# Safety Decidability for Pre-Authorization Usage Control with Finite Attribute Domains

P.V. Rajkumar and Ravi Sandhu

**Abstract—**This paper considers the safety problem for the pre-authorization sub-model of the well-known $UCON_{ABC}$ usage control model, that is, $Pre\_UCON_A$. It is shown that $Pre\_UCON_A$ with finite attribute domains has decidable safety even if arbitrary object creation is allowed. This result eliminates the previously known restrictions for obtaining safety decidability in this context, which only allow a finite bounded number of objects to be created. Our result specifically permits unbounded object creation, so the set of objects is potentially infinite. In the proof, we show that the set of reachable protection tuples in infinite state $Pre\_UCON_A$ models is finite and computable. We also provide a construction for decision procedure which answers the safety question by examining the reachable protection tuples.

**Index Terms—**Usage control authorization, safety, decidability, automated analysis and decision procedures

## 1 Introduction

$UCON_{ABC}$ [1] is the well-known usage control model developed to support diverse security requirements of modern information systems in a unified manner. The notion of subject and object attributes is central to $UCON_{ABC}$. $Pre\_UCON_A$ is an authorization sub-model of $UCON_{ABC}$ which has features to support classical mandatory access control (MAC), discretionary access control (DAC) and role-based access control (RBAC) policies, as well as consumable and regenerative usage rights, by means of attributes. $Pre\_UCON_A$ includes mutable attributes which are updated as a consequence of usage actions performed by subjects on objects.

A fundamental issue in any access or usage control model is to determine whether the given policies and the initial configuration actually enforce the *security intent* of the security architects. In $Pre\_UCON_A$, usage decisions are based on the attribute values of the subject and the object. Execution of a usage right can modify the attribute values of the involved subject and object, or possibly create a new object which in turn brings in new values for its attributes. Existence of the new object or modified attributes of an existing object can enable new permissions. Execution of the new permissions can further bring in additional new objects and modify existing attribute values. This cycle could continue indefinitely. Given these interactions, it is difficult for a security architect to manually check if the policy and the initial configuration could lead to any *unintended* security violations.

In this work, we present an automated decision procedure to check such unintended policy violations when the attributes are restricted to be from constant finite domains. Notably, this procedure is applicable even in presence of arbitrary object creation. This result eliminates the previously known restrictions [2] for such decidability, which amount to allowing only a finite bounded number of objects to be created (see proof of Theorem 3 in [2]). Our result allows the set of objects to grow without a finite bound.

One of the basic policy violations is unauthorized usage right leakage, that is, a subject which is not supposed to get a right over an object gets the right. Analysis of such leakages is often referred as safety analysis and an automated decision procedure which does that analysis is called a safety decision procedure (SDP). The usage control authorization model has been shown to simulate a single tape Turing machine whereby the Turing machine's halting problem is reduced to the safety analysis problem in $Pre\_UCON_A$ [2]. Hence, it is not possible to have an SDP for $Pre\_UCON_A$, in general. We emphasize that $Pre\_UCON_A$ permits unbounded attribute domains such as the set of objects in the system. Further, it has been shown that $Pre\_UCON_A$ with infinite domain attributes (such as natural numbers or strings over a finite alphabet) can encode the Post Correspondence Problem, even without any object creation [3].

In many practical systems security attributes, such as social-security number, roles, labels, gender, title, department, and so on, have constant finite domains, while the number of resources and processes within a system could be unbounded. This paper focusses on a finite attribute domain security system with infinite number of objects due to object creation. Restricted sub-models of usage control authorization with finite attribute values and with acyclic object creations have previously been shown to have SDPs [2].

In this paper, we prove that usage control authorization with finite attribute value domains is *sufficient* to ensure existence of SDPs. This is without any restrictions on number of object creations and the attribute update features of the scheme. This is a significant generalization of previous results in this arena and requires a novel construction that has not previously been applied to the

• *The authors are with the Institute for Cyber Security, University of Texas at San Antonio, San Antonio, TX 78249.*
*E-mail: rajkumarpv@gmail.com, ravi.sandhu@utsa.edu.*

*Manuscript received 5 Aug. 2014; revised 6 Apr. 2015; accepted 10 Apr. 2015. Date of publication 29 Apr. 2015; date of current version 2 Sept. 2016.*
*For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.*
*Digital Object Identifier no. 10.1109/TDSC.2015.2427834*

safety problem. Our construction is based on the notion of *protection tuples (PT)* and their *computability*. A protection tuple is a combination of attribute values that can enable usage rights. We prove that the *maximal set of reachable protection tuples* is finite in a usage schema with finite attribute domains, and we construct a decision procedure that finds the reachable set of protection tuples and answers the safety question.

The rest of this paper is organized as follows. Section 2 presents related work. Section 3 presents the usage control authorization scheme analyzed in this paper. Section 4 introduces the notion of protection tuples. Section 5 brings in the concept of reachable protection tuples, shows that this set is computable and gives a decision procedure for the safety problem based on this notion. The complexity of this problem turns out to be PSpace-Hard. Section 6 concludes the paper.

## 2 Related Work

Usage control is an unification approach to combine traditional access control with obligations, conditions, and attribute mutability. $UCON_{ABC}$ [1] is the first comprehensive model to support all these features under one umbrella. This model defines three decision components namely *Authorization*, *oBligation* and *Condition* (ABC) to support security policies involving authorization, obligation, and condition requirements, respectively.

Safety analysis of $Pre\_UCON_A$, an authorization fragment of $UCON_{ABC}$, has been shown to be an undecidable problem in general [2]. However, it has been shown that the safety is decidable for two restricted cases of $Pre\_UCON_A$ [2]. The first sub-model has finite attribute value domains without object creation. The second sub-model has finite attribute value domains with restricted object creation which imposes a finite bound on the number of objects that can be created in the system. Our work improves on the decidability result obtained in [2] by relaxing the finite object creation constraint without applying any additional restrictions.

Safety analysis of ongoing authorization model $On\_UCON_A$ with finite attributes and without object creation policies has been studied in [4]. Alternatively, safety analysis problem can be mapped to a satisfiability problem in some logical theories, thereby decidability results of logical theories are immediately available for safety analysis. This approach has been followed in [5], where in particular, decidability results of many-sorted first order logic have been applied to solve the usage control safety problem. The safety decidable fragment identified in [5] has finite number of objects and the objects' attributes can take values from infinite domains, but the update actions do not include arithmetic operations like addition and multiplication.

The safety analysis problem dates back to 1970's access control models. Safety of access control matrix (ACM) model was shown to be undecidable [6], in general. Monotonic mono-conditional ACM models have SDPs. The schematic protection model (SPM) [7], [8] introduced typed security entities. Each entity is associated with a security type which remains unchanged. The extended SPM with conditional tickets and revocation features are explored in [9], [10]. The typed access matrix (TAM) [11] is an extension of access control matrix model by adding types to the objects. Both the safety decidable TAM model and the SPM model support only a limited set of monotonic access control policies. The Dynamic-Typed Access Matrix (DTAM) [12] model introduces the constructs needed to change the object types in the TAM model to support non-monotonic access control policies. The DTAM model with limited object creations has SDPs.

