On the Relationship between Finite Domain ABAM and PreUCON$_\mathrm{A}$

Asma Alshehri and Ravi Sandhu

Institute for Cyber Security & Department of Computer Science,
University of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78249, USA
nmt366@my.utsa.edu, ravi.sandhu@utsa.edu

Abstract. Several access control models that use attributes have been proposed, although none so far is regarded as a definitive characterization of attribute-based access control (ABAC). Among these a recently proposed model is the attribute-based access matrix (ABAM) model [14] that extends the HRU model [4] by introducing attributes. In this paper we consider the finite case of ABAM, where the number of attributes is finite and the permissible values (i.e., domain) for each attribute is finite. Henceforth, we understand ABAM to mean finite ABAM. A separately developed model with finite attribute domains is PreUCON$_\mathrm{A}$ [10], which is a sub-model of the usage control UCON model [9]. This paper explores the relationship between the expressive power of these two finite attribute domain models. Since the safety problem for HRU is undecidable it follows safety is also undecidable for ABAM, while it is known to be decidable for PreUCON$_\mathrm{A}$ [10]. Hence ABAM cannot be reduced to PreUCON$_\mathrm{A}$. We define a special case of ABAM called RL-ABAM2 and show that RL-ABAM2 and PreUCON$_\mathrm{A}$ are equivalent in expressive power, but each has its own advantages. Finally, we propose a possible way to combine the advantages of these two models.

1 INTRODUCTION

Attribute-Based Access Control (ABAC) is a form of access control that has recently caught the interest of both academic and industry researchers. High-level definitions and descriptions of ABAC are generally accepted, but heretofore there has been no particular unified model or standardization of ABAC. The National Institute of Standards and Technology (NIST) recently described a high level access control model that uses attributes [5, 6]. Jin et al [7] have proposed a unified ABAC model that can be configured to the traditional access control models (i.e., DAC, MAC and RBAC). Researchers have also studied combining attributes with RBAC. Kuhn et al [8] presented models that combine ABAC and RBAC in various ways, while Yong et al [13] proposed extending the roles of the RBAC with attributes. Al-Kahtani et al [1] introduced the notion of using attributes in user-role assignment of RBAC model. Chadwick et al [3] describe the use of X.509 certificates to enforce RBAC. Bennett et al [2] showed that online social network policies can be cast in an ABAC framework. Thus there has been a tradition of research on combining or relating attributes to various access control models, old and new.

A novel approach to combining attributes with the access matrix was developed by Zhang et al [14], who defined the attribute-based access matrix (ABAM) model by adding attributes to the classic HRU model [4]. In the HRU model each cell $[s_i, o_j]$ of the access matrix contains a set of rights that subject s_i can exercise over object o_j. In general, a subject is also an object while every object is not necessarily a subject. Subjects and objects are collectively called entities. ABAM additionally associates a set of attributes $ATT(o)$ with each entity o. A notable aspect of ABAM is that its commands not only test for and modify rights in access matrix cells like in HRU, but can further test for and modify attribute values. In the finite ABAM the set of attributes is finite and each attribute can take values from only a finite fixed set. Henceforth we understand ABAM to mean finite ABAM. The ABAM model is reviewed in Section 2.1.

The features of attribute testing and modification, also called attribute mutability, were adapted in ABAM [14] from the earlier UCON model [9]. UCON incorporates various additional features such as ongoing authorization and updates, as well as obligations and conditions. Here we focus on a sub-model of UCON called PreUCON_A [10, 9] where attribute testing and modification are carried out prior to allowing access. Similar to finite ABAM, in finite PreUCON_A the set of attributes is finite and each attribute of an entity can only take on a finite set of permissible values. Henceforth, we understand PreUCON_A to mean finite PreUCON_A. PreUCON_A is reviewed in Section 2.2.

In this paper we investigate the theoretical relationship between ABAM and PreUCON_A. Our first observation is that ABAM is an extension of HRU and thereby inherits the undecidable safety results of HRU. On the other hand PreUCON_A is known to have decidable safety analysis [10]. It follows that ABAM cannot be reduced to PreUCON_A. On the other hand, we show how PreUCON_A can be reduced to ABAM (Section 3). This construction inspires us to define a restricted version of ABAM named RL-ABAM2, which stands for right-less ABAM with two parameters as will be explained (Section 4). We then prove that PreUCON_A and RL-ABAM2 theoretically have equivalent expressive power (Section 5). Section 6 concludes the paper.

2 BACKGROUND

In the following we respectively review the ABAM and PreUCON_A models.

