ABAM: An Attribute-Based Access Matrix Model

Xinwen Zhang
Department of Information and Software Engineering
George Mason University
xzhang6@gmu.edu

Yingjiu Li
School of Information Systems
Singapore Management University
yjli@smu.edu.sg

Divya Nalla
School of Information Systems
Singapore Management University
divyanalla@smu.edu.sg

August 9, 2005

Abstract

In traditional access control models like mandatory access control (MAC), discretionary access control (DAC), and role-based access control (RBAC), authorization decisions are determined according to the identities of subjects and objects, which are authenticated by a system completely. Recent access control practices, such as digital rights management (DRM), trust management, and usage control, require flexible authorization policies. In such systems, a subject may be only partially authenticated according to one or more attributes. Authorization policies are specified with subject and object attribute values. In this paper we propose an attribute-based access matrix model, named ABAM, which extends the original access matrix model. We show that ABAM enhances the expressive power of the access matrix model by supporting attribute-based authorizations and dynamic permission propagations. Specifically, ABAM is comprehensive enough to encompass traditional access control models as well as some usage control features. As expressive power and safety are two fundamental but conflictive objectives of an access control model, we study the safety property of ABAM and conclude that the safety problem is decidable for a restricted case where attribute relationship graph allows no cycles containing creating-attribute tuples. The restricted case is shown to sustain good expressive power to model practical systems.

1 Introduction

Protection systems aim at protecting various resources from damage or unauthorized access, and allowing multiple users to share the same resources. A model should be defined in such a way that it is capable of expressing practical systems, most of which are dynamic. In a dynamic system, a state change can be caused by some external actions, or operations from subjects inside the system (including system administrators). In an access control system, a state is specified by a set of subjects, objects, and access control configurations which determine authorization relations between the subjects and objects. Traditionally, an state change can be happened by an administrative operation. Modern access control systems is more dynamic that an access operation of a subject can change the system state.

Instead of totally authenticated subject identities in traditional access control models, recent practices like DRM, trust management, and usage control, consider partial authenticated subjects, which are specified by one or more attributes, such as role names, credit balances, locations, ages, etc. Partial authentication means that a system only requires the authentication of one or more attributes of a subject. For example, a

bookstore offering a discount to students of a state university only requires a student' membership while not the complete identity of the student. An object can also be specified by attributes such as type, directory, etc. A system state is specified not only by the access rights that subjects have on objects, but also their attribute values. An authorization decision is determined by predicates over subject and/or object attributes. For example, a subject can download a music file only when his/her credit is more than a constant value. In a dynamic system, as the result of an authorization, subject and/or object attribute values can be updated, and a system state changes to a new state, which may imply new permission propagations.

Access matrix is a simple but useful model first formalized by Harrison, Russo, and Ullman (HRU) [8]. In HRU, an object is represented by a column and a subject is represented by a row and a column. A right in a cell specifies that the subject of the row has the right on the object of the column. An access matrix model is specified by a set of subjects, a set of objects, an initial access matrix, and a fixed set of commands. A command takes a set of object (include subject) parameters as input and updates the matrix by adding or removing rights in cells, or creating or destroying objects, based on checking the presence of rights in a number of cells. A system state changes through execution of commands. Traditional access matrix is built on identities of subjects and objects.

In this paper we propose ABAM, an attribute-based access matrix model, by combining the simple access matrix model, as well as attributes-based authorizations which are motivated by recently presented access control systems. The main features of ABAM that differ from HRU include the specification of system states, the structure of access matrix, primitive operations, and command definitions. We show that ABAM has more expressive power than HRU, and can simulate some other extended models from HRU.

In any protection system, it is desirable to show that a particular situation is safe. Safety problem is the one to determine whether or not a given subject can eventually obtain an access privilege to a given object, in any state reachable from the initial state of the system. If there is an algorithm that is able to correctly decide this, then the safety problem is said to be decidable, where the algorithm could be in NP-complete or polynomial class. The strong feature of HRU is its expressive power. It could model many protection systems with various policies, while the safety of HRU is undecidable in general. The original work by Harrison, Ruzzo, and Ullman showed some situations for which the safety of HRU is decidable. But the expressive power of these situations was much less than that of the general HRU. In this paper we show that with some reasonable restrictions, ABAM is safety decidable, while maintaining expressive power for practical systems.

The rest of the paper is organized as follows. Section 2 presents some background and related work. Section 3 gives a formalized specification of ABAM. The expressive power of ABAM is illustrated in Section 4 by simulating many other access control models and expressing policies for various applications. In Section 5 we study the safety problem of ABAM with some restrictions on the model. Some related issues are discussed in Section 6. Section 7 summarizes this paper.

2 Background and Related Work

In this section we present some background knowledge on the research of expressive power and safety analysis in protection systems, as well as some recently proposed attribute-based authorization approaches. The difference between these related work and our approach is discussed.

2.1 Expressive Power and Safety Analysis

The first attempts at modeling protection systems [6, 7, 4, 14] were abstract formulations of the reference monitors and protected objects of particular protection systems. Other informal models have also been

proposed [11, 3, 18] to express a variety of protection policies.

HRU is the first formalized model of access control mechanisms in computing systems. The HRU defines a configuration of a protection system to consist of a set of current subjects and objects, and an access matrix. The matrix has a row for each subject and a column for each object. Subjects are also considered to be objects and hence have a row and a column in an access matrix. The cell [a, b] in an access matrix contains the rights that authorize subject 'a' to perform operations on object 'b'. The present rights in cells can enable adding or removing rights in a cell, or creating or destroying objects, which results in a change of the protection state. A protection system is changed by means of commands.

It has been proved that the safety of HRU is undecidable in general, and decidable in a restricted case of a mono-operational (commands with a single operation) protection system [8]. But this decidable model is too restricted to model protection systems in general. It has been proved that safety is decidable for mono-conditional (commands with a single condition) monotonic protection systems [9] (monotonic systems are those in which there are no delete or destroy operations). It has also been proved that safety of mono-conditional systems with create, enter, and delete (but without destroy) commands, is decidable.