The main idea behind the RBAC model [13] is to specify and enforce enterprise-specific security policies in a natural way as they present themselves in an organizational structure. The RBAC model provides a simplified approach to manage users and resources by decoupling the permissions and users. This model has been extended to support temporal [14] and numerous other features.

Adminstration of RBAC and its safety problem are explored in [14], [15], [16], [17], [18], [19], [20], [21], [22]. Security analysis of RBAC has been elaborately addressed in [22]. Recently, some of the program analysis techniques have been used to study the ARBAC safety problem. Abstraction refinement approach is applied for ARBAC security analysis in [23], [24] and the technique is implemented in a bounded model checker called Mohawk [24]. A fragment of first order logic along with the satisfiability modulo theories have been applied in ARBAC security analysis [25]. The approach presented in [26] translates the ARBAC policies into imperative style programs and employs an inter-procedural analyzer for security verification.

Usage control is fundamentally a different model than the ARBAC models both in terms of its design and application. The ARBAC models are exclusively meant to administer the user-role assignment, role-permission assignment, and role-role assignment components of the RBAC model. Whereas the usage control model has been designed to cater to much broader range of applications than the RBAC model itself. Usage control has rich features like attribute mutability, consumable rights and object creation which are not part of the RBAC design.

The present work significantly improves on previous usage control safety analysis results. We prove the existence of an SDP for $Pre\_UCON_A$ with finite attribute domains. We believe that the SDP can be extended to many other models of practical interest, so long as the attribute domains are finite. Detailed consideration of such extensions is beyond the scope of this paper.

## 3 Usage Control Authorization Model

The usage control authorization model follows the standard convention of representing user processes as *subjects* and resources as *objects*. Subjects are considered as a subset of objects in this model. Each object has a unique name and a finite set of attributes. An attribute's value can be accessed using the *dot* operator, as in $object\_name.attribute\_name$, e.g., $o_i.color = 'red'$. The model supports both mutable and immutable attributes, as well as dynamic creation and deletion of objects. A usage right is a permission defined over an object. We use the terms right and permission interchangeably.

This section introduces the model features necessary for safety analysis. Readers are referred to [1] for understanding the expressivity and applications of this model.

### 3.1 Pre-Authorization Model

The pre-authorization model has only pre-enforcement commands. In such commands, the decision module grants permission at the beginning of usage right execution and the permission remains enabled till the end of the execution. The model allows attribute updates before the usage of the permission, called a *preUpdate*. Recall, that mutable attributes are updated as a consequence of usage actions performed by a subject on an object. Both subject and object attributes can be updated as per the command specification. There can also be *create* and *delete* operations in a pre-enforcement command, in addition to attribute updates.

### 3.2 Usage Control Scheme

**Definition 3.1.** *A usage control scheme $U_\Theta$ has three components as follows:*

*(i) an object schema $OS_\Delta$,*
*(ii) a set of usage rights $UR = \{r_1, r_2, \ldots, r_m\}$, and*
*(iii) a set of usage control commands $\{UC_1, UC_2, \ldots, UC_n\}$.*

#### 3.2.1 Object Schema and Rights

The object schema specifies the attributes of objects and the domains from which attribute values are drawn.

**Definition 3.2.** *The object schema $OS_\Delta$ is of the form $[a_1 : \Omega_1, a_2 : \Omega_2, \ldots, a_n : \Omega_n]$, where each $a_i$ is the name of an attribute and $\Omega_i$ is a finite set specifying $a_i$'s domain.*

Note that the set of subjects or objects does not qualify to be an attribute domain $\Omega_i$, because in general these sets are unbounded due to creation.

For simplicity we assume there is a single object schema which applies uniformly to all objects. In practice different objects usually will have different attributes. For example, a user could have a role attribute or a gender attribute while a file may have attributes such as a sensitivity label. Generalization of our safety results to systems with multiple object schemas for different kinds of objects is straightforward as shown in [2]. We assume each attribute value is atomic being a single element from the attribute's domain. Set-valued attributes, such as role, are common in practice but can be reduced to atomic attributes for our purpose since all attribute domains are constant and finite. For example, a set-valued attribute with domain $\{\phi, \{a\}, \{b\}, \{a, b\}\}$ can be recast as an atomic-valued attribute with domain $\{set\_empty, set\_a, set\_b, set\_ab\}$, where each of these values is understood to denote the corresponding set to simulate the set-valued attribute. Optionally, attributes can have a default value from the domain at creation time which does not need to be explicitly assigned. Attributes with default values are specifically identified with bold case italicized letters within the object schema.

The second component of a usage control scheme is a finite set of usage rights $UR$ which defines the permission $r_i$ that can be enabled by a usage control command. The interpretation of an individual right $r_i$ is not specified in the model. The usage of a right by a subject on an object is contingent on their attribute values as determined by usage control commands as discussed below.

#### 3.2.2 Usage Control Commands

The third component of a usage control scheme is a finite set of usage control commands. Each command has a name and is associated with a right $r$ that it authorizes, written as a subscript to the name. Each command has two formal parameters $s$ and $o$, of which $s$ is the subject attempting to access $o$ with permission $r$.

There are two kinds of usage control commands: non-creating commands in which the object $o$ exists prior to execution of the command and creating commands in which the object $o$ is created during execution of the command.

#### 3.2.3 Non-Creating Commands

A non-creating command has the following structure.

$\mathbf{Command\_Name}_r(s, o)$
**PreCondition:** $f_b(s, o) \rightarrow \{yes, no\}$;
**PreUpdate:** $s.a_{i_1} := f_{1,a_{i_1}}(s, o)$;
$\ldots$
$s.a_{i_p} := f_{1,a_{i_p}}(s, o)$;
$o.a_{j_1} := f_{2,a_{j_1}}(s, o)$;
$\ldots$
$o.a_{j_q} := f_{2,a_{j_q}}(s, o)$;

Here $f_b(s, o)$ is a Boolean function which takes the attribute values of $s$ and $o$ as input and evaluates to $yes$ or $no$. If the result is $yes$ then the PreUpdate is performed and the usage right $r$ is granted. Otherwise, the command terminates without granting $r$. In the PreUpdate part, zero or more attributes of $s$ and $o$ are independently updated to new values computed from their attribute values prior to the command execution. In general, it is possible for $s$ to equal $o$ whereby a subject can perform an operation on itself.