2.1 The ABAM Model

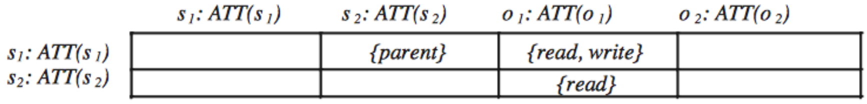
ABAM is defined in terms of access control matrix and commands in the tradition of HRU [4], TAM [11] and other access matrix based formal models. The basic components of the ABAM model are subjects and their attributes, objects and their attributes, access rights, access matrix, primitive operations and commands. These are explained below.

Subjects and Objects. Entities in ABAM are objects O and subjects S. Subjects are active entities that can invoke access requests or execute permissions on objects. Subjects can be the target of access requests so $S \subseteq O$. Objects that are not subjects are called pure objects. When an object is created, a unique identity (ID) recognized by the system is given to that object and cannot be changed after the creation. This ID is never reused.

Attributes and Attribute Tuples. In ABAM, the set of attributes G_{ATT} is attached to each entity. Each attribute is a variable with a specific data type. An attribute of an entity can be assigned an atomic value v_i, which comes from domain V_i for that attribute, or a set value $\{v_1, .., v_i, .., v_k\} \subseteq V_i$. Also, a *null* value will be assigned if the entity does not have that attribute. For entity o, the set of attributes of o come from $G_{ATT}= \{a_1, .., a_i, .., a_n\}$, and the value v_i of each attribute a_i is from the domain $V(a_i) = \{v_1, .., v_i, .., v_m\}$. The ordered set of attributes and domains $G_v = [a_1{:}V(a_1)$, ..., $a_i{:}V(a_i)$, ..., $a_n{:}V(a_n)]$ is a combination of G_{ATT} and $V(a_i)$. The attribute value tuple for entity o is $ATT(o)$ $= (a_1 = v_1$, ... , $a_i = v_i$, ... , $a_n = v_n)$, where $v_i \in V_i$ or $v_i \subseteq V_i$ for $1 \leq i \leq n$. The result of updating a_i from v_i to v_i' changes $ATT(o)$ to $ATT'(o) = ($ $a_1 = v_1$, ... , $a_i = v_i'$, ... , $a_n = v_n)$. An entity attribute is denoted as *ent.att* where *ent* refers to entity name and *att* is the attribute name.

Rights and Access Matrix. An access matrix is a matrix with columns representing all objects (subjects and pure objects), and rows representing the set of all subjects. Each object in the columns and rows is associated with its attribute tuple. Access rights in the $[s_i, o_j]$ cell of the matrix specify the access that subject s_i has to object o_j. All entities in rows (subjects) can access other entities in the column (subjects and pure objects) by executing given access rights (e.g., read, write, execute). The set of all access rights is denoted by R and each cell $[s, o]$ is a subset of R. Figure 1 shows an example of an access matrix [14].

Attribute Predicates. A predicate P is a Boolean expression constructed using attributes and constants with appropriate relation symbols. There are two kinds of predicates. A unary predicate has one attribute variable and a constant, e.g. $Alice.credit \leq 100$. A binary predicate has two different attribute variables, e.g. $s1.roles \subset s2.roles$. Binary predicates can be built over attributes from the same entity or two different entities.



	s_1: ATT(s_1)	s_2: ATT(s_2)	o_1: ATT(o_1)	o_2: ATT(o_2)
s_1: ATT(s_1)		{parent}	{read, write}	
s_2: ATT(s_2)			{read}	

Fig. 1. ABAM access matrix [14]

Primitive Operations. A primitive operation is the basic action that a subject can execute over an object which cause changes in the status of the access matrix. The primitive operations are defined as follows.

1. Enter r into [s,o]: Enters generic right r into cell [s, o] in the access matrix.
2. Delete r from [s,o]: Deletes generic right r from cell [s, o] in the access matrix.
3. Create subject s:ATT(s): Creates a new subject s with attribute tuple $ATT(s)$.
4. Destroy subject s: Removes subject s and its attribute tuple from the system.
5. Create object o:ATT(o): Creates a new object o with attribute tuple $ATT(o)$.
6. Destroy subject s: Removes object o and its attribute tuple from the system.
7. Update attribute ent.att $= v'$: Updates the attribute tuple $ATT(o)$ to $ATT'(o)$ $= (a_1 = v_1, \ldots , a_i = v'_i, \ldots , a_n = v_n)$ where $v_i \in V_i$ and $v_i \neq v'_i$.

The first six are essentially similar to their counterparts in HRU, whereas the seventh is new to ABAM.

Commands. A command in ABAM involves three parts: parameters (entities with possibly new attribute values), conditions, and a sequence of primitive operations. ABAM commands allow primitive operation to be executed if the condition on existing rights is satisfied, as well as the specified predicates on attributes of the entities evaluate to true. The set of all commands is $G_\alpha = \{\alpha_1, \alpha_2, .., \alpha_h\}$. Each individual command is defined as follows.

