Model-based Behavioral Attestation

Masoom Alam
Institute of Management Sciences, Pakistan
mmalam@imsciences.edu.pk

Xinwen Zhang
Samsung Information Systems America, San Jose, CA, USA
xinwen.z@samsung.com

Mohammad Nauman
Institute of Management Sciences, Pakistan
nauman@imsciences.edu.pk

Tamleek Ali
Institute of Management Sciences, Pakistan
tamleek@imsciences.edu.pk

Jean-Pierre Seifert
Samsung Information Systems America, San Jose, CA, USA
j.seifert@samsung.com

ABSTRACT

Remote attestation is an important characteristic of trusted computing technology which provides reliable evidence that a trusted environment actually exists. Existing approaches for the realization of remote attestation measure the trustworthiness of a target platform from its binaries, configurations, properties or security policies. All these approaches are low-level attestation techniques only, and none of them define what a trusted behavior actually is and how to specify it. In this paper, we present a novel approach where the trustworthiness of a platform is associated with the behavior of a policy model. In our approach, the behavior of a policy model is attested rather than a software or hardware platform. Thus, the attestation feature is not tied to a specific software or hardware platform, or to a particular remote attestation technique, or to an individual type of security policy. We select usage control (UCON) as our target policy model as it is a comprehensive and flexible model. We propose a framework to identify, specify, and attest different behaviors of UCON.

Categories and Subject Descriptors

D.4.6 [**OPERATING SYSTEMS**]: Security and Protection—*Access controls; Information flow controls*

General Terms

Security

Keywords

Remote attestation, UCON, behavioral attestation, high-level framework

1. INTRODUCTION

The term "Trusted Computing" refers to a technology introduced by the *Trusted Computing Group* (TCG) [2], in which PCs, consumer electronic devices, PDA's and other mobile devices are equipped with a special hardware chip called *Trusted Platform Module* (TPM).

TCG defines trust as follows: "*trust is the expectation that a device will behave in a particular manner for a specific purpose*" [3]. The term *particular manner* is concerned with the question of *how* a task is expected to be performed; *specific purpose* refers to a particular task or scenario like usage of an object, web service access, or some computational activity. In other words, *"trust is directly associated with the expected behavior of a particular task"*.

Remote attestation is an essential characteristic of trusted computing which provides reliable evidence that a trusted environment actually exists. This feature enables a trusted computing platform to remotely certify to third parties in enciphered form the behavior of its running software and the status of its hardware and software components. In a typical remote attestation scenario (cf. Fig 1), a *challenger* verifies the trustworthiness of a *target* or *remote platform* before dispatching a resource, or before or during an access to an object. If the target platform has provable trusted environment, sensitive information can be released to it.

Several approaches have been proposed in the literature for the realization of remote attestation. For instance, configuration - based attestation requires that a target platform presents the trusted configurations of its platform to a challenger. Based on the configurations, the challenger determines the trustworthiness of a target platform. However, revealing all system configurations may give some insights into the target platform, thus making a security attack inevitable [13].

Binary attestation has been studied in detail in two approaches: IMA [10] and PRIMA [6]. In IMA, a target platform presents the trusted status (i.e. binary hashes) of all its components loaded after booting to a challenger. However, IMA alone is not very practical in open and heterogeneous environments. PRIMA [6] enhances the IMA approach by

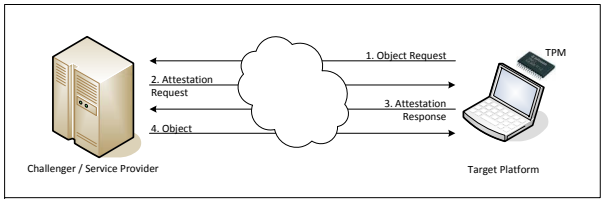

Figure 1: A Typical Remote Attestation Scenario

SACMAT'08, June 11–13, 2008, Estes Park, Colorado, USA.

enabling integrity measurement via information flow control using SELinux [1] policies. However, PRIMA still relies on the binary measurements of trusted subjects, and only considers a simple behavior of the system: low integrity data is filtered before flowing to high integrity subjects. Therefore, PRIMA alone cannot capture the dynamic behaviors of general platforms where many other security policies can be required.

In order to overcome the limitations of binary and configuration based attestations, property-based attestation [9, 4] proposes to collectively map related system configurations to some properties. For example, "Multi Level Security (MLS) enabled" is a property which can be mapped to system configurations such as the support of MLS in SELinux policies. In this way, the configurations of a target platform are not disclosed to a challenger. However, considering an enormous growth in hardware and software platforms especially in open environments like Internet, the mapping of all system configurations to properties is not very feasible and such mappings quickly become intractable.