#### 3.2.4 Creating Commands

The structure of a creating command is as follows:

$\mathbf{Command\_Name}_r(s, o)$
**PreCondition:** $f_b(s) \rightarrow \{yes, no\}$;
**PreUpdate:** $create\ o$;
$s.a_{i_1} := f_{1,a_{i_1}}(s)$;
$\ldots$
$s.a_{i_p} := f_{1,a_{i_p}}(s)$;
$o.a_1 := f_{2,a_1}(s)$;
$\ldots$
$o.a_n := f_{2,a_n}(s)$;

The $create\ o$ operation brings a new object into the usage control system. The object identifier $o$ is considered as a meta value, in that each execution of create operation in the system assigns a unique object identifier which is never re-used. Here $f_b(s)$ is a Boolean function which takes the attribute values of $s$ and evaluates to $yes$ or $no$. If the result is $yes$ then the PreUpdate is performed and the usage right $r$ is granted. Otherwise, the command terminates without creating $o$ or granting $r$. In the PreUpdate part, zero or more attributes of $s$ are updated to new values computed from $s$'s attribute values prior to the command execution. Moreover, all attributes of the newly created object $o$ are assigned similarly computed attribute values. Some of the assignments to $o$'s attributes may be omitted for attributes that have a default value defined, in which case the default value will be assigned.

For both commands we assume that the entire command executes as an atomic action. We further assume that each attribute can be updated by at most one assignment statement in the PreUpdate part. Moreover, reference to an attribute of $s$ and $o$ on the right hand side of an assignment is understood to mean the value of the attribute prior to the command being executed. Thereby the ordering of the update assignments is not material.

The $Pre\_UCON_A$ model also includes a *delete* operation. For purpose of safety analysis we can ignore the delete operation. Intuitively delete can be simulated by means of a special boolean attribute, call it *deleted*, so that $o.deleted = yes$ indicates that object $o$ has been deleted while $o.deleted = no$ indicates the object is existing. Additional details for this argument are provided in the appendix.

### 3.3 Example Usage Control Schema

The following toy game example illustrates components of usage control schema. The example schema permits (i) players to *mark* a white ball into a red ball, (ii) players to *hit* a ball after he marks three white balls into red balls, and (iii) existing players can create new players in the game.

**Example 3.3.** The components of the usage control schema $U_{\Theta game}$ are given as follows.

1) Object schema $OS_{\Delta game}$ = [*x*:{*0,1,2,3*}, *color*:{*red, white, blue*}]
2) Usage rights UR = {*mark, hit, addplayer*}
3) Usage commands UC = {$UC_1$, $UC_2$, $UC_3$} where

$UC_1$: $\mathbf{Permit}_{mark}(s, o)$
**PreCondition:** $s.color = blue \wedge o.color = white$;
**PreUpdate:** $o.color := red$;
$s.x = s.x + 1$;

$UC_2$: $\mathbf{Permit}_{hit}(s, o)$
**PreCondition:** $s.color = blue \wedge s.x = 3 \wedge o.color \neq blue$;

$UC_3$ : $\mathbf{Permit}_{addplayer}(s, o)$
**PreCondition:** $s.color = blue$;
**PreUpdate:** $createo$;
$o.x = 0$;
$o.color = blue$;

The object schema $OS_{\Delta game}$ has two attributes namely '*x*'and '*color*'. The attribute *x* can take a value from the finite set {*0, 1, 2, 3*} and the attribute *color* can take a value from the set {*blue*, *red*, *white*}. In this scheme, color attribute of the players are always *blue* and the attribute *x* of the balls are always *0*. The usage rights *mark*, *hit*, and *addplayer* are protected by the usage commands $UC_1$, $UC_2$ and $UC_3$ respectively.

We use example 3.3 as a running example to illustrate the concepts and definitions introduced in the remaining parts of this paper.

### 3.4 Usage Control Configuration

The state of an usage control authorization system is called a usage configuration, defined as follows.

**Definition 3.4.** *A usage configuration $U_\Xi$ is defined as the set of objects $\{o_1, o_2, \ldots, o_n\}$ in a state along with the values of the attributes for each object $o_i$.*

**Example 3.5.** An example usage configuration for the usage schema $U_{\Theta game}$ of example 3.3 with one user and four balls is given as follows:

$$\begin{aligned} U_\Xi = \{ & s.x = 0, s.color = blue, \\ & o_1.x = 0, o_1.color = white, \\ & o_2.x = 0, o_2.color = white, \\ & o_3.x = 0, o_3.color = white, \\ & o_4.x = 0, o_4.color = white\} \end{aligned}$$

Where appropriate the state can be named as in $U_{\Xi init}$ for the initial state and in $U_{\Xi k}$ for state $k$. The set of all possible usage configurations for a usage control scheme $U_\Theta$ is denoted by $\widehat{U_{\Xi\Theta}}$. Since the number of objects is unbounded $\widehat{U_{\Xi\Theta}}$ is, in general, a countably infinite set.

A usage control system begins in its specified initial state and evolves thereafter by execution of usage control commands at the discretion of the subjects in the system. Usage commands are referred to as $UC_i$ and are specified with formal parameters. A usage command instance for usage command $UC_i$ is written as $uc_i$ and will be executed with actual parameters substituted for formal parameters.

### 3.5 Safety Question

A safety question is written as $\Upsilon_{(s_1,o_1,r)}$. For a given usage control scheme $U_\Theta$ and an initial configuration $U_{\Xi init}$, the safety question asks if it is possible for subject $s_1 \in U_{\Xi init}$ to gain usage right $r$ for object $o_1 \in U_{\Xi init}$. A safety decision procedure takes as input $U_\Theta$, $U_{\Xi init}$ and $\Upsilon_{(s_1,o_1,r)}$. It returns *yes* if subject $s_1$ can acquire usage right $r$ for $o_1$ and *no* otherwise.

In order to make the technical presentation easier, we analyse a different safety question denoted $\Upsilon_{(r)}$, defined as follows. For a given usage control scheme $U_\Theta$ and an initial configuration $U_{\Xi init}$, is it possible for *any* subject $s$ to gain usage right $r$ for *any* object $o$? An SDP which can answer $\Upsilon_{(r)}$ can be used to find the answer for the original safety question $\Upsilon_{(s_1,o_1,r)}$ by applying it to augmented versions of the schema $U'_\Theta$ and initial configuration $U'_{\Xi init}$, defined below. Let $UC^{[r]}$ be the set of usage commands in $U_\Theta$ which grant the usage right $r$ in the original safety question $\Upsilon_{(s_1,o_1,r)}$.

- The augmented usage schema $U'_\Theta$ has
  - Object schema $OS'_\Delta = [a_1 : \Omega_1, a_2 : \Omega_2, \ldots, a_n : \Omega_n, color : \{red, \mathbf{white}\}]$, where each $a_i : \Omega_i \in OS_\Delta$ and *color* is a new attribute, different from $a_1, a_2, \ldots, a_n$, with the default value *white*.
  - Set of usage rights $UR' = \{r_1, r_2, \ldots, r_n, r'\}$, where $r_i \in UR$ and the $r'$ is a new usage right, different from $r_1, r_2, \ldots, r_n$, introduced in the augmented schema.
  - Set of usage commands $UC' = UC \cup UC_{aug}$. The new commands in $UC_{aug}$ each grant the new usage right $r'$. Each command in $UC^{[r]}$ has an augmented version in $UC_{aug}$. The **PreCondition** Boolean function in each usage command in $UC_{aug}$ is identical to that of its counterpart in $UC^{[r]}$ with the additional check $s.color = red \wedge o.color = red$. The **PreUpdate** part of the commands in $UC_{aug}$ is empty.