Command $\alpha_i(X_1 : ATT(X_1), X_2 : ATT(X_2), .., X_k : ATT(X_k))$
if $r_1 \in [X_{s1}, X_{o1}] \wedge r_2 \in [X_{s2}, X_{o2}] \wedge ...r_m \in [X_{sm}, X_{om}] \wedge p_1 \wedge p_2 \wedge ...p_n$
then $op_1; op_2; ...; op_l$ **end**

The name of the command is α_i. $X_1, X_2, \ldots, X_k$ are subject or object parameters; $r_1, r_2, ..., r_m$ are generic rights; $s_1, s_2, ..., s_m$ and $o_1, o_2, ..., o_m$ are integers between 1 and k; $ATT(X_1), ATT(X_2), ..., ATT(X_k)$ specify new values of attributes for the respective entities (if any is updated by the command); $p_1, p_2, ..., p_n$ are predicates built over old or new attribute tuples of $X_1, X_2, ..., X_k$. The "if" part of the command is called the condition of α. Update operations can update an attribute from an old value $v_i \in V_i$ to a different new value $v'_i \in V_i$ or from an old value set $\{v_1, v_2, .., v_r\} \subseteq V_i$ to a new different subset of V_i. The operations $op_1; op_2; ...; op_l$ in the body of the command are executed sequentially and the entire command executes atomically. Each op_i consists of one of the seven primitive operations enumerated above.

Command Example. The following ABAM command enables the first subject to update attribute a_2 of the second subject to v'_i, provided the specified condition is true.

Command $Update(s_1 : ATT(s_1), s_2 : ATT(s_2))$
if $r_1 \in [X_{s1}, X_{s_2}] \wedge s_1.a_1 = v_i \wedge s_1.a_2 \leq s_2.a_2$
then *update attribute* $s_2.a_2 = v'_i$ **end**

An ABAM command allows only conjunctive form of condition. In case of a disjunctive form of condition, we need to have one command for each component condition. For negated predicates, ABAM command accomodates it by simply defining a normal predicate for a negated one. Therefore, without loss of generality, we can consider the condition of ABAM command to be an arbitrary propositional logic formula.

2.2 The PreUCON$_{\mathbf{A}}$ Model

We now describe the PreUCON$_{\mathrm{A}}$ model.

Subject, Objects, Attributes, and Rights. The PreUCON$_{\mathrm{A}}$ model has objects as resources and subjects as user processes. Similar to the ABAM model, subjects are a subset of objects. Each object has a finite set of attributes and a unique name. Object attributes can be accessed by using dot notation to associate object name with attribute name, as in $name_{object}.name_{attribute}$, e.g., o.security = ‘high’. This model supports the dynamic creation and deletion of objects. The permission defined over an object is called a usage right.

Usage Control Scheme. There are three components of a usage control scheme U_{Θ}.

- an object schema OS_{Δ},
- a set of usage rights UR = $\{r_1, r_2, ..., r_m\}$, and
- a set of usage control commands $\{UC_1, UC_2, ..., UC_n\}$

The object schema OS_{Δ} is the combination of the attributes of objects and domains from which attribute values come. $OS_{\Delta} = (a_1 : \Omega_1, a_2 : \Omega_2, ..., a_n : \Omega_n)$, where each a_i is the name of an attribute, and Ω_i is the domain of a_i that has a finite set of values which can be assigned to the attribute a_i. Each object will have an ordered attribute value tuple $AVT =< v_1, v_2, ..., v_n >$, where n is the number of attributes in the object schema and each $v_i \in \Omega_i$. Attributes can be assigned to an atomic value or a set value. Also, attributes can be set to a default value from the domain at creation time.

Usage rights defines the rights r_i that can be granted by a usage control command. UR is finite. Giving a right to a subject to be executed on an object depends on the attribute value of subjects and objects as specified by usage control commands discussed below.

The usage control commands comprise a finite set of commands. Each command has a name that is linked with the authorized right r when executing this command. There are two formal parameters for each command, s and o. Subject s is the actor that seeks to access the target object o with right r. Also, commands can be either non-creating commands in which the object o exists before the execution of the command, or creating commands in which the object o is created during the execution of the command. The structure of creating and non-creating commands is shown in Table 1.

Non-Creating Commands	Creating Commands
Command-Name$_r$(s, o)	**Command-Name$_r$(s)**
PreCondition: $f_b(s,o) \rightarrow \{yes, no\}$;	**PreCondition:** $f_b(s) \rightarrow \{yes, no\}$;
PreUpdate:	**PreUpdate:** create o;
$s.a_{i_1} := f_1, a_{i1}(s,o)$;	$s.a_{i_1} := f_1, a_{i1}(s)$;
...	...
$s.a_{i_p} := f_1, a_{ip}(s,o)$;	$s.a_{i_p} := f_1, a_{ip}(s)$;
$o.a_{j_1} := f_2, a_{j1}(s,o)$;	$o.a_1 := f_2, a_1(s)$;
...	...
$o.a_{j_q} := f_2, a_{jq}(s,o)$;	$o.a_n := f_2, a_n(s)$;

Table 1. PreUCON$_\mathrm{A}$ commands

In the PreCondition section, the Boolean function $f_b(s,o)$ takes the attribute values of s and o as input and returns true or false. In case of false, the command terminates without executing any updating or granting rights r, while in case of true, the update attribute operations in the PreUpdate section will be executed, and the operation permitted by right r is allowed. Zero or more of the attributes of the input s and o are updated individually to new values that are calculated from their old values, which existed before the command execution.