A number of protection models were developed to provide stronger safety. Lipton and Snyder [16] introduced the take-grant model with linear time algorithms for safety. It was deliberately designed to be of limited expressive power, so that it would not exhibit the undecidable safety of HRU. Sandhu's schematic protection model (SPM) [19] was developed to fill the gap in expressive power between take-grant and HRU, while sustaining efficient safety analysis. SPM was later extended to Extended SPM (ESPM) [1] to allow multiple parents for a child. ESPM is formally equivalent to monotonic HRU in expressive power, and retains the safety properties of SPM [2].

Sandhu also proposed a typed access matrix (TAM) model [20] which has stronger safety properties and broad expressive power than HRU. The concept of strong typing is introduced in TAM to achieve stronger security. To study the safety properties of TAM, the monotonic case of TAM was considered. Monotonic TAM (MTAM) consists of only the monotonic primitive operations (the remove and destroy operations are not present). MTAM is an acyclic MTAM when it is assumed that an object cannot create another object of the same type, and that there are only a finite number of generations (since there are finite number of types in the model). The safety of MTAM is undecidable in general, but is decidable (NP-hard) in the acyclic case. Specifically, a simplified version of acyclic MTAM (acyclic case of ternary MTAM) is defined, for which, the safety has polynomial complexity. The TAM has a variety of decidable safety cases, but most of which are limited to monotonic systems.

Safety problem has been proven to be decidable mostly for strongly constrained systems (like monotonic systems). Recently, the dynamic typed access matrix model (DTAM) was proposed [23] which allows the type of an object to change dynamically. It has been proved that the safety problem is decidable for non-monotonic protection systems in DTAM. To show this, a type-relationship (TR) graph is introduced. The safety problem for non-monotonic systems becomes decidable if the TR-graph of the systems has no cycle with respect to parent-child relationship between objects. We follow the same approach to study the safety problem in ABAM.

2.2 Attribute-based Authorization

Many attribute-based authorization systems have been proposed recently. Al-Kahtani and Sandhu [12] proposed an attribute-based user-role assignment model for RBAC. Traditionally a user is manually assigned to appropriate roles by a security administrator. RB-RBAC (Rule-Based RBAC) provides the mechanism to dynamically assign users to roles based on a finite set of authorization rules which are defined on user attributes. User attributes are provided along with the authentication information or can be fetched from databases. These rules take into consideration the attributes users own and any constraints set forth by the

enterprise security policies. The model also allows dynamic revocation of assigned roles based on conditions specified in attributes.

Li et al. [15] present a role-based trust management (RT) as attribute-based authorization framework. The authorization decisions are based on the chains of digitally signed attribute credentials through which credential issuers assert their judgments about the attributes of entities, such as users and organizations. A central issue is that the data contained in credentials is often sensitive and must be protected. In other words, the credentials that must be presented to obtain access are themselves subject to access control. Wang et al. [24] propose an attribute-based authorization framework, which is extended from the flexible authorization framework (FAF) [10]. While these approaches focus on the individual policies of a system, we aim to an attribute-based access model by extending the simple and efficient access matrix. Access matrix model has been studied well in the literature and widely used in existing systems. This makes our approach very practical and easy to implement.

Park and Sandhu [17] propose the usage control model (UCON) consisting of a family of models built around the decision factors authorizations, obligations, and conditions, where authorizations are determined by a set of predicates based on subject and/or object attributes. In addition to these three decision factors, UCON has two important properties called decision continuity and attribute mutability. The continuity property describes the continuous enforcement of authorizations, obligations, and conditions by evaluating usage requirements throughout an access, while mutability captures the attribute value changes as side-effects of subject's actions.

The ABAM proposed in this paper captures the features of attribute-based authorizations and mutability specified in UCON. The difference between ABAM and UCON is that in ABAM, multiple subjects and objects along with their attributes are considered in a decision, while for UCON, a policy refers to a specific pair of (subject, object). More specifically, in the UCON model, a usage decision is made based on an individual pair of subject and object; whereas in our model, conditions are checked involving multiple subjects and objects to perform certain usage decisions.

3 A Formal Model of ABAM

ABAM is defined in terms of access control matrix and commands following the traditions of defining HRU, TAM and other access control models. Subjects, objects, attributes, access rights, access matrix, primitive operations, and commands are the basic components constituting the new model.

3.1 Subjects, Objects, and Access Rights

Definition 1 *All the entities to be protected in a system (passive entities) are called objects (O). The set of all active entities (for example processes and users) that can invoke some access requests or execute some permissions on an object are called subjects (S).*

A subject can be accessed by another subject. For example, a process can be initiated or killed by another process. Hence subjects can also be considered as objects. Following the general concept in traditional access control models, we consider the set of subjects in ABAM to be a subset of the set of objects. The objects that are not subjects are pure objects. Sometimes we refer to a subject or an object as an entity when the distinction is immaterial.

Each object in ABAM is specified by a unique identity recognized by the system. Note that an identity here is not an "identity" in traditional access control models which is totally authenticated by a system and used for authorizations, but a simple name or ID assigned by the system when an object is created, and cannot be changed after creation.

Definition 2 *Access rights (R) are the kinds of access that subjects can execute on objects (e.g., read, write, execute).*

3.2 Attributes and Attribute Tuples

Definition 3 *Each entity has a set of attributes. Each attribute is a variable of a specific data type, which is dependent on what the attribute is, and determines a domain from which a value can be assigned to the attribute in any state.*

An attribute value of an entity is denoted as $ent.a$ where ent is the entity name (i.e., the entity's identity) and a is the attribute name, and $ent.a \in dom(a)$, where $dom(a)$ is the value domain of a, and $null \notin dom(a)$. Hereafter, we assume that an entity name without any attribute specified denotes its identity. Without loss of generality we assume that in a system each entity has the same set of attributes, denoted as AT. An attribute with value of $null$ indicates that the entity does not have the attribute, or the attribute value has not be assigned after creation. For example, an administrator (s_1) in an organization has an administrative role (e.g., ar_1), then $s1.ad\_role = ar_1$. While a normal employee (s_2) does not have this attribute, then $s_2.ad\_role = null$.