Each of these existing low-level remote attestation techniques is useful in some scenarios but becomes infeasible in others. The fundamental issue of these techniques lies on the lack of a mechanism to identify and attest the behaviors of a platform instead of binaries and static configurations. In order to capture the dynamic behaviors of a system, there is a need to use these techniques together in a supporting manner so that the weaknesses of one can be addressed by another.

Semantic remote attestation [5] proposes a program or software's semantic analysis and requires a Trusted Virtual Machine (TVM) running on a target platform. In order to verify the behavior of a running software, the TVM monitors the security policy attached to the software which is based on different properties of the software. TVM is binary attested in order to ensure that the security policy attached to a software is indeed enforced. Li et al. [7] have coined the term behavioral attestation and presented an approach where the behavior of the security policy of a target platform is attested. In their approach, a complete analysis of the entire security policy of a target platform is proposed. For behavioral analysis, the approach uses a very simple example from Unix password file that "Alice can do this and Bob cannot"; therefore, the approach has a very limited demonstrated scope.

These two approaches propose that the trustworthiness of a target platform should be associated with the behavior of its security policies. However, none of these approaches define what the correct behavior of a security policy is and how to specify it. Without such a benchmark, there is nothing with which the enforcement behavior can be compared to draw conclusions about trustworthiness. Moreover, due to an enormous number of different types of security policies, e.g., different operating system policies, web service policies, and so on, it is also not feasible to associate the trustworthiness of a platform with its security policies and such associations quickly become intractable.

Our Approach: In this paper, we present a novel approach – Model-Based Behavioral Attestation (MBA) – in which the trustworthiness of a target platform is associated with the behavior of its *policy model*. As a policy model provides an abstract and formal representation of the security properties of a platform, we define trust as follows: "*trust is the expectation that a* policy model *will behave in a particular manner for a specific purpose*," where, *"the behavior of a policy model refers to the aggregate of its observable actions or reactions in response to its environment."* Thus, in MBA, a target platform is trustworthy for a challenger if each behavior of that policy model is trustworthy, which is followed by the target platform for the specific purpose of the challenger.

Contributions: Our contributions in this paper are threefold: 1) We present MBA, which is a high-level framework that abstracts the details of low-level attestation techniques. It is not a new attestation technique. Rather, it provides a framework through which the existing low-level techniques can be selected based on different scenarios and transformations. 2) We generalize behavioral attestation [7] and associate behaviors with policy *models* instead of individual security policies. Thus, the attestation is more exact, simple and scalable. 3) We identify and specify the correct behavior of UCON as an example target policy model for demonstration of MBA.

UCON is a comprehensive usage control model, which covers all aspects of usage control (cf. Section 2). This selection is motivated by the fact that usage control scenarios are concerned with the enforcement of access control policies about an object, which does not necessarily remain within the domain of its stakeholder. For example, in health care scenarios, medical data can be accessed within private clinics, laboratories and/or other hospitals. Without guaranteeing the correct enforcement of usage control policies, it is impossible to impose constraints on object usage. Thus, UCON is a classical model for remote attestation.

Outline: Section 2 gives an overview of UCON. Section 3 presents our approach of Model-based Behavioral Attestation. The identification of different behaviors associated with the UCON model is described in Section 4 and its specification in Section 5. Section 6 elaborates the attestation of enforcement behaviors. In Section 7, we conclude our contributions and present future directions for this research.

2. USAGE CONTROL (UCON)

Access to an object is not limited to server side and there are several unresolved issues after the release of the object to a remote platform. These issues are concerned with the continuous control over the usage of an object by a subject after it is released, e.g., the decrease, expiry, or revocation of permissions of a subject. Park and Sandhu [8] have coined the term usage control and proposed a model called *Usage CONtrol* (UCON), which enhances traditional access control models [12, 11] in two respects: 1) continuity of an access decision and, 2) mutability of attributes. *Continuity of an access decision* means that a decision to allow access to an object is not only made before access but also during access and may result in revocation of access permissions if policy conditions are no longer satisfied. *Mutability of attributes* means that attributes of subjects or objects may change[1] as side effects of access, which may also result in a change in ongoing or subsequent access decisions.