The structure of a creating command is mostly similar. The input parameter is only s, and the function $f_b(s)$ is a Boolean function that takes the attributes of s as an input and returns a $true$ or $false$ value. In case of $false$, the command terminates without executing any creating, updating, or granting rights r, while in the case of true, the command of creating an object should be executed before doing any updating of the object attributes. Zero or more of the attributes of the input s and new object o are updated individually to new values that are calculated from old values of s, which existed before the command execution.

3 Expressing PreUCON$_\mathrm{A}$ IN ABAM

In this section we consider how to express PreUCON$_\mathrm{A}$ in ABAM. The reductions we consider in this paper are state-matching reductions [12], which is the accepted formal criteria for theoretical equivalence of access control models.

There are two challenges in reducing PreUCON$_\mathrm{A}$ to ABAM. First, the formulas in the PreCondition part of Table 1 are arbitrary computable Boolean functions, whereas ABAM only permits propositional logic formulas. Second, the update functions in the body of a PreUCON$_\mathrm{A}$ command are arbitrary computable functions. In ABAM only specific new values are allowed in the update operation. However, due to the finite domain assumption these functions from PreUCON$_\mathrm{A}$ can be computed for all possible attribute values of s and o, and the results can be "compiled" into multiple ABAM commnds. We show how to do this for the body of the PreUCON$_\mathrm{A}$ command. An analogous construction applies to the PreCondition part, but is not shown here for lack of space and straightforward similarity.

The attributes assignment formulas can be handled by having an ABAM command for each possible combination of attribute values. Consider an update operation $s.a_i := f(s, o)$. There are only a finite number of possible results for the value of $s.a_i$, depending on the value domain of a_i and whether a_i is atomic or set valued. For each possible value of a_i we can determine which input combinations of the attribute values of s and o will produce that result, if only by exhaustive enumeration of f for these combinations. The example below illustrates this idea more concretely.

The PreUCON$_{\mathrm{A}}$ components are subjects (S), objects (O), usage rights (UR) and an object schema OS_{Δ}. To express PreUCON$_{\mathrm{A}}$ in ABAM, we can define the following analogous ABAM components where the subscript ABAM is used to distinguish the ABAM component from the corresponding PreUCON$_{\mathrm{A}}$ component.

- $O_{ABAM} = O$ and $S_{ABAM} = S$
- $R_{ABAM} = UR = \{ur_1, ur_2, ..., ur_k\}$
- M_{ABAM} with a row for every S_{ABAM} with its attribute tuple, and a column for every O_{ABAM} with its attribute tuple.
- $[s_i, o_j] = \phi$, where $s_i \in S_{ABAM}$, and $o_i \in O_{ABAM}$
- $G_{ATT} = \{a_1, .., a_i, .., a_n\}$
- $G_V = OS_{\Delta} = [a_1 : \Omega_1, a_2 : \Omega_2, ..., a_n : \Omega_n]$

We illustrate the construction of ABAM commands by the following example. Let the object schema $OS_{\Delta} = [a_1 : \{1, 2\}, a_2 : \{2, 3\}, a_3 : \{1, 2, 3\}]$ and usage rights $UR = \{update\}$. The initial values for s and o attributes are [1,2,3] and [2,3,1] respectively for $[a_1, a_2, a_3]$. Command$_{update}$ is as follows:

Command$_{update}$(s, o)
PreCondition: $s.a_1 \leq 2 \vee o.a_2 \leq 3$
PreUpdate: $o.a_3 := max(s.a_3, o.a_3);$

Since $s.a_3 = 3$ and $o.a_3 = 1$, the new value of $o.a_3$ is 3 which is the maximum of two values. The corresponding ABAM components of the PreUCON$_{\mathrm{A}}$ schema will be as follows.

- $O_{ABAM} = O$ and $S_{ABAM} = S$
- $R_{ABAM} = UR = \{update\}$
- M_{ABAM} with a row for every S_{ABAM} with its attribute tuple, and a column for every O_{ABAM} with its attribute tuple.
- $[s_i, o_j] = \phi$, where $s_i \in S_{ABAM}$, and $o_i \in O_{ABAM}$
- $G_{ATT} = \{a_1, a_2, a_3\}$
- $G_V = OS_{\Delta} = [a_1 : \{1, 2\}, a_2 : \{2, 3\}, a_3 : \{1, 2, 3\}]$

The possible ABAM commands for PreUCON$_{\mathrm{A}}$ command$_{update}$ are given in Table 2. Since attribute a_3 has only three possible values we need three ABAM commands. This construction easily extends to multiple attributes. It is evident will need a large number of commands for each PreUCON$_{\mathrm{A}}$ command that uses such formulas.