Definition 4 *For an entity ent, an* attribute value tuple *(or simply* attribute tuple*) is a function $ATT_{ent} : AT \rightarrow dom(AT) \cup \{null\}$ that assigns a value to each attribute of the entity.*

The attribute tuple of an entity specifies its state in a system. Changing the attribute value(s) of an entity changes its attribute tuple. For example, consider an object o. In a particular state of the system, the object has the attribute tuple $ATT_o = (a_1 = v_1, a_2 = v_2, \ldots, a_n = v_n)$, where $n = |AT|$ and $v_i \in dom(a_i) \cup \{null\}$ for $1 \leq i \leq n$. Updating an attribute of a_i from value v_i to v_i' means that the attribute tuple changes from ATT_o to $ATT_o' = (a_1 = v_1, a_2 = v_2, \ldots, a_i = v_i', \ldots, a_n = v_n)$, where $v_i' \in dom(a_i)$.

3.3 Attribute Predicates

An attribute predicate is a polynomially computable boolean-valued function built from attributes and constants with appropriate operation and relation symbols. We can distinguish different types of predicates in ABAM, such as unary predicates and binary predicates. A unary predicate is built from one attribute variable and constants, e.g., $Alice.credit \geq \$100.00$, $file1.classification = `supersecure'$. A binary predicate is built from two different attribute variables, e.g., $dominate(Alice.clearance, file1.classification)$, $Alice.credit \geq ebook.value$, $(Alice, r) \in file1.acl$, where $file1.acl$ is object $file1$'s access control list. Note that the two attributes in a binary predicate can be from a single entity, or two different entities. In real systems, we can define more general predicates built on any number of subject attributes and/or object attributes. This significantly improves the flexibility and expressive power of ABAM.

3.4 Access Matrix

Definition 5 *An* access matrix *is a matrix with a row for each subject with its attribute tuple, and a column for each object with its attribute tuple.*

The set of all subjects along with their attribute tuples are used to represent the rows in an access matrix, and the set of all objects with their attribute tuples represent the columns. An element in cell $[s, o]$, representing the row s and column o, stores the set of all access rights that s can exercise over o.

Figure 1 shows an access matrix for subjects $S = \{s_1, s_2\}$ and objects $O = \{o_1, o_2\}$.

	s_1: $ATTs_1$	s_2: $ATTs_2$	o_1: $ATTo_1$	o_2: $ATTo_2$
s_1: $ATTs_1$		*{parent}*	*{read, write}*	
s_2: $ATTs_2$			*{read}*	

Figure 1: An access matrix

3.5 Protection State

Definition 6 *A* protection state *(or simply* state*) of an ABAM system is specified by*

- *a set of subjects S,*
- *a set of objects O,*
- *a set of subject attribute tuples $ATT_S = \{ATT_s\}$, where each ATT_s is the attribute tuple of subject $s \in S$,*
- *a set of object attribute tuples $ATT_O = \{ATT_o\}$, where each ATT_o is the attribute tuple of object $o \in O$, and*
- *an access matrix M with a row for every subject with its attribute tuple, and a column for every object with its attribute tuple.*

3.6 Primitive Operations

A primitive operation is an operation that can change the state of a system. The basic primitive operations can be defined as follows.

Definition 7 *The* primitive operations *(or simply* operations*) in ABAM are defined as in the Table 1, where $t = (S, O, ATT_S, ATT_O, M)$ and $t' = (S', O', ATT'_S, ATT'_O, M')$ are the states before and after a single primitive operation.*

The first six operations are similar to those in HRU, except that in each creation (subject or object), the created entity is assigned with null-valued attributes. Generally these attribute values can be changed with following update operation(s) in a command. For each creation, an unique identity is assigned to the new entity. For simplicity we assume the identity of a destroyed entity cannot be reused by the system. An update operation changes an attribute value to a new value, thus change the attribute tuple of the object. The new value can be a constant, or a polynomially computable function of the old value or other attribute values. In ABAM, a system state transform is not only reflected by the permission distributions in a matrix, but also attribute updates due to authorization decisions. An update, in turn, can cause other permission propagations in the new system state. This enhances the expressive power of ABAM beyond HRU as shown in the next section.

3.7 Commands

A command in ABAM consists of a condition and a sequence of primitive operations. The condition of a command in HRU checks the presence of some rights in the access matrix, whereas in ABAM, along with right presence, some predicates based on the attributes of subjects and objects can also be evaluated.

A command in ABAM is defined as follows.