Policy statements in UCON are categorized as *authorizations*, *obligations* and *conditions*. Authorizations refer to predicates which are based on subject or object

[1]Only the mutability of subject and object attributes is allowed in UCON.

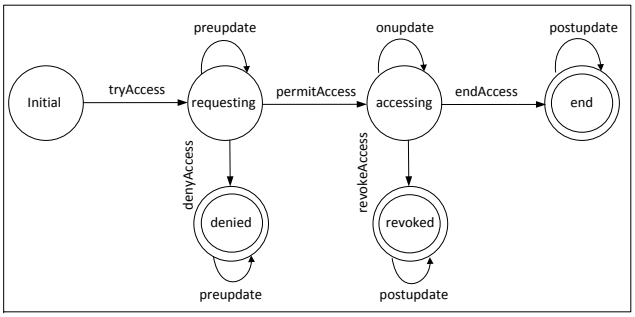

Figure 2: UCON Model States

attributes only. Obligations are directives to a subject to perform additional actions before or during an access. Predicates exclusively based on environment attributes such as system time, device type etc., are categorized as conditions.

Zhang et al. [15] have proposed a formal model for UCON policy specification. In this paper, we use this formalized model to present our approach of behavior specification and as a basis for a framework for behavioral attestation. A brief introduction to the logical specification of UCON is presented in the following.

Figure 2 shows the state transition model of UCON. Each state is associated with a state transition action. For instance, state *accessing* is associated with the state transition action *permitAccess* and *revoked* is associated with *revokeAccess*. For attributes mutability, each state has attribute update actions associated with it. State *requesting*, for instance, is associated with the attribute update action *preupdate*, *accessing* is associated with *onupdate*, and so on. According to [15], these attribute update actions are dependent on the type of the corresponding attribute and a separate program/software can be used to update different types of attributes. For example, date and time attributes can be updated by one software whereas, role attribute can be updated by a different software. Upon the successful update of an attribute, the corresponding attribute update action returns *true*; otherwise it returns $false$. A state transition action can be associated with authorization and condition predicates, obligations and attribute update actions, each of which returns either *true* or $false$. In this paper, we use the definition of UCON model given in [15].

Definition 1. *A* UCON model *is a 5-tuple:* $M = (T, P_A, P_C, A_A, A_B)$. *Where:*

- T *is a set of sequence of system states (e.g., accessing, revoked etc),*
- P_A *is a finite set of authorization predicates built from the attributes of the subjects and objects,*
- P_C *is a set of conditions predicates built from the system (or environment) attributes,*
- A_A *is a finite set of usage control (attribute update) actions,*
- A_T *is a finite set of obligation actions.*

A UCON policy p *is a also a 5-tuple, each element of which is a subset of the corresponding element in* M. *A* UCON system *consists of a set of subjects* S, *a set of objects* O, *a set of rights* R, *a set of attributes* ATT, *and a set of UCON policies.*

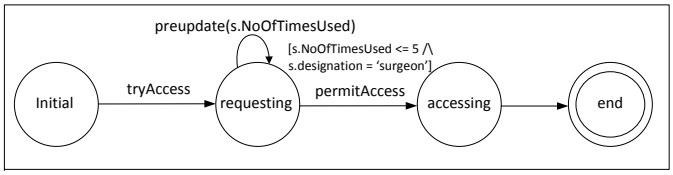

Figure 3: An Example UCON Policy Instance

Each policy model is built from different components. Based on Definition 1, we have identified 12 *components* of a UCON model. These are subjects, subject attributes, objects, object attributes, environment attributes, rights, authorizations, obligations, conditions, system states, state transition actions and attribute update actions.

Further, each UCON policy has the following meta information associated with it:

1. **Decision Timings:** Decision timings specify the state at which the usage decision is made. There are two types of decision timings: *pre* and *on*. The decision timing *pre* means that the decision to allow an access to an object is made before the access and *on* refers to usage control scenarios where decision to allow the access to an object is coupled with continuous usage of the object.
2. **Update Timings:** Update timings define the time when attributes are updated. Update timing 0 means that no update is allowed; 1 means before an access; 2 means during and 3 means after access. A policy can have accumulated update timings such as 0 only or 123.
3. **Policy Statement Type:** Depending on the type of a policy statement, a policy can be made up of authorizations only, obligations only, conditions only or any of their combination. The policy type is reflected in the policy name. Policy type $preAB_0$ includes policies in which policy statements consist of authorizations and obligations only, decision are made before an access and no attribute update is allowed.