Updating to value 1	Updating to value 2	Updating to value 3
Command $update(s : ATT(s), o : ATT(o))$ **if** $s.a_1 \leq 2 \vee o.a_2 \leq 3 \wedge (s.a_3 = 1 \vee o.a_3 = 1)$ update attribute $o.a_3 = 1$ enter *update* into $[s, o]$; delete *update* from $[s, o]$; **end**	**Command** $update(s : ATT(s), o : ATT(o))$ **if** $s.a_1 \leq 2 \vee o.a_2 \leq 3 \wedge ((s.a_3 = 1 \wedge o.a_3 = 2) \vee (s.a_3 = 2 \wedge o.a_3 = 1)($ update attribute $o.a_3 = 2$ enter *update* into $[s, o]$; delete *update* from $[s, o]$; **end**	**Command** $update(s : ATT(s), o : ATT(o))$ **if** $s.a_1 \leq 2 \vee o.a_2 \leq 3 \wedge ((s.a_3 = 1 \wedge o.a_3 = 3) \vee (s.a_3 = 2 \wedge o.a_3 = 3) \vee (s.a_3 = 3 \wedge o.a_3 = 1) \vee (s.a_3 = 3 \wedge o.a_3 = 2) \vee (s.a_3 = 3 \wedge o.a_3 = 3))$ update attribute $o.a_3 = 3$ enter *update* into $[s, o]$; delete *update* from $[s, o]$; **end**

Table 2. Possible ABAM commands

4 Right-Less ABAM with Two Parameters (RL-ABAM2)

PreUCON$_\mathrm{A}$ has the ability to grant a non-persistent right for each command. In other words, by the end of any command execution, the given right is taken back from the actor. In contrast, an ABAM command has the power of granting one or more rights to the actor, maintining the given rights in the corresponding cell of the actor, and permitting two or more parameters (more targets) for each command. These ABAM features will cause difficulties for expressing ABAM in PreUCON$_\mathrm{A}$. Moreover, unrestricted use of rights in ABAM will result in undecidable safety as in HRU [4], whereas PreUCON$_\mathrm{A}$ has decidable safety [10]. Therefore, in general it is not possible to reduce ABAM to PreUCON$_\mathrm{A}$. These considerations lead us to focus on a restricted form of ABAM inspired by the construction in the previous section.

4.1 RL-ABAM2 Definition

RL-ABAM2 is the reduced model of ABAM, where RL indicates "right less" and the two denotes the number of parameters required in a command. Object, subject, attributes, attribute tuples, rights, access matrix, predicates, and primitive operations are all the same as in ABAM. However, an RL-ABAM2 command has more limited characteristics than an ABAM command in terms of number of parameters, the if statement section, and the existence of rights. In RL-ABAM2, a command is defined as follows:

```
**Command** $\alpha_i(X_1 : ATT(X_1), X_2 : ATT(X_2))$
if $p_1 \wedge p_2 \wedge ...p_n$
**then**
$op_1; op_2; ...; op_l$ ;
enter $r_1$ into $[X_1, X2]$;
delete $r_1$ from $[X_1, X2]$;
```

```
...
enter $r_k$ into $[X_1, X2]$;
delete $r_k$ from $[X_1, X2]$;
end
```

In the above RL-ABAM2 command, the number of parameters is only two. Moreover, the rights check part is eliminated in the "if" statement section, so the predicates P are the only part that appears. The body of the command will have all kinds of operations, but every right entered into any cell needs to be deleted prior to the end of the command. In general, the RL-ABAM2 model is a special case of ABAM model.

4.2 Expressing PreUCON$_{\mathbf{A}}$ in RL-ABAM2

In Section 3, we discussed how to express the PreUCON$_{\mathrm{A}}$ in the ABAM. In fact, the result of expressing PreUCON$_{\mathrm{A}}$ commands to ABAM commands is RL-ABAM2 commands which have two parameters, no check for rights, and a delete right operation for each entered right. Thus, we can state that Section 3 is already expressing PreUCON$_{\mathrm{A}}$ in RL-ABAM2.

5 Expressing RL-ABAM2 in PreUCON$_{\mathbf{A}}$

In this section we show how to reduce RL-ABAM2 to PreUCON$_{\mathrm{A}}$.