TABLE 1
Summary of Symbols

| Symbols | Description |
|---|---|
| $U_\Theta$ | The usage schema, does not change |
| $U_\Xi$ | A usage configuration for given $U_\Theta$ |
| $U_{\Xi init}$ | An initial configuration for given $U_\Theta$ |
| $U_{\Xi k}$ | Usage configuration in state $k$ for given $U_{\Xi init}$ |
| $\widehat{U_{\Xi\Theta}}$ | The set of all usage configurations for given $U_\Theta$ |
| $UC_i$ | Usage control command |
| $uc_i$ | Usage control command instance |
| $\Upsilon_{(s_1,o_1,r)}$ | Safety question: can $s_1 \in U_{\Xi init}$ gain $r$ for $o_1 \in U_{\Xi init}$? |
| $\Upsilon_{(r)}$ | Safety question: can any subject gain $r$ for any object? |
| $AVT$ | Attribute value tuple, $AVT = <v_1, v_2, \ldots, v_n>$ |
| $AVT_\phi$ | Empty $AVT = <\phi, \phi, \ldots, \phi>$, $\phi \notin \Omega_i$ |
| $AVT_{\Xi i}(o)$ | $<o.a_1, o.a_2, \ldots, o.a_n>$ |
| $PT$ | $PT = <AVT_1, AVT_2>$, $AVT_1 \neq AVT_\phi$ |
| $\widehat{PT_\Theta}$ | Set of all possible $PT$s for given usage schema $U_\Theta$ |
| $MPT$ | Multiset of $PT$s in an $U_\Xi$ |
| $[\![MPT]\!]$ | Determinant, number of distinct $PT$s |
| $[\![[\![MPT]\!]]\!]$ | Absolute determinant, total number of $PT$s |
| $\widehat{MPT}$ | Set of all possible MPTs for given $U_\Theta$ |
| $\eta_{max}$ | $\eta_{max} =$ maximum $MPT = \lvert\widehat{PT_\Theta}\rvert$ for given $U_\Theta$ |
| $f_\Xi$ | $f_\Xi : \widehat{U_{\Xi\Theta}} \rightarrow \widehat{MPT}$ |
| $PT \in \widehat{U_\Xi}$ | $(\exists U_{\Xi p} \in \widehat{U_\Xi}) PT \in f_\Xi(U_{\Xi p})$ |
| $\widehat{U_{\Xi i}} \approx_{PT} \widehat{U_{\Xi j}}$ | $(\forall PT \in PT_\Theta) PT \in \widehat{U_{\Xi i}} \Leftrightarrow PT \in \widehat{U_{\Xi j}}$ |
| $\widehat{U_{\Xi init\uparrow 0}}$ | $\{U_{\Xi init}\}$ |
| $\widehat{U_{\Xi init\uparrow 1}}$ | Set of usage configurations reachable from $U_{\Xi init}$ in 0 or 1 step |
| $\widehat{U_{\Xi init\uparrow k}}$ | Set of usage configurations reachable from $U_{\Xi init}$ in 0 or up to k steps |
| $\widehat{U_{\Xi i\uparrow *}}$ | Set of usage configurations reachable from $U_{\Xi i}$ in 0 or any number of steps |

- The initial configuration $U'_{\Xi init}$ is identical to $U_{\Xi init}$ except that $s_1.color := red$, $o_1.color := red$, and $s_i.color := white$, $o_i.color := white$, for $i \neq 1$.

The augmented usage schema $U'_\Theta$ does not change any of the usage commands in $U_\Theta$. The additional attribute *color* has not been used anywhere except in the new usage command in $UC_{aug}$ which do not make any state changes. Moreover, $s_1$ and $o_1$ are the only objects whose *color* attribute can be *red*. Hence, the result of $\Upsilon_{(r')}$ on $U'_\Theta$ and $U_{\Xi' init}$ would be same as the result of $\Upsilon_{(s_1,o_1,r)}$ on $U_\Theta$ and $U_{\Xi init}$.

## 4 Protection Tuples

This section introduces the notion of protection tuples and a method to compactly represent them using a multiset notation. The protection tuples basically are attribute values of the subjects and objects in a usage configuration. Since the system allows unbounded creation of objects, the size of the protection tuple set can be unbounded. However, if we ignore multiple occurrences then the number of distinct protection tuples is bounded by the maximum value $\eta_{max}$. We also define a function $f_\Xi$ which maps a usage configuration to the compact representation of protection tuples.

For convenience symbols and notation used in this paper are summarized in Table 1, grouped into blocks by horizontal lines. The symbols in the first three blocks were introduced in the previous section, while the fourth block summarizes this section. Symbols in the fifth block will be defined in the next section.

For ease of exposition it is convenient to assume that the attributes are ordered $a_1, a_2, \ldots, a_n$ so that the attribute values for an object can be represented as a similarly ordered n-tuple as follows.

**Definition 4.1.** *An attribute value tuple (AVT) is an ordered n-tuple $<v_1, v_2, \ldots, v_n>$, where n is the number of attributes in the object schema and each $v_i \in \Omega_i$.*

Each object $o \in U_{\Xi t}$ has an $AVT$, denoted $AVT_{\Xi t}(o) = < v_{t_1}, v_{t_2}, \ldots, v_{t_n} >$ where $o.a_i = v_{t_i} \in U_{\Xi t}$, which can change in different states. For convenience, we write $AVT_{\Xi t}(o) = <o.a_1, o.a_2, \ldots, o.a_n>$. We define the notion of an empty $AVT$ as follows.

**Definition 4.2.** *An empty attribute value tuple, denoted $AVT_\phi$, is a special n-tuple $AVT_\phi = <\phi, \phi, ..., \phi>$, where $\phi$ is a special symbol which does not belong to any $\Omega_i$.*

$AVT_\phi$ denotes that the attribute values of an object being created are not yet ready to be used in usage command preconditions. Since each usage command involves two formal parameters we define the following notion to refer to a pair of $AVT$s.

**Definition 4.3.** *A protection tuple is defined as an ordered pair of AVTs, $PT = <AVT_1, AVT_2>$ where $AVT_2$ can be $AVT_\phi$ but $AVT_1$ cannot.*

Every ordered pair of objects in a usage configuration has an associated $PT$ with $AVT_1 \neq AVT_\phi$ and $AVT_2 \neq AVT_\phi$. This $PT$ accounts for possible non-creating command instances involving this ordered pair of objects. Additionally every

object has an associated $PT$ with $AVT_1 \neq AVT_\phi$ and $AVT_2 = AVT_\phi$, to account for possible creating commands. We write $\widehat{PT_\Theta}$ to denote the set of all possible $PT$s for a given usage schema $U_\Theta$.