The following example UCON policy formalizes the high-level security requirement from medical domain that access to a medical record is allowed at most 5 times and only to surgeons. In this example, state *accessing* is made conditional by associating authorization predicates with the state transition action *permitAccess* (cf. Fig 3). Before entering the *accessing* state (i.e., before access to the *medicalRecord* is granted), the $NoOfTimesUsed$ attribute of the subject s is updated using the attribute update action *preupdate* in the state *requesting*. This is an example of a $preA_1$ type policy.

Example 1. *The medical record can be read at most 5 times and only by a surgeon.*

1) $permitaccess(s,medicalRecord,read) \rightarrow$
2) $((s.NoOfTimesUsed \leq 5) \wedge (s.designation = \text{'surgeon'}))$
3) $\wedge\ preupdate(s.NoOfTimesUsed)$
4) $preupdate(s.NoOfTimesUsed):$
5) $(s.NoOfTimesUsed' = s.NoOfTimesUsed + 1)$

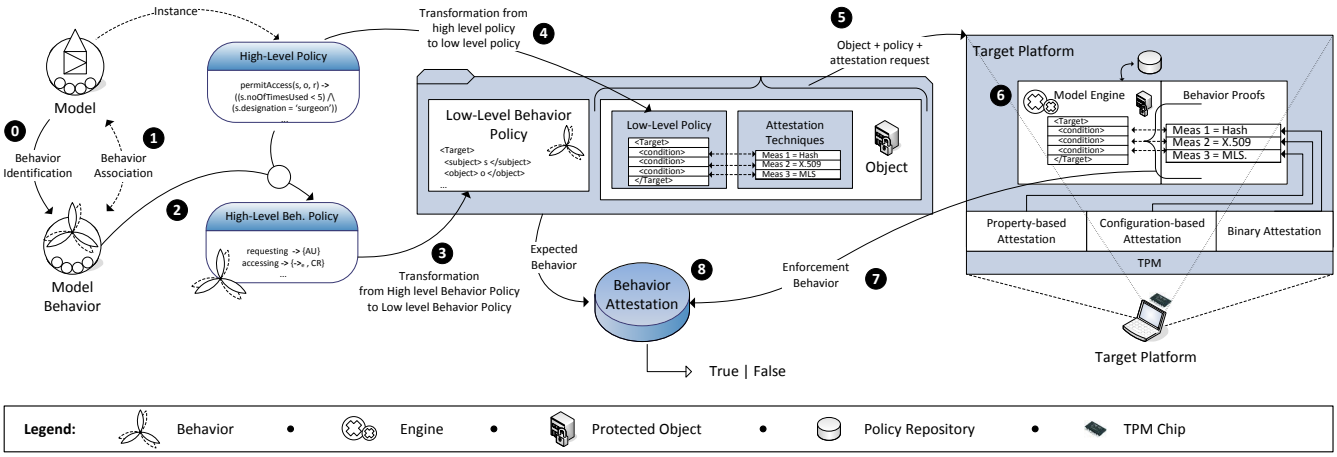

Figure 4: Model-based Behavioral Attestation (MBA)

Typically, remote attestation is to ensure that a remote platform has the trusted environment where sensitive data and resources can be released, or services and applications can be deployed. The two novel features of UCON – continuity of access decision and mutability of attributes – require that usages of data or access to services have to be continuously controlled even out of the stakeholder's domain and under dynamic computing environments. This makes UCON a prime candidate for remote attestation. For this reason, we have chosen UCON as an example target policy model for our MBA technique. Our approach of MBA is presented in detail in the following sections.

3. MODEL-BASED BEHAVIORAL ATTESTATION

Model-based Behavioral Attestation (MBA) is a framework for identifying, specifying and remotely attesting the behaviors associated with different components of a policy model. It is a *high-level* framework, built on top of low-level attestation techniques and is a general approach that can be realized for a variety of access control and usage control models. The biggest advantage of the MBA approach is that it is not tied to any specific software or hardware platform, or to a particular attestation technique or to an individual type of security policy. The framework helps to improve the realization of remote attestation by breaking down a statement like, "this system enforces a particular policy model," into *individual elements* of the policy enforcement. The behavior of each of these individual elements can then be attested through existing approaches like binary or property-based attestation.

Hereafter, we provide step-by-step design principles for the development of an attestation system based on MBA.