Operations	Conditions	New States
enter r into $[s, o]$	$s \in S$ $o \in O$ $r \in R$	$S' = S, O' = O$ $M'[s, o] = M[s, o] \cup \{r\}$ $\forall s' \in S', o' \in O', M'[s', o'] = M[s', o']$ for $s' \neq s$ and $o' \neq o$ $\forall a \in AT, o' \in O', ATT'_{o'}(a) = ATT_{o'}(a)$
delete r from $[s, o]$	$s \in S$ $o \in O$ $r \in R$	$S' = S, O' = O$ $M'[s, o] = M[s, o] - \{r\}$ $\forall s' \in S', o' \in O', M'[s', o'] = M[s', o']$ for $s' \neq s$ and $o' \neq o$ $\forall a \in AT, o' \in O', ATT'_{o'}(a) = ATT_{o'}(a)$
create subject s'	$s' \notin S$	$S' = S \cup \{s'\}, O' = O \cup \{s'\}$ $\forall s \in S, o \in O, M'[s, o] = M[s, o]$ $\forall o \in O, M'[s', o] = \emptyset$ $\forall s \in S, M'[s, s'] = \emptyset$ $\forall a \in AT, o \in O, ATT'_o(a) = ATT_o(a)$ $\forall a \in AT, ATT'_{s'}(a) = null$
destroy subject s	$s \in S$	$S' = S - \{s'\}, O' = O - \{s'\}$ $\forall s' \in S', o' \in O', M'[s', o'] = M[s', o']$ $\forall a \in AT, o' \in O', ATT'_{o'}(a) = ATT_{o'}(a)$
create object o'	$o' \notin O$	$S' = S, O' = O \cup \{o'\}$ $\forall s \in S, o \in O, M'[s, o] = M[s, o]$ $\forall s \in S, M'[s, o'] = \emptyset$ $\forall a \in AT, o \in O, ATT'_o(a) = ATT_o(a)$ $\forall a \in AT, ATT'_{o'}(a) = null$
destroy object o	$o \in O$	$S' = S, O' = O - \{o'\}$ $\forall s' \in S', o' \in O', M'[s', o'] = M[s', o']$ $\forall a \in AT, o' \in O', ATT'_{o'}(a) = ATT_{o'}(a)$
update attribute $o.a_i = v'_i$	$o \in O$ $v'_i \in dom(a_i)$	$S' = S, O' = O$ $\forall s' \in S', o' \in O', M'[s', o'] = M[s', o']$ $\forall a \in AT$ and $a \neq a_i, \forall o' \in O', ATT'_{o'}(a) = ATT_{o'}(a)$ $ATT'_o(a_i) = v'_i$

Table 1: Primitive operations

Command $\alpha(X_1, X_2, \ldots, X_k)$
if $r_1 \in [X_{s1}, X_{o1}] \wedge r_2 \in [X_{s2}, X_{o2}] \wedge \ldots r_m \in [X_{sm}, X_{om}] \wedge$
$p_1 \wedge p_2 \wedge \ldots p_m$
then
op_1; op_2; ...; op_n
end

Here, α is the name of the command, $X_1, X_2, \ldots, X_k$ are object parameters; r_1, r_2, ..., r_m are generic rights; $s1$, $s2$, ..., sm and $o1$, $o2$, ..., om are integers between 1 and k; p_1, p_2, ..., p_n are predicates built over the attributes of the parameters. $op_1, op_2, \ldots, op_n$ are primitive operations; The if part of the command is called the condition of α. If a command has no condition, it is an unconditional command, otherwise, it is a conditional command. If a command has at least one "create subject" or "create object" operation, then this command is a creating command, otherwise it is non-creating. A command is invoked by substituting actual objects as parameters. The operations are executed sequentially if the condition of the command and the conditions of each primitive operations are true.

Without loss of generality, we assume that in a single command, there is at most one update operation for single attribute of an object, since multiple updates on the same attribute can be integrated into one. Also, we can assume that all non-update operations appear before all update operations in a command, since this sequence does not change the final state after the execution of a command.

Note that a disjunctive form of condition can also be easily modelled by having one command for each component. Also, negated predicates are not required explicitly, since we always can define a normal predicate for a negated one.

3.8 ABAM Model

After introducing the basic components, we complete this section with the definition of ABAM authorization scheme and system.

Definition 8 *An ABAM* authorization scheme *(or simply* scheme*) is a 4-tuple* (R, AT, P, C)*, where* R *a fixed set of generic rights,* AT *is a fixed set of attribute names,* P *is a fixed set of attribute predicates, and* C *is a fixed set of commands.*

Definition 9 *An ABAM* system *is specified by an ABAM authorization scheme and an initial state (an initial set of subjects and their attribute tuples, an initial st of objects and their attribute tuples, and an initial access matrix).*

In an ABAM system, commands can only be executed serially, and each execution is atomic. That is, either the condition of a command is satisfied and all primitive operations are performed successfully, or no change happens on the current state.

4 Expressive Power of ABAM

With the extensions on the conditions and primitive operations beyond HRU, ABAM can express not only traditional models, but also modern access control systems. In this section, first we use the new model to express HRU and TAM model, then we study the compliance of the new model to the basic usage control framework proposed by Park and Sandhu [17], and then we present a real life example to illustrate the flexibility of authorization with ABAM.

4.1 Modelling HRU

Consider an ABAM command as follow.

Command $\alpha(X_1, X_2, \ldots, X_k)$
if $r_1 \in [X_{s1}, X_{o1}] \wedge r_2 \in [X_{s2}, X_{o2}] \wedge \ldots r_m \in [X_{sm}, X_{om}] \wedge$
$p_1 \wedge p_2 \wedge \ldots p_m$
then
op_1; op_2; . . .; op_n
up_1; . . .; up_m;
end

where $up_1, \ldots, up_m$ are all update primitive operations, and $op_1, \ldots, op_m$ are all non-update primitive operations. If all predicates are trivially true in any state, then the update operations can be ignored since they do not affect the conditions of any commands. Therefore this can be considered as an HRU command as the follow.

Command $\alpha(X_1, X_2, \ldots, X_k)$
if $r_1 \in [X_{s1}, X_{o1}] \wedge r_2 \in [X_{s2}, X_{o2}] \wedge \ldots r_m \in [X_{sm}, X_{om}]$
then

op_1; op_2; ...; op_n
end

Thus, an HRU scheme can be constructed by ignoring the updates operations and predicates in the ABAM scheme. From the same initial state, the reachable states of a system with these two schemes are the same. Since this HRU scheme is general, then any HRU can be simulated with a restricted form of ABAM and HRU can be considered as a special case of ABAM. This implies that ABAM is at least as expressive as HRU. Actually, as it is shown shortly, ABAM is more expressive than HRU by simulating TAM and other models. This reduction also implies that the safety problem of a general ABAM is undecidable.