For a given usage configuration, in general there will be multiple object pairs with identical $PT$s, as well as multiple individual objects with identical $PT$s with the second component being $AVT_\phi$. This leads us to introduce the following notion.

**Definition 4.4.** *A collection of protection tuples constitutes a multiset $MPT = \{PT_1^{i_1}, PT_2^{i_2}, ..., PT_\eta^{i_\eta}\}$, where each $PT$ is a protection tuple and $\forall x \forall y((PT_x = PT_y) \Rightarrow (x = y))$. The superscript $i_x$ denotes the number of distinct occurrences of $PT_x, 1 \leq x \leq \eta$ in the usage configuration.*

For example, if a collection of $PT$s has a total of $t_1$ $PT$s and among them a subset of $t_2 \leq t_1$ $PT$s are different from each other then that collection constitutes the $MPT = \{PT_1^{i_1}, PT_2^{i_2}, ..., PT_\eta^{i_\eta}\}$, where $\eta = t_2$ and $\Sigma_{x=1}^{\eta} i_x = t_1$. Accordingly, we define the determinant ($[\![]\!]$) of an MPT as the number of distinct $PT$s in the multiset, that is $[\![MPT]\!] = \eta$, and the absolute determinant ($[\![[\![]\!]]\!]$) as $[\![[\![MPT]\!]]\!] = \sum_{PT_x^{i_x} \in MPT} i_x$. We use the notation $PT \in MPT$ and $PT^i \in MPT$ more or less interchangeably depending on whether or not the multiplicity of the $PT$ is relevant or not in a particular context.

In this paper we are primarily concerned with $MPT$s that can occur for a given usage schema $U_\Theta$. Since the usage schema is fixed when the system is created and does not change, we usually omit explicit mention of the schema. The set of all possible MPTs for a given usage schema $U_\Theta$ is denoted by $\widehat{MPT}$. $\widehat{MPT}$ is countably infinite in general, due to unbounded multiplicity of individual $PT$s for the given schema. Correspondingly, the maximum absolute determinant of an $MPT \in \widehat{MPT}$, i.e. maximum $[\![[\![MPT]\!]]\!]$, is unbounded. However, the maximum determinant of an $MPT \in \widehat{MPT}$, i.e. maximum $[\![MPT]\!]$, is finite due to the assumption of finite attribute value domains which limits the number of distinct $PT$s that can occur. This finite bound, denoted $\eta_{max}$, is critical to the results of this paper. We have the following observation.

**Lemma 4.5.** *Let $\eta_{max}$ denote the maximum determinant of an MPT for a given usage schema $U_\Theta$. $\eta_{max} = (\Pi_{i=1}^{n}(|\Omega_i|))^2 + (\Pi_{i=1}^{n}(|\Omega_i|))$, where $|\Omega_i|$ is the domain size of attribute $a_i$ and $n$ is the number of attributes defined in the object schema $OS_\Delta$ of $U_\Theta$.*

**Proof.** The maximum number of distinct AVTs for a given $U_\Theta$ is $\Pi_{i=1}^{n}(|\Omega_i|)$. The maximum number of distinct $PT$s with $AVT_2 \neq AVT_\phi$ is $(\Pi_{i=1}^{n}(|\Omega_i|))^2$. The maximum number of distinct $PT$s with $AVT_2 = AVT_\phi$ is $\Pi_{i=1}^{n}(|\Omega_i|)$. Since construction of the initial configuration is arbitrary, an $MPT$ with these maximum numbers can be instantiated in an initial configuration. □

For instance, for the usage schema of example 3.3 we have $\eta_{max} = (4 * 3)^2 + (4 * 3) = 156$. Note that $\eta_{max}$ can be equivalently defined to be $|\widehat{PT_\Theta}|$.

It remains to formally define the relationship between a usage configuration $U_\Xi$ and its $MPT$. The notion of a multiset sum is useful for this purpose.

**Definition 4.6.** *The multiset sum $X \uplus Y$ of two multisets $X$ and $Y$ is defined as the multiset $Z$ for which each member $m \in Z$ has the sum of the multiplicities $m$ has in $X$ and $Y$ (where possibly one, but not both, of these could be zero).*

This definition naturally extends to more than two multisets, and is commutative and associative.

The relationship between a usage configuration $U_\Xi$ and its $MPT$ is defined by the tuple aggregation function $f_\Xi : \widehat{U_{\Xi\Theta}} \rightarrow \widehat{MPT}$ as follows.

**Definition 4.7.** *Let $AVT(o_i)$ be the AVT of object $o_i$ in a given configuration $U_\Xi$. Then $f_\Xi(U_\Xi)$ is the MPT defined as follows:*

$$\begin{aligned} f_\Xi(U_\Xi) = \uplus_{o_i, o_j \in U_\Xi} \{ & \{< AVT(o_i), AVT(o_j) >\} \\ & \uplus \{< AVT(o_j), AVT(o_i) >\} \\ & \uplus \{< AVT(o_i), AVT(o_i) >\} \\ & \uplus \{< AVT(o_j), AVT(o_j) >\}\} \\ & \uplus \\ & \uplus_{o_i \in U_\Xi} \{\{< AVT(o_i), AVT_\phi >\}\}. \end{aligned}$$

The intuition behind this definition is straightforward. The $MPT = f_\Xi(U_\Xi)$ is the multiset sum of $PT$s generated by all pairs of objects in usage configuration $U_\Xi$, and $PT$s with $AVT_2 = AVT_\phi$ generated by all objects in $U_\Xi$.

**Example 4.8.** The following multiset represents the *MPT* of the usage configuration $U_{\Xi i}$ given in example 3.5.

$$\begin{aligned} f_\Xi(U_{\Xi i}) = \{ & < [0, blue], \phi >^1, < [0, white], \phi >^4, \\ & < [0, blue], [0, blue] >^1, < [0, white], [0, blue] >^4, \\ & < [0, blue], [0, white] >^4, < [0, white], [0, white] >^{16}\}. \end{aligned}$$

## 5 Reachable Protection Tuples

In this section, we prove that the set of reachable protection tuples for a given usage schema $U_\Theta$ and initial state $U_{\Xi init}$ is computable. We begin by introducing the notion of a set of reachable configurations as follows.

**Definition 5.1.** *A set of usage configurations $\{U_{\Xi p_1}, U_{\Xi p_2}, \ldots, U_{\Xi p_t}\}$ is said to be reachable from the initial configuration $U_{\Xi init}$ in $k$ or less steps if and only if for each $U_{\Xi p_i}$, either $U_{\Xi p_i} = U_{\Xi init}$ or there exists a sequence of command instances $uc_{i_1} uc_{i_2} \ldots uc_{i_t}, 1 \leq t \leq k$ which begins at $U_{\Xi init}$ and end at $U_{\Xi p_i}$. The set is denoted as $\widehat{U_{\Xi init \uparrow k}}$.*

The set of configurations reachable in zero steps $\widehat{U_{\Xi init \uparrow 0}}$ is the singleton $\{U_{\Xi init}\}$. The set of configurations reachable from $U_{\Xi init}$ in unbounded number of steps is denoted by $\widehat{U_{\Xi init \uparrow *}}$. In general $\widehat{U_{\Xi init \uparrow *}}$ is countably infinite, since the number of objects that can be created is unbounded. Each newly created object by definition results in a new configuration. However, for a fixed $k$ the set of reachable configurations is finite as shown below.

