at UTSA and started working with Dr. Ravi Sandhu since 2012. Her research interests include security and privacy in cyber space. In particular, her focus is on Attribute and Relationship Based Access Control and application of them in the cloud platform.