4.2 Modelling TAM

Consider an ABAM system with the following restrictions:

- each entity has only one attribute $type$;
- in each creating command, when an entity is created, its $type$ attribute is assigned with a constant with an update operation;
- there are no other update operations in any command;
- a unary predicate is defined as: $type = c$ where $c \in dom(type)$;
- each command has the following form:

  Command $\alpha(X_1, X_2, \ldots, X_k)$
  if $r_1 \in [X_{s1}, X_{o1}] \wedge r_2 \in [X_{s2}, X_{o2}] \wedge \ldots r_m \in [X_{sm}, X_{om}]$
  $(X_1.type = t_1) \wedge (X_2.type = t_2) \wedge \ldots \wedge (X_k.type = t_k)$
  then
  op_1; op_2; ...; op_n
  end

  where $t_1, \ldots, t_k \in dom(type)$, and if X_i is created in α, then $(X_i.type = t_i)$ is not included in the condition part.

In this model, since each entity's type is fixed when it is created, each command can be transformed into the following one.

Command $\alpha(X_1 : t_1, X_2 : t_2, \ldots, X_k : t_k)$
if $r_1 \in [X_{s1}, X_{o1}] \wedge r_2 \in [X_{s2}, X_{o2}] \wedge \ldots r_m \in [X_{sm}, X_{om}]$
then
op_1; op_2; ...; op_n
end

That is, this ABAM scheme can be regarded as a general TAM scheme. Therefore, we can say that ABAM is at least as expressive as TAM. Similarly, the dynamic typed access matrix model (DTAM) presented by Soshi [23] can be simulated with ABAM based on an assumption that the type attribute of each entity can be updated, but $dom(type)$ is a fixed set. This means, ABAM is at least as expressive as DTAM, and more expressive than TAM.

4.3 Modelling UCON pre-Authorization Models

Park and Sandhu [17] introduced a family of core models in UCON according to the point of enforcing usage control policies and attribute updates. Pre-authorization models are those where an access control is enforced before a subject starting to use an object and the control decision is determined by the subject and/or object attribute values. According to the updates performed before, during, and after an usage process, there are several pre-authorization core-models, called $preA_1$, $preA_2$, and $preA_3$, respectively. The model without update is called $preA_0$. A authorization policy takes one subject and one object as parameters, and grants an access request by evaluating a set of attribute predicates based on the subject and/or the object. Therefore, logically, $preA_0$, $preA_1$, and $preA_3$ can be simulated with ABAM. Specifically, for a $preA_0$ policy regarding to a subject s's access to an objet o with right r, if the authorization decision is determined by predicates $p_1, \ldots, p_i$ based on s and/or o, then the following ABAM command can simulate this policy.

Command $policy_1(s, o)$
if $p_1 \wedge p_2 \wedge \ldots \wedge p_i$
then
enter r into $[s, o]$
remove r from $[s, o]$
end

In UCON, an access is considered as a process. Predicates are evaluated before an access in $preA_0$, and if all of them are true, the access is approved. After the usage process, the right is revoked by the system. In ABAM, this is simulated by entering the right at the beginning of the command body, and removing it at the end. This mechanism is also implied in HRU [8].

For $preA_1$ and $preA_3$, we simulate them by adding update operations before the entering operation or after the removing operation in a command, as shown below.

Command $policy_2(s, o)$
if $p_1 \wedge p_2 \wedge \ldots \wedge p_i$
then
up_1; up_2; ...; up_n;
enter r into $[s, o]$
op_1; op_2; ...; op_k
remove r from $[s, o]$
up_{n+1}; up_{n+2}; ...; up_{n+m};
end

where up_1, ..., up_{n+m} are update operations on attributes of s or o.

MAC, DAC, RBAC, and some extended versions of these models are simulated with UCON $preA_0$ in [17, 25]. We conclude that ABAM can express these traditional access control models since these pre-authorization models are restricted forms of ABAM.

For $preA_2$ model, since an authorization decision is checked repeatedly during an access, which is called decision continuality, ABAM does not support this feature. We discuss this in Section 6.

4.4 An attribute-based Delegation Example

We show a simple delegation example with an ABAM scheme. Suppose in an organization, there are multiple departments doing various projects and holding confidential information (files) of each project.

The access to these confidential files is based on the security level ($role$) of the members in a group. For the purpose of flexible management, an access right can be delegated by a subject to another subject according to some policies. In this system, each subject is specified by attribute $dept\_id$ as its department ID and $role$ as its security level, and each object has one attribute $ac = \{(r, M_r, N_r)\}$, where r is a right, M_r is the maximum number of subjects that can have right r on this object at the same time, and N_r is the number of subjects that have right r on the object in the current state. For example, an object attribute $o.ac = \{(r, \infty, 10), (w, 1, 1), (v, 3, 1)\}$ means that at any state unlimited number of subjects can "read" this object, but at most one can "write" and three can "review". This attribute also specifies that in the current state there are ten subjects have the right of "read", one has "write", and one has "review". We define a set of functions for each generic right to return the numbers of subjects in each attribute value, e.g., $M_v(o.ac) = 3$, $N_v(o.ac) = 1$.

Consider the following two delegation policies.

1. A subject s_1 can delegate right v of an object o to another subject s_2 in the same department whenever s_2's role is senior to s_1's role.

2. A subject s_1 can delegate right v of object o to another subject s_2 in a different department only when both s_1 and s_2's roles are "manager". s_1 loses the right on the object after delegation.

These policies can be expressed by the following commands in an ABAM scheme.

Command $can\_delegate1\_review(s_1, s_2, o)$
if $v \in [s_1, o] \wedge (s_1.dept\_id = s_2.dept\_id) \wedge dominate(s_2.role, s_1.role) \wedge (N_v(o.ac) < M_v(o.ac))$
then
enter v into $[s_2, o]$
update attribute: $o.ac = \{\ldots, (r, M_v(o.ac), N_v(o.ac) + 1), \ldots, \}$
end

Command $can\_delegate2\_review(s_1, s_2, o)$
if $v \in [s_1, o] \wedge (s_1.dept\_id \neq s_2.dept\_id) \wedge (s_1.role = \text{“}manager\text{”}) \wedge (s_2.role = \text{“}manager\text{”})$
then
enter v into $[s_2, o]$
remove v from $[s_1, o]$
end

where $dominate(x, y)$ is a predicate which is true when x's role is higher than y's in an organization. In both the above commands, a subject s_1 can delegate a "review" permission of an object to a subject s_2 only if s_1 has this right. The first command checks a predicate on the object's attribute. Specifically, if the number of subjects that have right v on this object is less than the maximum number, the delegation can happen, and the object's attribute is updated. In the second command, a delegation grants the right to the new subject, while revokes the right from the original subject.