**Lemma 5.2.** *$\widehat{U_{\Xi init \uparrow k}}$ is finite.*

**Proof.** This is trivially true for $k = 0$, which is our basis case.

Assume the inductive hypothesis that $\widehat{U_{\Xi init \uparrow k}}$ is finite for $k \leq n$. Consider $k = n + 1$. A usage configuration

$U_{\Xi p} \in \widehat{U_{\Xi init\uparrow n+1}}$ must either already be in $\widehat{U_{\Xi init\uparrow n}}$, or must result from application of a single usage command to some usage configuration $U_{\Xi q} \in \widehat{U_{\Xi init\uparrow n}}$. The former case does not add any new usage configurations. Consider the latter case. The number of such possible $U_{\Xi q}$ configurations is finite by induction hypothesis. For any non-creating command instance $uc_i(s,o)$ there are only a finite number of combinations of applicable $(s,o)$ pairs that can be actual parameters to $uc_i$ in any $U_{\Xi q}$. For any creating command instance $uc_i(s,o)$ there are only a finite number of applicable subjects $s$ that can be the actual parameter to $uc_i(s,o)$ in any $U_{\Xi q}$ (note that the $o$ parameter in a creating command is not an input but rather a result of the command). Thus there can only be a finite number of new usage configurations in $\widehat{U_{\Xi init\uparrow n+1}}$ relative to $\widehat{U_{\Xi init\uparrow n}}$. □

The next two definitions introduce the crucial concept of protection tuple equivalence.

**Definition 5.3.** *A protection tuple PT belongs to a set of usage configurations $\widehat{U_{\Xi i}}$, written $PT \in \widehat{U_{\Xi i}}$, if and only if there is a usage configuration $U_{\Xi p} \in \widehat{U_{\Xi i}}$ such that $PT \in f_\Xi(U_{\Xi p})$.*

The set of reachable *PTs* is thereby defined as $f_\Xi(\widehat{U_{\Xi init\uparrow *}})$ or, equivalently, as $f_\Xi(\widehat{U_{\Xi init\uparrow k}})$ for some $k$. If this set is computable we can answer the safety question. Note that even though $\widehat{U_{\Xi init\uparrow *}}$ is countably infinite, $f_\Xi(\widehat{U_{\Xi init\uparrow *}}) \subseteq \widehat{PT_\Theta}$ is finite since $|\widehat{PT_\Theta}| = \eta_{max}$.

**Definition 5.4.** *Two sets of usage configurations $\widehat{U_{\Xi i}}$ and $\widehat{U_{\Xi j}}$ are said to be protection equivalent, written $\widehat{U_{\Xi i}} \approx_{PT} \widehat{U_{\Xi j}}$, if and only if $f_\Xi(\widehat{U_{\Xi i}}) = f_\Xi(\widehat{U_{\Xi j}})$.*

We now prove the following theorem which is central to the results of this paper.

**Theorem 5.5.** $\widehat{U_{\Xi init\uparrow \eta_{max}}} \approx_{PT} \widehat{U_{\Xi init\uparrow *}}$.

**Proof.** Follows from lemmas 5.6 and 5.7 below. □

**Lemma 5.6.** *Suppose there exists $k$ such that $\widehat{U_{\Xi init\uparrow k+1}} \approx_{PT} \widehat{U_{\Xi init\uparrow k}}$. Then for all $n \geq k+2$, $\widehat{U_{\Xi init\uparrow n}} \approx_{PT} \widehat{U_{\Xi init\uparrow k}}$.*

**Proof.** By assumption, $\widehat{U_{\Xi init\uparrow k+1}} \approx_{PT} \widehat{U_{\Xi init\uparrow k}}$. Assume for contradiction that $\widehat{U_{\Xi init\uparrow k+2}} \not\approx_{PT} \widehat{U_{\Xi init\uparrow k+1}}$. Since $\widehat{U_{\Xi init\uparrow k+1}} \subseteq \widehat{U_{\Xi init\uparrow k+2}}$, this can happen only if there is some $PT_q \in f_\Xi(\widehat{U_{\Xi init\uparrow k+2}})$ and $PT_q \notin f_\Xi(\widehat{U_{\Xi init\uparrow k+1}})$. This $PT_q$ must be the result of some command $uc_i(s,o)$ enabled by a $PT_r \in f_\Xi(\widehat{U_{\Xi init\uparrow k+1}})$. But then $PT_r \in f_\Xi(\widehat{U_{\Xi init\uparrow k}})$ and $PT_q$ could have been produced in $f_\Xi(\widehat{U_{\Xi init\uparrow k+1}})$, which is a contradiction to $PT_q \notin f_\Xi(\widehat{U_{\Xi init\uparrow k+1}})$. This argument can be applied inductively for $n > k+2$. □

**Lemma 5.7.** $\widehat{U_{\Xi init\uparrow \eta_{max}+1}} \approx_{PT} \widehat{U_{\Xi init\uparrow \eta_{max}}}$.

**Proof.** There are at most $\eta_{max}$ *PTs* that can be added to a given $\widehat{U_{\Xi init\uparrow 0}}$. The longest chain of such additions is at most $\eta_{max}$ (which would happen if only one new $PT$ is added at each step). Thus it is not possible to have a $PT_q \in f_\Xi(\widehat{U_{\Xi init\uparrow \eta_{max}+1}})$ and $PT_q \notin f_\Xi(\widehat{U_{\Xi init\uparrow \eta_{max}}})$. □

It remains to prove the following theorem.

**Theorem 5.8.** *The set $\widehat{U_{\Xi init\uparrow \eta_{max}}}$ is computable and can be used to answer any safety question $\Upsilon_{(r)}$.*

**Proof.** By definition $\Upsilon_{(r)}$ can be true if and only if there is some $PT$ in $f_\Xi(\widehat{U_{\Xi init\uparrow *}})$ which satisfies the pre-condition of a creating or non-creating command that grants $r$. By Theorem 5.5 such a $PT$ must then also belong to $f_\Xi(\widehat{U_{\Xi init\uparrow \eta_{max}}})$.

Procedure Q given below computes $\widehat{U_{\Xi init\uparrow \eta_{max}}}$ from which $f_\Xi(\widehat{U_{\Xi init\uparrow \eta_{max}}})$ can be trivially computed. Let $\widehat{U_{\Xi init\uparrow =k}}$ denote the set of configurations that can be reached from $U_{\Xi init}$ by using exactly $k$ usage commands. This is similar to Definition 5.1 except for requiring exactly $k$ commands rather than $k$ or less.