Steps 0 - 1) Behavior Identification & Association: In this step, a target policy model is analyzed thoroughly to identify a general set of behaviors associated with its distinct components (cf. Fig 4 – Step 0-1). This identification helps to categorize which set of behaviors of a policy model should be trusted. These behaviors are used to determine the trustworthiness of the target platform at which the policy model is realized. This step has to be performed only once for each policy model. The results can then be re-used in all systems which need to attest the policy model.

Step 2) Behavior Specification: In the behavior specification step, the system designer specifies the behavior of a policy model's components. A formal specification of such behaviors helps to abstract the complex details of the underlying hardware and software platforms.

We note that a policy is an instance of a model and is composed of several components of the model. The set of behaviors associated with different components of a policy are collectively referred as the *expected behavior* of the policy. An automatic way of associating such behaviors with different components of a policy model can simplify the behavior association and its specification. The end product of this step is a *high-level behavior policy*. We include one such algorithm which can automate this procedure during the specification of UCON behaviors in Section 5.

Steps 3 - 4) Transformations: Transformations play a key role in MBA. In Step 3, the policy of a target model is transformed to some concrete and low-level policy language like XACML [14]. The XACML behavior policies will specify the expectation of a challenger regarding the usage of her objects. This step bridges the gap between the high-level policy specifications and its corresponding low-level implementation. In Step 4, the high-level behavior policy is transformed to a *low-level behavior policy*. Depending on the underlying computing environment, different behavior transformations can be defined for one high-level policy. For example, for a relatively closed environment such as a hospital, a relaxed set of transformations can be employed – these relaxed transformations may require only property-based attestation. However, for open environments, a more restricted set of transformations might be used. The biggest advantage of having different transformations is that the high-level policy and its corresponding behavior policy remain the same and do not change for different underlying computing environments.

Steps 5 - 8) Behavior Attestation: In an MBA scenario, a target platform receives the object, enforcement policy and specification of measurements for collecting proofs of policy enforcement (Step 5). It uses the object as dictated by the policy and while doing so, collects proofs of its behaviors (Step 6). It then sends this behavior – called *enforcement behavior* – to the challenger (Step 7). In the behavior attestation step (Step 8), the enforcement behavior

of the policy is compared to its expected behavior. The verification mechanisms help the challenger in sorting out the trustworthiness of the target platform. This procedure can also be automated easily. We provide an algorithm for this automation in the context of UCON model behavioral attestation in Section 6.

In this paper, we focus on the specification and attestation of behaviors. Transformations are concerned with the implementation of MBA and are part of future work.

Our overall approach can be defined as follows:

DEFINITION 2. *(Behavior Specification) Let $\mathcal{M}$ be a target model in our model-based attestation. If $\mathcal{T}$ represents a finite set of different components in model $\mathcal{M}$ such that $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ and $E_x b$ represents a finite set of expected behaviors of $\mathcal{T}$ such that $E_x b = \{e_x b_1, e_x b_2, \ldots, e_x b_n\}$ then Behavior Association (BA) is a function that maps the available components $\mathcal{T}$ in model $\mathcal{M}$ to their expected behaviors $E_x b$. Formally:*

$$BA\text{: } \mathcal{T} \leftharpoonup E_x b$$

where $\leftharpoonup$ represents the behavior association relation.

DEFINITION 3. *(Behavior Verification) If $E_n b$ represents a finite set of enforcement behaviors of $\mathcal{T}$ in model $\mathcal{M}$ such that $E_n b$=$\{e_n b_1, e_n b_1, \ldots, e_n b_n\}$, then the comparison of enforcement behavior with the expected behavior takes the form: $\{e_n b_1 \otimes e_x b_1, e_n b_2 \otimes e_x b_2, \ldots, e_n b_n \otimes e_x b_n\}$ where $\otimes$ represents the comparison relation between the enforcement and expected behaviors. We define Behavior Verification (BV) as a partial function that maps the comparison of enforcement and expected behaviors ($E_n b \otimes E_x b$) to some boolean value. Formally:*

$$BV\text{: } E_n b \otimes E_x b \rightarrow \{true \mid false\}$$

4. BEHAVIOR IDENTIFICATION AND ASSOCIATION

In order to formally specify the behavior of a policy model, its components need to be identified. A policy model is built from its distinct components, the behavior of each of which effects the overall behavior of the policy model. Thus, in our viewpoint, *the behavior of a policy model is the aggregate of observable actions or reactions of its distinct components in response to their environment.* In this statement, the *environment* of a policy model is defined by its policy instance and *observable actions* refer to those actions which are performed by different components of a policy model during policy enforcement. Observability of actions means that there must be a way of communicating to the challenger that the required actions were performed.