These commands cannot be modelled by HRU or TAM, since attribute predicates are considered in condition part. Also, they cannot be simulated with UCON because of the multiple objects involved in a single command.

5 Safety Analysis of ABAM

Since safety property is a fundamental and closely related problem to expressive power in an access control model, in this section we describe a restricted case of ABAM and study its safety property. It is shown that the safety of this restricted case is decidable. The expressiveness of this case is also discussed.

For safety analysis, we assume that each attribute has a finite value domain, therefore the set of all possible attribute tuples (say, $\mathcal{A}$), which all of the subjects and objects in a system can assume, is finite. In a particular state, each entity in the system assumes its attribute tuple from $\mathcal{A}$. Any change of a single attribute tuple (i.e., replacing a tuple with another tuple in $\mathcal{A}$) moves the system state from one to another.

5.1 Normalization of an ABAM Scheme

Before safety analysis, we transform each command in an ABAM scheme into a set of *normalized* commands, with a *normalization* process. Consider the following ABAM command.

Command $\alpha(X_1, X_2, \ldots, X_k)$
if $r_1 \in [X_{s1}, X_{o1}] \wedge r_2 \in [X_{s2}, X_{o2}] \wedge \ldots r_m \in [X_{sm}, X_{om}] \wedge$
$p_1 \wedge p_2 \wedge \ldots p_m$
then
$op_1; \ldots; op_n$;
$up_{11}; \ldots; up_{1m_1}$
$\ldots$
$up_{k1}; \ldots; up_{km_k}$
end

where $up_{i1}; \ldots; up_{im_i}$ are update actions for object X_i ($1 \leq i \leq k$), and $op_1, \ldots, op_n$ are non-update actions. Note that if an object is destroyed in α, all of its updates can be ignored in the body.

The normalization process works like this. For any attribute tuple $ATT_1, \ldots, ATT_k \in \mathcal{A}$ of $X_1, \ldots, X_k$, respectively, if all the predicates $p_1, \ldots, p_i$ are true, then a normalized command α_n is generated with the following format:

$\alpha_n(X_1 : ATT_1, \ldots, X_k : ATT_k)$:
if $r_1 \in [X_{s1}, X_{o1}] \wedge r_2 \in [X_{s2}, X_{o2}] \wedge \ldots r_m \in [X_{sm}, X_{om}]$
then
$op_1; \ldots; op_n$;
update attribute tuple $X_1 : ATT_1 \rightarrow ATT'_1$
$\ldots$
update attribute tuple $X_k : ATT_k \rightarrow ATT'_k$
end

where $ATT'_1, \ldots, ATT'_k$ are the attribute tuples of $X_1, \ldots, X_k$ after their corresponding update actions, respectively. If α is a creating command and X_i is created in the body, then we consider $ATT_i(X_i.a) = null$ for all $a \in AT$. Without loss of generality, if X_i is not updated in α, then $ATT'_i = ATT_i$.

This process is repeated with every possible attribute tuple of X_i in $\mathcal{A}$ ($1 \leq i \leq k$). Since each object has a finite number of attribute tuples, for single command this normalization process is guaranteed to terminate, and a finite number of normalized commands are generated. The set of all normalized commands in a scheme is denoted as C_n, and the scheme with all normalized commands is called *normalized scheme*.

It can be easily verified that for an ABAM system, every state that it can reach from the initial state can be reached with the corresponding normalized scheme. Also, each reachable state with the normalized scheme can be reached with the original scheme. Therefore, we can use the normalized commands for safety analysis. From this we assume that an ABAM system always has normalized commands.

5.2 Acyclic ABAM Scheme

We first define the attribute-relation graph of a normalized ABAM scheme, based on which we define the acyclic ABAM scheme.

Definition 10 *For a creating normalized command* $\alpha(X_1 : ATT_1, X_2 : ATT_2, \ldots, X_k : ATT_k)$*,*

- $ATT_1, \ldots, ATT_k$ *are* creating-parent attribute tuples *in* α*;*
- *if* X_i *is created in* α*,* ATT_i' *is a* creating-child attribute tuple *in* α*.*
- *if* X_i *is created in* α *for every* $1 \leq i \leq k$*, then* ATT_i'*s is a* creating-orphan attribute tuples *in* α*.*

Definition 11 *For a normalized command* $\alpha(X_1 : ATT_1, X_2 : ATT_2, \ldots, X_k : ATT_k)$*,* ATT_i' *is said to be an* updating-child attribute tuple *in* α*, and* ATT_i *is said to be an* updating-parent attribute tuple *in* α *for every* $1 \leq i \leq k$*.*

Definition 12 *If the execution of a command* $\alpha(X_1 : ATT_1, X_2 : ATT_2, \ldots, X_k : ATT_k)$ *creates new object* X_i*, then* X_i *is said to be a* child *in* α*. Otherwise, it is a* parent *in* α*.*

Definition 13 *A descendant of an entity* X *is recursively defined as itself or a child of a descendant of* X*.*

Definition 14 *The* attribute-relation (AR) graph (V, E) *of an ABAM scheme* (R, AT, P, C) *is a directed graph with the set of vertices* V *to be the set of all attribute tuples (the set* $\mathcal{A}$ *defined early in this section), and the set of edges* $E \subseteq V \times V$*. A pair* $(v_1, v_2) \in E$ *iff*

- $\exists \alpha \in C$*,* v_1 *is a creating-parent attribute tuple in* α*, and* v_2 *is a creating-child attribute tuple in* α*; or*
- $\exists \alpha \in C$*,* v_1 *is an updating-parent attribute tuple in* α*, and* v_2 *is an updating-child attribute tuple in* α*.*

From an AR graph, it can be seen that if there is a cycle with creating-parent attribute tuples, then an entity in a state may create an infinite number of objects. Also, objects with orphan attribute tuples can be created infinitely. The existence of cycles and orphan attribute tuples in the AR graph is critical to decide whether the number of entities in a protection system are finite or not. Note that a self loop is regarded as a special case of a cycle with length one.