**Procedure Q**:
Let $\widehat{U_{\Xi init\uparrow =0}} = \{U_{\Xi init}\}$;
Let $\widehat{U_{\Xi init\uparrow 0}} = \{U_{\Xi init}\}$;
Let $k = 1$;
While $k \leq \eta_{max}$ do begin
  $\widehat{U_{\Xi init\uparrow =k}} = \emptyset$;
  For each $U_{\Xi p} \in \widehat{U_{\Xi init\uparrow =k-1}}$ do begin
    Add all $U_{\Xi q}$ reachable from $U_{\Xi p}$
    in one command to $\widehat{U_{\Xi init\uparrow =k}}$
    without any duplicates;
  end for;
  $\widehat{U_{\Xi init\uparrow k}} = \widehat{U_{\Xi init\uparrow k-1}} \cup \widehat{U_{\Xi init\uparrow =k}}$;
  $k = k + 1$;
end while;

By the argument given in the proof of Lemma 5.2, each iteration of the for loop above is computable, and thereby Procedure Q computes $\widehat{U_{\Xi init\uparrow \eta_{max}}}$. □

## 5.1 Complexity

Theorem 5.8 proves the existence of SDPs. Implementation of an SDP based on the approach used in the proof would consume huge amount of computational resources. While there are obvious optimizations that could be made, we have the following result concerning complexity of the safety problem.

**Theorem 5.9.** *Possible SDPs for the usage authorization control model with finite attribute value domains are in PSPACE-Hard.*

*Proof sketch.* The hardness proof is based on a polynomial time reduction which maps an arbitrary instance of satisfiability of quantified Boolean formula (QBF) to an instance of safety problem in usage control pre-authorization model. Possible SDPs for the model should also solve the satisfiability of QBF which is a hard problem in PSPACE. Hence, the possible SDPs for the model are in PSPACE-Hard and the detailed proof is given in [3].

Our attempts to devise *efficient* SDPs have resulted in very restricted models. For example, a fragment of the model with Boolean attribute domains, restricted update actions and bi-conditional usage predicates, has an efficient SDP with a polynomial time algorithm. However, its expressivity is severely limited. Efficient design of SDPs for useful specialized cases is beyond the scope of this paper and is left for future work.

## 6 Conclusion and Discussion

In this work, we proved the existence of a safety decision procedure for automating the safety analysis of usage control authorization schemes with potentially unbounded number of objects. We also provided construction of one such SDP. This result significantly improves the state of the art in the safety decidability result by relaxing the boundedness in object creation. In practice, it will help us to design security mechanisms without keeping limits on the size of resource pools they meant to protect. This result is particulary relevant for today's computing systems with intrinsically dynamic resource pools.

We believe that the safety analysis of other sub models of $UCON_{ABC}$ with finite attribute domains is also decidable. Extending this procedure for safety analysis of sub models $UCON_{ABC}$ with ongoing authorization and obligations may be feasible. Developing these results in detail is outside the scope of this paper.

## Appendix

In Section 3.2.2 we mentioned that delete operations in the $Pre\_UCON_A$ model can be ignored for purpose of safety analysis. This appendix provides additional details in support of this argument. A deleting command has the following structure.

$\mathbf{Command\_Name}_r(o_1, o_2)$
**PreCondition:** $f_b(o_1, o_2) \rightarrow \{yes, no\}$;
**PreUpdate:** *delete* $o_2$;
$o_1.a_{i_1} := f_{1,a_{i_1}}(o_1, o_2)$;
...
$o_1.a_{i_p} := f_{1,a_{i_p}}(o_1, o_2)$;

We argue that given any usage control schema $U_\Theta$ with a deleting usage command, we can construct another usage schema $U'_\Theta$ without the deleting command such that $U_\Theta$ and $U'_\Theta$ are in *simulation* relation with respect to safety properties.

Let the usage control schema $U_\Theta = (OS_\Delta, UR, UC = \{UC_1, UC_2, \ldots, UC_{n-1}, UC_d\})$ be the schema with a deleting command $UC_d$. We construct the schema $U'_\Theta = (OS'_\Delta, UR, UC')$ as follows

- $OS'_\Delta = [a'_1 : \Omega'_1, a'_2 : \Omega'_2, \ldots, a'_n : \Omega'_n, deleted = \{yes, \mathbf{no}\}]$, where $\forall i, a'_i = a_i, \Omega_i = \Omega'_i$ and *deleted* is a special attribute with a special domain $\{yes, no\}$ with default value '*no*'.
- UR′=UR
- UC′=$\{UC'_1, UC'_2, \ldots, UC'_{n-1}, UC'_d\}$, where
  - –the delete command $UC_d$ in $UC$ is replaced with an update command $UC'_d$ in $UC'$.
    The $PreCondition$ in $UC'_d$ is $f'_{b_d}(o_1, o_2) \wedge f_{\overline{d}}(o_1, o_2)$, where $f'_{b_d}(o_1, o_2) = f_{b_d}(o_1, o_2)$ and the new predicate $f_{\overline{d}}(o_1, o_2)$ is defined as ($o_1.deleted = no\, o_2.deleted = no$).
    The $PreUpdates$ in $UC'_d$ are same as those in $UC_d$ except *delete* $o_2$ which is replaced with an update function $o_2.deleted := yes$.
  - –the create commands $UC'_{i,1 \le i \le n-1} \in UC'$ are same as $UC_i \in UC$ except the $PreCondition$. The $PreCondition$ in $UC'_i$ is $f'_{b_i}(o_1) \wedge f_{\overline{d}}(o_1)$, where $f'_{b_i}(o_1) = f_{b_i}(o_1)$ and the new predicate $f_{\overline{d}}(o_1)$ is defined as ($o_1.deleted = no$).
  - –the update commands $UC'_{j,1 \le j \le n-1} \in UC'$ are also same as $UC_j \in UC$ except the $PreCondition$. The $PreCondition$ in $UC'_j$ is $f'_{b_j}(o_1, o_2) \wedge f_{\overline{d}}(o_1, o_2)$, where $f'_{b_j}(o_1, o_2) = f_{b_j}(o_1, o_2)$ and the new predicate $f_{\overline{d}}(o_1, o_2)$ is defined as ($o_1.deleted = no$ && $o_2.deleted = no$).

Let $U_{\Xi init} = \{o_1, o_2 \ldots o_k\}$ be the initial configuration of $U_\Theta$, the initial configuration of $U_{\Theta'}$, that is $U_{\Xi init'}$, is constructed from $U_{\Xi init}$ by adding the extra binary attribute "*deleted*" with the value "*no*" to each object $o_i \in U_{\Xi init}$.

For any given initial configuration $U_{\Xi init}$, a usage command $UC_i$ is executable in a usage configuration $U_{\Xi x}$, $U_{\Xi x} \in U_{\Xi init\uparrow *}$ iff there exist a $U_{\Xi x'}$, $U_{\Xi x'} \in U_{\Xi init'\uparrow *}$ such that the usage command $UC'_i$ is executable in $U_{\Xi x'}$ and vice-versa.