Based on the components of UCON specified in Section 2, we identify three types of behaviors which need to be associated with the UCON model. These are (1) active subject/object behaviors, (2) state transition behaviors, and (3) attribute update behaviors.

1. *Active subject/object behaviors* capture the behaviors of all subjects and objects and their corresponding rights, which are active for a given system state. These behaviors correspond to the subjects, objects, and rights of UCON.

2. *State transition behaviors* capture the behaviors of state transitions when associated authorizations, obligations, and conditions are fulfilled. These behaviors correspond to authorizations, obligations, conditions, system states, and state transition actions of UCON. Moreover, these behaviors also capture the behavior of attributes involved in related predicates.

3. *Attribute update behaviors* capture the behaviors of attribute update actions such as *preupdate*, *onupdate*, and *postupdate*. These behaviors correspond to the update actions of subject and object attributes.

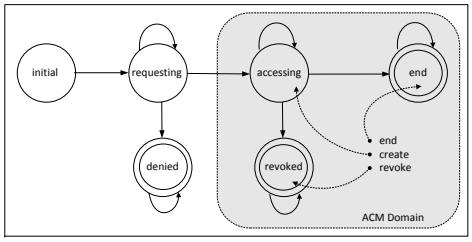

Figure 5: UCON ACM Domain

Note that although UCON considers conditions as decision making factors, it does not capture changes to conditional information (system attributes). Therefore, behaviors of system attribute updates are not included in MBA. In general, MBA trusts that system attributes are monitored and updated in a trusted way.

In general, the criteria for behavior analysis is complete, if it covers all the components of a given target policy model. Our behavior analysis of UCON is complete, since it covers all the components of the UCON model given in Definition 1.

4.1 Active Subject/Object Behaviors

UCON is a session oriented model in which a state transition activates or deactivates a subject and/or an object. We use Access Control Matrix (ACM) to capture the behaviors of subjects or objects added and removed as a result of different state transitions. We define UCON ACM as follows:

DEFINITION 4. *The ACM of a UCON system state is defined as $A : O \times R \rightarrow 2^S$ where O is a set of active objects, R is a set of rights, and S is a set of active subjects.*

Thus, if $s \in A(o, r)$, it means that subject s is exercising right r on object o in some state. For simplicity, we assume that there is only one usage session for a single (s, o, r) existing at one time. However, one subject can access multiple objects and one object can be accessed by multiple subjects at the same time. Figure 5 shows different UCON states and their associated *ACM actions*. The ACM actions mark the domain of active subjects and objects. For example, a subject or object is considered active when a state transition from *requesting* to *accessing* occurs. The ACM action *create* captures the corresponding behavior. Likewise, the ACM action *revoke* and *end* capture the behaviors when an access has been revoked and when an access is ended, respectively. These ACM actions are described below.

ACM Action *create*: Whenever a subject is permitted to access an object, the ACM action *create* adds a new subject s and object o to the set of active subjects S and the set of active objects O, respectively. The new set of active

subjects and set of active objects are denoted by S' and O', respectively. Further, the creator s is added to ACM for (o, r) and the modified ACM (A') is the new ACM. Formally:

$$create(s, o, r) = (O', S', R, A') \ \ where$$

$$S' = S \cup \{s\},\ O' = O \cup \{o\} \ \ and$$

$$A'(o, r) = A(o, r) \cup \{s\}$$

ACM Action *revoke***:** The ACM action *revoke* captures the behavior when a subject or object is removed from the set of active subjects and the set of active objects due to state transition from *accessing* to *revoked*. The *revoke* ACM action is defined as follows:

$$revoke(s, o, r) = (O', S', R, A') \ \ where$$

$$S' = S \ominus \{s\} =^{def} \begin{cases} S & |A^{-1}(\{s\})| \geq 2 \\ S - \{s\} & otherwise \end{cases}$$

$$O' = O \ominus \{o\} =^{def} \begin{cases} O & \sum_{i=1}^{n} |A(o, r_i)| \geq 2 \\ O - \{o\} & otherwise \end{cases}$$

$$and \ \ A'(o, r) = A(o, r) - \{s\}$$

In order to obtain the current set of active subjects, subject s is removed from the set of active subjects ($S \ominus \{s\}$). However, the removal places a restriction that if subject s is accessing more than one objects on the ACM – verified through the inverse of A – subject s is not removed. Similarly, before removing object o from the ACM, it is also verified that object o is not being accessed by any subjects through any right. Further, the ACM is updated at (o, r) by removing s from the ACM entry (o, r). The new set of active subjects and objects are denoted by S' and O', respectively.