Lemma 1 *Consider an AR graph that has no cycles containing creating-parent attribute tuples. If* $\alpha(X_1 : ATT_1, X_2 : ATT_2, \ldots, X_k : ATT_k)$ *is a creating normalized command, then* $ATT_i' \neq ATT_i$*. That is, all the creating-parent attribute tuples in a creating commands have to be updated to different ones.*

Proof: The lemma is proved by contradiction. Assume that X_i is a parent in α, and the body of α has no update attribute operations on X_i (that is, the $ATT_i' = ATT_i$ in the normalized command). Then, ATT_i is both a updating-parent attribute tuple and a updating-child attribute tuple with respect to update attributes in (from Definition 11). As ATT_i is a creating-parent attribute tuple (since X_i is a parent), this means that there is a cycle in the AR graph with a creating-parent attribute tuple. This is a contradiction to our assumption. □

From the above lemma, it can be concluded that, if an AR graph has no cycles containing creating-parent attribute tuples in a scheme, then the creating commands make irreversible changes on the attributes of parent entities.

Definition 15 *An ABAM scheme is said to be* acyclic *if and only if*

1. *there are no creating-orphan attribute tuples, and*
2. *the AR graph of the scheme has no cycle that contains creating-parent attribute tuples.*

The definition of an acyclic protection system means that the set of all entities derived from an initial state have distinct attribute tuples. No two entities derived from the initial state have the same attribute tuples in any state of the system.

5.3 Safety Analysis of Acyclic ABAM Systems

This section gives a formal proof of the decidability of the safety problem for a restricted case of ABAM.

Theorem 1 *The safety problem for an ABAM system is decidable if its scheme is acyclic.*

Proof: To prove this, we first show that the number of entities in an arbitrary protection state of an acyclic protection system has an upper bound. Since there are no orphan attribute tuples, every object is a descendant of an entity in the initial state. Now, if we can prove that the number of descendants of an arbitrary entity X_i in the system is finite, then the total number of entities in an arbitrary protection state is finite.

Let N_{max} be the maximum number of create operations in a command in the authorization scheme. This number is finite since the number of primitive operations in the body of any command is finite. Consider an existing entity X_i. If a creating command α can be executed with X_i as a parameter, the attribute tuple of X_i is always a creating-parent attribute tuple in α. From Lemma 1, α must update at least one attribute of X_i by replacing ATT_i with ATT_i' from set $\mathcal{A}$, and $ATT_i' \neq ATT_i$.

Thus, if multiple creating commands are executed with X_i as a parameter, the attribute tuple of X_i is changed for each of the creating commands. The bound on the number of execution of these creating commands depends on the number of times the attribute tuple of X_i can be changed. Since the total number of attribute tuples is a finite number $|\mathcal{A}|$, the maximum number of creating commands that can be executed with X_i as a parameter is $|\mathcal{A}| - 1$. Then the maximum number of direct children of during the lifetime of the system is $N_{max} \times (|\mathcal{A}| - 1)$. Also, the maximum number of generations of descendants of X_i is $|\mathcal{A}|$, since if it is greater than $|\mathcal{A}|$, there must exist two entities with the same attribute tuple, which contradicts the acyclic scheme property assumed. Thus, the total number of descendants of an arbitrary entities in the system is finite, that is, the number of subjects and objects in an arbitrary protection state of the system has an upper bound. This implies that the total number of distinct protection states of the system is also finite. Hence we can check whether or not a particular subject has a certain right over a particular object in every reachable state from the initial state. Thus, the safety problem is decidable. □

5.4 Complexity of Safety Problem

Harrison, Ruzzo, and Ullman have shown that monotonic mono-operational HRU system without creations has NP-complete safety analysis in the number of commands [8, 20]. This can be polynomially reduced to an acyclic ABAM system without creating commands.

Theorem 2 *The complexity of safety analysis for acyclic ABAM systems with finite domain of each attribute is NP-hard in the number of commands in the scheme.*

Proof: As we have shown in Section 4, HRU is a special case of ABAM. Thus, for a monotonic mono-operational HRU scheme without creations, it can be polynomially reduced to an ABAM scheme, where

1. all attributes have finite domains,

2. all predicates are trivially true, and

3. no creating and destroying operations in all commands.

This is a restricted model of our decidable ABAM proven above since there are no creating-parent attribute tuples in the scheme. Since monotonic mono-operational HRU is NP-complete in the number of commands, we can conclude that this restricted ABAM is NP-hard. Therefore, an acyclic ABAM system with finite attribute domain has NP-hard safety, since it subsumes this special case. □

5.5 Expressiveness of Acyclic ABAM Models

We have considered a restricted case of ABAM for which the safety problem is decidable. In this restricted case, (i) each attribute takes values from a finite domain, (ii) the protection system has no orphan attribute tuples, and (iii) the AR-graph has no cycles that contain creating-parent attribute tuples. In other words, the protection system is acyclic.

Since our decidable model allows non-monotonic operations in a system, this introduces more expressive power than existing decidable HRU and TAM models where only monotonic operations are allowed. From Section 4.2 we know that a restricted model of ABAM is equivalent to DTAM, then we can say that our restricted model has at least the expressive power of acyclic DTAM. Apart from this, the restricted model also inherits the features of the usage control model such as attribute mutability. Thus, we can say that the restricted model is more expressive than these existing models.

6 Discussions

Our model inherits UCON's attribute mutability feature and involves multiple subjects and objects in single access control decision. In this sense our model is even more flexible then UCON in some pre-authorization applications as shown in Section 4. On the other side, like other tradition access control models, ABAM does not capture the decision continuity (ongoing authorizations) that is proposed in UCON [17]. Continuity means that authorization decisions are continuously checked and enforced throughout an access process. This property is useful for the control of relatively long-lived usage or for immediate revocation of usage when some authorization conditions are not satisfied.