Hence, the usage schema $U_\Theta$ is safe with respected to the safety question $\Upsilon_{(s_1, o_1, r)}$ iff the schema $U'_\Theta$ is safe and vice-versa. Rigorous formal proof for this claim can be constructed using the state-matching reduction technique given in [27].

## References


[1] J. Park and R. Sandhu, "The $UCON_{ABC}$ usage control model," *ACM Trans. Inf. Syst. Security*, vol. 7, no. 1, pp. 128–174, Feb. 2004.

[2] X. Zhang, R. Sandhu, and F. Parisi-Presicce, "Safety analysis of usage control authorization models," in *Proc. ACM Symp. Inf., Comput. Commun. Security*, Mar. 21–24, 2006, pp. 243–254.

[3] P. V. Rajkumar, "Formal and semi-formal methods for application specific security and usage control," Ph.D. dissertation, Indian Inst. Technol., Kharagpur, India, Aug. 2012.

[4] Z. Zhigang, W. Jiandong, and M. Yuguang, "Study and safety analysis on $UCON_{onA}$ model," in *Proc. 1st Int. Workshop Database Technol. Appl.*, Apr. 25–26, 2009, pp. 103–106.

[5] S. Ranise and A. Armando, "On the automated analysis of safety in usage control: A new decidability result," in *Proc. 6th Int. Conf. Netw. Syst. Security*, Nov. 21–23, 2012, pp. 15–28.

[6] M. Harrison, W. Ruzzo, and J. Ullman, "Protection in operating systems," *Commun. ACM*, vol. 19, no. 8, pp. 461–471, Aug. 1976.

[7] R. Sandhu, "The schematic protection model: Its definition and analysis for acyclic attenuating schemes," *J. ACM*, vol. 35, no. 2, pp. 404–432, 1988.

[8] R. Sandhu, "Expressive power of the schematic protection model," *J. Comput. Security*, vol. 1, no. 1, pp. 59–98, 1992.

[9] V. Varadharajan and C. Calvelli, "Extending the schematic protection model - i. conditional tickets and authentication," in *Proc. IEEE Symp. Res. Security Privacy*, May 16–18, 1994, pp. 213–229.

[10] V. Varadharajan, "Extending the schematic protection model - ii. revocation." *ACM SIGOPS Oper. Syst. Rev.*, vol. 31, pp. 64–77, Jan. 1997.

[11] R. Sandhu, "The typed access matrix model," in *Proc. IEEE Symp. Security Privacy*, May 4–6, 1992, pp. 122–136.

[12] M. Soshi, M. Maekawa, and E. Okamoto, "The dynamic-typed access matrix model and decidability of the safety problem," *IEICE Trans. Fundamentals*, vol. E87-A, no. 1, pp. 190–203, Jan. 2004.

[13] D. Ferraiolo, R. Sandhu, D. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," *ACM Trans. Inf. Syst. Security*, vol. 4, no. 3, pp. 224–274, Aug. 2001.

[14] J. Joshi, E. Bertino, and A. Ghafoor, "An analysis of expressiveness and design issues for the generalized temporal role-based access control model," *IEEE Trans. Dependable Secure Comput.*, vol. 2, no. 2, pp. 157–175, Apr.–Jun. 2005.

[15] R. Sandhu, V. Bhamidipati, and Q. Munawer, "The ARBAC97 model for role-based administration of roles," *ACM Trans. Inf. Syst. Security*, vol. 2, no. 1, pp. 105–135, Feb. 1999.

[16] J. Crampton and G. Loizou, "Administrative scope: A foundation for role-based administrative models," *ACM Trans. Inf. Syst. Security*, vol. 6, no. 2, pp. 201–231, May 2003.

[17] S. Osborn, "Information flow analysis of an RBAC system," in *Proc. 7th ACM Symp. Access Control Models Technol.*, Jun. 03–04, 2002, pp. 163–168.

[18] M. Koch, L. Mancini, and F. Parisi-Presicce, "Decidability of safety in graph-based models for access control," in *Proc. 7th Eur. Symp. Res. Comput. Security*, Oct. 14–16, 2002, pp. 229–243.
[19] N. Li, J. Mitchell, and W. Winsborough, "Design of a role-based trust-management framework," in *Proc. IEEE Symp. Security Privacy*, May 12–15, 2002, pp. 114–130.
[20] S. Stoller, P. Yang, C. Ramakrishnan, and M. Gofman, "Efficient policy analysis for administrative role based access control," in *Proc. 14th ACM Conf. Comput. Commun. Security*, Oct. 29–Nov. 2, 2007, pp. 445–455.
[21] E.Bertino, B. Catania, E. Ferrari, and P. Perlasca, "A logical framework for reasoning about access control models," *ACM Trans. Inf. Syst. Security*, vol. 6, pp. 71–127, 2003.
[22] N. Li and M. Tripunitara, "Security analysis in role-based access control," *ACM Trans. Inf. Syst. Security*, vol. 9, no. 4, pp. 391–420, Nov. 2006.
[23] K. Jayaraman, M. Tripunitara, V. Ganesh, M. Rinard, and S. Chapin, "MOHAWK: Abstraction-refinement and bound-estimation for verifying access control policies," *ACM Trans. Inf. Syst. Security*, vol. 15, no. 4, p. 18, Apr. 2013.
[24] K. Jayaraman, V. Ganesh, M. Tripunitara, M. Rinard, and S. Chapin, "Automatic error finding in access-control policies," in *Proc. ACM Conf. Comput. Commun. Security*, Oct. 2011, pp. 17–21.
[25] A. Armando and S. Ranise, "Scalable automated symbolic analysis of administrative role-based access control policies by smt solving," *J. Comput. Security*, vol. 20, no. 4, pp. 309–352, Jul. 2012.
[26] A. Ferrara, P. Madhusudan, and G. Parlato, "Security analysis of role-based access control through program verification," in *Proc. IEEE 25th Comput. Security Found. Symp.*, Mar. 2012, pp. 113–125.
[27] M. V. Tripunitara and N. Li, "A theory for comparing the expressive power of access control models," *J. Comput. Security*, vol. 15, no. 2, pp. 231–272, Apr. 2007.

**Rajkumar P.V.** received the PhD degree from the Indian Institute of Technology, Kharagpur, India. He is a postdoctoral fellow in the Institute for Cyber Security, University of Texas at San Antonio, TX. His research interests are formal verification and information security.

**Ravi Sandhu** is the founding executive director in the Institute for Cyber Security, University of Texas San An-tonio, and holds an Endowed chair. He is an inventor on 29 patents. He is a past editor-in-chief of the *IEEE Transactions on Dependable and Secure Computing*, a past founding editor-in-chief of ACM *Transactions on Information and System Security*, and a past chair of ACM SIGSAC. He founded ACM CCS, SACMAT, and CODASPY, and has been a leader in numerous other security conferences. His research has focused on security models and architectures, including the seminal role-based access control model. His papers have accumulated over 26,000 Google Scholar citations, including over 6,400 citations for the RBAC96 paper. He is a fellow of the ACM, IEEE, and AAAS.