ACM Action *end***:** Similar to *revoke*, the ACM action *end* conditionally removes the active subject and object from the ACM. Formally:

$$end(s, o, r) = (O', S', R, A')$$

The ACM action *revoke* is caused by a call from usage control system whereas, the ACM action *end* is caused by a subject itself. The effects of both the transitions are same, thus, both ACM actions result in the same behavior. Based on the ACM actions, we now define the trustworthiness of the UCON ACM as follows:

Definition 5. *The UCON ACM behavior is trustworthy if all of the following conditions hold:*

- $create(s, o, r) \rightarrow o \in O' \wedge s \in S' \wedge s \in A'(o, r)$
- $revoke(s, o, r) \rightarrow S' = S \ominus \{s\} \wedge O' = O \ominus \{o\} \wedge A'(o, r) = A(o, r) - \{s\}$
- $end(s, o, r) \rightarrow S' = S \ominus \{s\} \wedge O' = O \ominus \{o\} \wedge A'(o, r) = A(o, r) - \{s\}$

Three symbols $\mathcal{CR}$, $\mathcal{EN}$, $\mathcal{RK}$ are defined to show that trustworthiness is expected for ACM actions create, end and revoke, respectively.

The above definition states that the UCON ACM is trustworthy if following the ACM action $create(s, o, r)$, subject s is added to the set of active subjects (represented by S'), object o is added to the set of active objects (represented by O') and the ACM itself contains an entry (o, r) such that $s \in A(o, r)$. Similarly, based on the definitions of the ACM actions *end* and *revoke*, the trustworthiness of the ACM actions *end* and *revoke* is defined.

In general, for every identified behavior, the formal specification takes the form: first, the behavior and its trustworthiness are defined. Afterwards, a symbol is used to denote that trustworthiness is expected for the corresponding behavior.

4.2 State Transition Behaviors

In UCON, a state transition action is associated with a combination of authorization, obligation and condition statements. For simplicity, we assume that a logical expression represents all of them such that a state transition occurs only if the logical expression associated with the transition is *true*. We define a state transition behavior as follows:

Definition 6. *Given a state t_i and a logical expression e, we write $t_{i-1} \rightarrow_e t_i$ if the transition from state t_{i-1} to t_i is allowed by the logical expression e where e is a conjunction of authorization, obligation, and condition statements.*

Logical expressions contain subject, object and/or environment attributes. These attributes play a key role in sorting out usage control decisions in UCON. Due to their significant importance, the correct enforcement of UCON policies requires that all attributes involved in a particular usage control scenario should be trusted. If attributes, or the procedures responsible for updating them are not trusted, the end result is an untrusted attributes mutability, or untrusted decision continuity. Consider an example, when attribute like subject location or number of times that an object can be accessed is updated in an untrusted way, or the behavior of the login shell representing a subject or the memory area used to load an object is not trusted, the state transition resulting from this attribute predicate is not a valid transition; i.e., the resulting system state is not trusted. Instead of relying on normal attribute mutability and decision continuity – the key UCON concepts, we propose trusted attribute mutability and trusted decision continuity.

Trusted attribute mutability means that subject or object attributes can be updated, if and only if, the corresponding attribute behaviors are trusted and the procedures responsible for updating them are also trusted. Similarly, trusted decision continuity means that in addition to decision continuity features of UCON, all the attributes involved in a particular state transition should be in a trusted state.

Each subject s or object o is represented by a set of attributes in UCON. For trusted decision continuity, we define each attribute $attr_i$ to be a 3-tuple (*name*, *value*, *behavior*), where *name* and *value* have the same semantics as defined in UCON, and *behavior* represents the trusted status of the attribute. At model level, *behavior* is an abstract entity which can have different semantics for different attributes. For example, for memory, it can mean the status of memory protection against intrusions; for a binary file, it can mean the integrity of the file. The actual

semantics are augmented in the behavior transformation step. The behavior of an attribute is either $TRUSTED$ or $UNTRUSTED$, where a $TRUSTED$ value corresponds to a Boolean value *true* and $UNTRUSTED$ corresponds to a Boolean value $false$. We define the trustworthiness of a state transition behavior as follows:

DEFINITION 7. *Let* $Attributes = \{attr_1, attr_2, \ldots, attr_n\}$ *represent a finite set of subject, object and environment attributes defined in a logical expression e such that* $t_{i-1} \rightarrow_e t_i$. *Then, the trustworthiness of a state transition behavior takes the form:*

$$\forall j.\ attr_j.behavior = TRUSTED \wedge e = TRUE$$

where j=1,. . . ,n.