The reason that ABAM does not support ongoing authorizations is that the condition part and the primitive operations are separated in a command structure (see section 3.7). The condition is checked once and the operations are executed sequentially if the condition is true. A possible approach to enforce ongoing authorizations in our model is to interleave conditions with primitive operations such that conditions of the authorizations are always examined during the execution of a command. The conditions as well as corresponding primitive operations can be structured in a nested form in continuous authorizations such that an authorization can be revoked at any time when a nested condition is not satisfied (e.g., as side-effects of attribute updates or primitive operations). While interleaving of conditions and operations may provide more expressive power, the safety property of such model is not yet clear. It remains an interesting topic for future study.

7 Conclusions

The contribution of this paper is three-fold. First, we propose a new model of attribute-based access control. Then, we study the expressive power of this new model. Finally, we analyze the safety property of this model

with a restricted case. The main difference between this and previous models is that attributes are introduced into access control matrix, primitive operations, and commands. Traditional access control models such as HRU, TAM and DTAM are shown to be special cases of our model. Besides these, the new model has some features of usage control which makes it more general for modern information systems. The safety problem of this new model is proven to be decidable for a restricted case where all attribute domains are finite, and the attribute relationship graph allows no cycles containing creating-parent attribute tuples. It has also been shown that the restricted case has broader expressive power than DTAM.

References

[1] P. E. Amman, and R. S. Sandhu. The Extended Schematic Protection Model. Technical Report, George Mason University, 1990.

[2] P. E. Amman, and R. S. Sandhu. Safety Analysis for the Extended Schematic Protection Model. In Proceedings of IEEE Symposium on Research in Security and Privacy, pp.87-97, 1991.

[3] G. R. Andrews. COPS - A Protection Mechanism for Computer Systems. PhD thesis and Tech. Report 74-07-12, Computer Science Program, University of Washington, Seattle, Washington, July 1974.

[4] D. E. Bell, and L. J. LaPadula. Secure Computer Systems, vol.I: Mathematical Foundations and vol.II: A Mathematical Model. MITRE Corp. Tech. Rep. MTR-2547, 1973.

[5] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralised Trust Management. In Proceedings of IEEE symposium on Security and Privacy, pp. 164-173, May 1996.

[6] J.B. Dennis, and E.C. Van Horn. Programming Semantics for Multiprogrammed Computations. In Communications of the ACM, 9, 143-155, March 1966.

[7] G.S. Graham, and P.J. Denning. Protection - Principles and Practice. In AFIPS conference proceedings, SJCC, Vol. 40, AFIPS Press, pp. 417-429, 1972.

[8] M. H. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in Operating Systems. Communications of ACM 19(8) , pp. 461-471, 1976.

[9] M. H. Harrison, and W. L. Ruzzo. Monotonic Protection Systems. In DeMillo et al. (Editors), Foundations of Secure Computations, Academic Press, pp.337-365, 1978.

[10] S. Jajodia, P. Samarati, and V. Subrahmanian, A Logical Language for Expressing Authorizations, IEEE Symposium On Research in Security and Privacy, Oakland, California, 1997.

[11] A.K. Jones. Protection in Programmed Systems. PhD Thesis, Dept. of Computer Science, Carnegie-Mellon University, Pittsburg, June 1973.

[12] Mohammad A. Al-Kahtani and Ravi Sandhu, A Model for Attribute-Based User-Role Assignment, Annual Computer Security Applications Conference, 2002.

[13] M. Kaplan. IBM Cryptolopes, Superdistribution and Digital Rights Management, online, available at http://www.research.ibm.com/people/k/kaplan/cryptolope-docs/crypap.html. 1996.

[14] B.W. Lampson. Protection. In Proceedings of Fifth Princeton symposium on Information Sciences and Systems, Princeton University, pp. 437-443, March 1991.

[15] N. Li, W. H. Winsborough and J. C. Mitchell, Design of a role-based trust management framework, In Proceedngs of the 2002 IEEE Symposium on Security and Privacy, pp 114-130, May 2002.

[16] R. J. Lipton, and L. Snyder. A Linear Time Algorithm for deciding Subject Security. Journal of ACM, Vol. 24, No. 3, pp. 455-464, 1977.

[17] J. Park, and R. Sandhu. The $UCON_{ABC}$ Usage Control Model, ACM Transactions on Information and Systems Security, Feb, 2004.

[18] G. J. Popek. Correctness in Access control. In Proceedings of National Computer Conference, pp. 236-241, 1974.

[19] R. S. Sandhu The Schematic Protection Model: Its Definition and Analysis for Acyclic Attenuating Schemes. In Journal of ACM, Vol 35, No. 2, pp. 404-432, April 1988.

[20] R. S. Sandhu. The Typed Access Matrix Model. In Proceedings of the IEEE Symposium on Security and Privacy, pp. 122-136, May 1992.

[21] Paul Schneck. Persistent Access Control to Prevent Piracy of Digital Information. In Proceedings of the IEEE, vol.87, No.7, pp. 1239-1250, July 1999.

[22] Sibert et al. 1995. The Digibox: A Self-Protecting Container for Information Commerce. In Proceedings of USENIX workshop on Electronic Commerce.

[23] M. Soshi, Safety Analysis of the Dynamic-Typed Access Matrix Model, 6th European Symposium on Research in Computer Security, LNCS 1895, 2000.

[24] L. Wang, D. Wijesekera, and S. Jajodia , A logic-based framework for attribute based access control, In Proceedings of the 2004 ACM workshop on Formal methods in security engineering, 2004.

[25] X. Zhang, J. Park, F. Parisi-Presicce, and R. Sandhu. A Logical Specification for Usage Control, To appear in Proceedings of 9th ACM Symposium on Access Control Models and Technologies 2004.