5.1 General Construction

Given an RL-ABAM2 schema with the following components: objects $O_{RL-ABAM2}$, subjects $S_{RL-ABAM2}$, access rights $R_{RL-ABAM2} = \{r_1, .., r_k\}$, attributes tuple $ATT(o_i) = < a_1 = v_1, .., a_n = v_n >$, where $o_i \in O_{RL-ABAM2}$, and a list of all attributes which are linked with their domains $G - V_{\{RL-ABAM2\}} = [a_1{:}V(a_1)$, ..., $a_i{:}V(a_i)$, ..., $a_n{:}V(a_n)]$, each RL-ABAM2 commands will have the following structure:

```
Command $\alpha_i$ $(s_i : ATT(s_i), o_j : ATT(o_j))$
if $p_1 \wedge p_2 \wedge ...p_n$
then
create object $X2 : ATT(X2)$;
update attribute $s_i.a_k = v'_i$;
update attribute $o_i.a_s = v'_j$;
enter $r_i$ into $[s_i, o_j]$;
delete $r_i$ from $[s_i.o_j]$;
end
```

This structure for RL-ABAM2 commands can be assumed without loss of generality. The create operation (if present) comes first, followed by update operations, and at the end, all enter and delete operations. For each parameter, zero or more attributes of $o_j \in O_{RL-ABAM2}$ or $s_i \in S_{RL-ABAM2}$ can be updated from v_i to v_i', as well as one or more rights $r_i \in (R_{RL-ABAM2})$ can be entered into cell $[s_i, o_j]$ and deleted.

The corresponding PreUCON$_\mathrm{A}$ components of the RL-ABAM2 schema are as follows:

- Entity in PreUCON$_\mathrm{A}$ are objects $O_{Pre\_UCON_A}$
- $O_{Pre\_UCON_A} = O_{RL-ABAM2}$
- $S_{Pre\_UCON_A} = S_{RL-ABAM2}$
- $UR_{Pre\_UCON_A} = R_{RL-ABAM2}$
  item $OS_\Delta = G - V_{\{RL-ABAM2\}}$

As discussed above, RL-ABAM2 has the power of entering and deleting many rights in one command, while a PreUCON$_\mathrm{A}$ command grants a single right. In the case of executing many operations over rights in the body of RL-ABAM2, applying a singleton PreUCON$_\mathrm{A}$ command can only cover one of the RL-ABAM2 rights. Consequently, multiple PreUCON$_\mathrm{A}$ commands are required to cover RL-ABAM2 rights. To preserve atomicity of the RL-ABAM2 command specific attributes are added as well as a special object for synchronuzation. Some parts of the corresponding PreUCON$_\mathrm{A}$ components of the RL-ABAM2 schema are extended as follows:

- Entity in PreUCON$_\mathrm{A}$ are objects $O_{Pre\_UCON_A}$
- $O_{Pre\_UCON_A} = O_{RL-ABAM2} \cup O_{lock}$
- $S_{Pre\_UCON_A} = S_{RL-ABAM2}$
- $UR_{Pre\_UCON_A} = Command - R_{RL-ABAM2}$
- $Auxiliary - OS_\Delta = [lock{:}V(lock),\ type{:}V(type),\ R\_to\_select{:}V(R\_to\_select),\ position{:}V(position)]$
- $OS_\Delta = G - V_{\{RL-ABAM2\}} \cup Auxiliary - OS_\Delta$

The domain for each of these additional attributes is as follows: V(*lock*) = {0, 1}, V(*type*) = {ordinary, lock}, V(*R_to_select*) = $UR_{Pre\_UCON_A}$, and V(*position*) = {1,2}. The initial values for the proposed attributes are set as follows: For all $o \in O_{RL-ABAM2}$: o.type = ordinary, o.lock = 0, o.position = ϕ, and o.R_to_select = ϕ. For O_lock: O_lock.type = lock, O_lock.lock = 1, O_lock.position = ϕ, O_lock.R_to_select = ϕ.

To apply a RL-ABAM2 command in PreUCON$_\mathrm{A}$ commands, a sequence of steps is introduced as follows:

1- Give a lock to the first parameter of the RL-ABAM2 command
2- Decide the second parameter of the Rl-ABAM2 command
3- Implement a sequence of PreUCON$_\mathrm{A}$ commands
4- Release the lock from the first parameter (actor) of the RL-ABAM2 command

To implement the first step, a command called get_lock will be executed with the first parameter of RL-ABAM2 s_i and the special object O_lock:

```
**Command** $get\_lock\ (s_i : ATT(s_i), O\_lock : ATT(O\_lock))$
**if** $s_i.type = ordinary \land O\_lock.type = lock \land s_i.lock = 0 \land O\_lock.lock = 1$
**then**
    update attribute $s_i.lock = 1$;
    update attribute $O\_lock.lock = 0$;
    update attribute $s_i.position = 1$;
    update attribute $s_i.R\_to\_select = UR_{Pre\_UCON_A}$
**end**
```

Then, the actor needs to decide the second parameter, and the below command will take care of the second step:

```
**Command** $pick\_target(s_i : ATT(s_i), o_j : ATT(o_j))$
**if** $s_i.type = ordinary \land s_i.lock = 1 \land o_j.lock = ordinary \land s_i.position = 1 \land o_j.position = \phi$
**then**
    update attribute $o_j.position = 2$;
**end**
```

The third step contains an ordered series of PreUCON$_\mathrm{A}$ commands which depend on the number of the operation over rights in the body of an RL-ABAM2 command ($UR_{Pre\_UCON_A} = \{r_1, r_2, ..., r_k\}$). The structure of the ordered series of commands is as follows:

```
**Command**$-r_1(s_i, o_j)$
    **PreCondition:**   $f_b(s_i, o_j) \land$
                        $s_i.R\_to\_select = UR_{Pre\_UCON_A} \land s_i.lock = 1$
                        $\land\ s_i.position = 1 \land o_j.position = 2$;
    **PreUpdate:**
                        $create\ o$;
                        $s_i.a_k = v'_i$;
                        $o_j.a_s = v'_j$;
                        $s_i.R\_to\_select = UR_{Pre\_UCON_A} - \{r_1\}$
```

```
**Command**$-r_2(s_i, o_j)$
    **PreCondition:**   $s_i.R\_to\_select = UR_{Pre\_UCON_A} - \{r_1\} \land s_i.lock = 1$
                        $\land\ s_i.position = 1 \land o_j.position = 2$;
    **PreUpdate:**
                        $s_i.R\_to\_select = UR_{Pre\_UCON_A} - \{r_1, r_2\}$
```

```
                 ....
                 ....
Command−$r_k(s_i, o_j)$
    PreCondition:    $s_i.R\_to\_select = \{r_k\} \wedge s_i.lock = 1 \wedge s_i.position = 1$
                     $\wedge\ o_j.position = 2;$
    PreUpdate:
                     $o_j.position = \phi$
                     $s_i.R\_to\_select = \phi$
```

Finally, the user can release the lock and give it back to the special object O_lock by using the following command:

```
Command $release\_lock\ (s_i : ATT(s_i), O\_lock : ATT(O\_lock))$
if $s_i.type = ordinary \wedge O\_lock.type = lock \wedge s_i.lock = 1 \wedge O\_lock.lock = 0 \wedge$
$s_i.R\_to\_select = \phi$
then
update attribute $s_i.lock = 0$;
update attribute $O\_lock.lock = 1$;
update attribute $s_i.position = \phi$; end
```

5.2 An Example

The following example shows components of RL-ABAM2 schema. The command add-survey allows contributors to add a new health survey to their list. The contributors are required to be diabetics and never participated before. Moreover, the add-survey command permits contributors to post answers to questions and to close the survey after finishing. By the end of the survey, post and close rights will be taken away. The RL-ABAM2 schema is as follows: $(R_{RL-ABAM2}) = \{post, close\}$, $G-V_{\{RL-ABAM2\}} = [disease{:}\{diabetic, epileptic\}, X{:}\{0,1\}]$. Furthermore, RL-ABAM2 command will have the following structure:

```
Command $add - survey\ (s : ATT(s), o : ATT(o)$
if $s.disease = diabetic \wedge s.X = 0$
then
create object $o : ATT(o)$;
update attribute $o.disease = diabetes$;
update attribute $X = 1$;
enter $post$ into $[s, o]$;
delete $post$ from $[s, o]$;
enter $close$ into $[s, o]$;
delete $close$ from $[s, o]$; end
```

The corresponding PreUCON$_{\mathrm{A}}$ components of the RL-ABAM2 schema will be as follows:

- $O_{Pre\_UCON_A} = O_{RL-ABAM2} \cup O_{lock}$
- $S_{Pre\_UCON_A} = S_{RL-ABAM2}$

- $UR_{Pre\_UCON_A} = \{post, close\}$
- $Auxiliary{-}OS_{\Delta} = [lock{:}\{0,1\}, type{:}\{ordinary, lock\}, R\_to\_select{:}\{post, close\}, position{:}\{1,2\}]$
- OS_{Δ} = [disease:{diabetic, dpileptic}, X:{0,1}] $\cup\, Auxiliary - OS_{\Delta}$

V($lock$) = $\{0,1\}$, V($type$) = {ordinary, lock}, V(R_to_select) = $UR_{Pre\_UCON_A}$, and V($position$) = {1,2}.

The initial values for the Auxiliary attributes are set as above, and to apply a RL-ABAM2 command in PreUCON$_{\mathrm{A}}$ commands, the four sequence steps will be implemented as follows:

1- Give a lock to the first parameter of the RL-ABAM2 command by using the *get_lock* command.
2- Decide the second parameter of the Rl-ABAM2 command by using the *pick_target* command.
3- Implement a sequence of PreUCON$_{\mathrm{A}}$ commands as follows:

Command$_{post}(s, o)$

PreCondition: $s.disease = disease \wedge s.X = 0 \wedge$
$s.R\_to\_select = \{post, close\} \wedge s.lock = 1 \wedge$
$s.position = 1 \wedge o.position = 2;$

PreUpdate:

$create\ o;$
$o.disease = diabetes;$
$s.X = 1;$
$s_.R\_to\_select = \{close\};$

Command$_{close}(s, o)$

PreCondition: $s.R\_to\_select = \{close\} \wedge s.lock = 1 \wedge$
$s.position = 1 \wedge o.position = 2;$

PreUpdate:

$o.position = \phi$
$s.R\_to\_select = \phi$

4- Release the lock from the first parameter (actor) of the RL-ABAM2 command by using *release_lock* command.

6 CONCLUSION

In this paper we have formally demonstrated the equivalence of PreUCON$_{\mathrm{A}}$ and RL-ABAM2, which are two finite domain ABAC models. We have argued that ABAM being a superset of HRU cannot be reduced to PreUCON$_{\mathrm{A}}$, because of the latter's decidable safety result. Hence, equivalence of PreUCON$_{\mathrm{A}}$ can only be established to some proper sub-model of ABAM such as RL-ABAM2. Our constructions suggest the power of using formulas in PreUCON$_{\mathrm{A}}$, absence of which in ABAM leads to having to an explosion of ABAM commands in the PreUCON$_{\mathrm{A}}$ to ABAM reduction. Conversely, the ability to activate multiple rights in a single

RL-ABAM2 command leads to multiple PreUCON$_A$ commands in the ABAM to PreUCON$_A$ reduction. These features could be combined in a more usable model. Finally, the study of ABAM indicates that a safe application of access rights could be based on the following principles. Firstly, do not use rights in the if part of commands. Secondly, some rights could be left behind by commands so their next use is more efficient. Our comparative study of PreUCON$_A$ and ABAM suggests there is a meaningful place for access matrix rights, even as access control research and practice is tending towards attributes.

Acknowledgement

This research is partially supported by NSF Grant CNS-1111925 and CNS-1423481.

References

1. Al-Kahtani, M.A., Sandhu, R.: Rule-based RBAC with negative authorization. In: 20th IEEE ACSAC, pp. 405-415, (2004).
2. Bennett, P., Ray, I. and France, R.: Modeling of Online Social Network Policies Using an Attribute-Based Access Control Framework. In: Information Systems Security, pp.79-97, Springer International Publishing (2015).
3. Chadwick, D.W., Otenko, A. and Ball, E.: Role-based access control with X. 509 attribute certificates. In: IEEE Internet Computing ,7(2), pp.62-69, (2003).
4. Harrison, M.A., Ruzzo, W.L. and Ullman, J.D.:Protection in operating systems. Communications of the ACM, 19(8), pp. 461-471 (1976).
5. Hu, V.C., Ferraiolo, D., Kuhn, R., Friedman, A.R., Lang, A.J, Cogdell, M.M, Schnitzer, A., Sandlin, K., Miller, R., Scarfone, K.: Guide to attribute based access control (ABAC) definition and considerations. NIST Special Publication 800, 162 (2014).
6. Hu, V.C., Kuhn, D.R., Ferraiolo, D.F.: Attribute-based access control. IEEE Computer, 48(2), pp. 85-88 (2015)
7. Jin, X., Krishnan, R., Sandhu, R.S.: A unified attribute-based access control model covering DAC, MAC and RBAC. In: DBSec, pp. 41-55 (2012).
8. Kuhn, D.R., Coyne, E.J., Weil, T.R.: Adding attributes to role-based access control. IEEE Computer, 43(6), pp. 79-81, (2010).
9. Park, J., Sandhu, R.: The UCON ABC usage control model. ACM Transactions on Information and System Security (TISSEC), 7(8), pp. 128-174, (2004).
10. PV, Rajkumar and Sandhu, R.: Safety Decidability for Pre-Authorization Usage Control with Finite Attribute Domains. In: IEEE Transactions on Dependable and Secure Computing, (2015).
11. Sandhu, R.S.: The typed access matrix model.In: Research in Security and Privacy, 1992, pp. 122-136, (1992)
12. Tripunitara, Mahesh V and Li, Ninghui. A theory for comparing the expressive power of access control models. Journal of Computer Security, 15(2):pp. 231-272, (2007).
13. Yong, J., Bertino, E., Roberts, M.T.D.: Extended RBAC with role attributes. In: PACIS 2006 Proceedings, paper 8, (2006).
14. Zhang, X., Li, Y., Nalla, D.: An attribute-based access matrix model. In: the 2005 ACM Symposium on Applied Computing, pp. 359-363 (2005).