We use $\rightarrow_e$ *to denote that trustworthiness is expected for the corresponding state transition behavior.*

4.3 Attribute Update Behaviors

Attribute update actions such as *preupdate*, *onupdate* and *postupdate* enable attributes mutability. We define attribute update behavior as follows:

DEFINITION 8. *Given an attribute* x_i, *and an update operation 'update', the attribute update behavior is defined as the application of the update operation on the attribute. The application of the operation yields a new value which is assigned to the attribute. Formally:*

$$update : x_i \rightarrow x_i'$$

The trustworthiness of attribute update behavior is defined as follows:

DEFINITION 9. *An attribute update behavior is trustworthy iff the corresponding attribute is in a trusted state and the procedure updating it is also trusted. Formally:*

$$update(x_i) = \begin{cases} true & iff\ \ x_i.behavior = TRUSTED \wedge \\ & update.behavior = TRUSTED \\ false & otherwise \end{cases}$$

where $i = 1, ..., n$ *and* x_i *represents the subject or object attribute that needs to be updated.*

We write $\mathcal{AU}_x$ *to denote that a trusted change is expected for attribute x, and* $\mathcal{AU}$ *for the set of all attributes that can be updated.*

The *update* represents the *preupdate*, *onupdate* and *postupdate* attribute update actions. Any attribute update action is an $(update, behavior)$ pair, where *behavior* represents the trusted status of the corresponding update procedure.

5. BEHAVIOR SPECIFICATION

In this section, we present the formal specifications of UCON model behaviors and propose an algorithm for the automation of behavior specification of a UCON policy. A UCON policy is an instance of the UCON model, and is composed of several states (e.g., requesting, accessing etc.) Hence, each e_xb_i of Definition 2 is associated with a single state of the UCON policy. Moreover, depending on the type of the policy, each state can have different UCON *components* associated with it. For example, the policy type $preABC_0$ designates that no attribute update should be performed which means that no attribute update action

```
1) Function: BA
2) Input: Wellformed S_tates(p)
3) Output: Expected Behavior E_x b of Policy p
4)  p_t = getPolicyType(p); E_x b = ∅
5)  UpdateTimings = getUpdateTimings(p_t)
6)  Constuct the set AUA based on the value of the udpate timings.
7)  // e.g. if update timing is 12 then AUA = {preupdate,onupdate}
8)  foreach state t_i(s,o,r) in S_tates do
9)  e_x b=∅;
10) // Based on Definition 8, associate Behaviors
11) // with the states of Policy p
12)  if !(AUA = ∅) then // check for update timings 0
13)   if (update(x_i) in t_i) then
14)    e_x b = e_x b ∪ {AU_xi} //Attribute Update Behavior
15)  //Associate Active Subject/Object Behavior as follows:
16)  if (t_i=accessing) then // if state is accessing
17)    e_x b = e_x b ∪ {CR, →_e} // Associate ACL action create
18)                              // and state transition behavior
19)  if (t_i=end) then // if state is end
20)    e_x b = e_x b ∪ {EN, →_e} // Associate ACL action end
21)                              //and state transition behavior
22)  if (t_i=revoked) then// if state is revoked
23)    e_x b = e_x b ∪ {RK, →_e} //Associate the ACL action revoke
24)                              //and state transition behavior
25) t_i ← e_x b
26) E_x b = E_x b ∪ {e_x b}
27) end foreach
28) return E_x b
```

Figure 6: BA (Behavior Association) Function

can be associated with any state of the policy. Likewise, $preABC_0$ also requires that decision should be made before an access which means that state *revoked* is not a part of the policy instance of type $preABC_0$.

The behavior of the policy is determined by the aggregate of the behaviors of the components present in different states. Thus, each UCON policy can have a different set of behaviors associated with it depending on its policy type.

For simplicity, we assume that all policies have accumulated authorization, obligation and condition statements. However, decision timings, i.e., *pre* and *on*, and update timings, i.e., 0,1,2 and 3 are distinguishing factors. For behavior specification, we define expected behaviors associated with the states of a UCON policy as follows:

DEFINITION 10. *Let* T *represent the set of states in UCON model such that* $T = \{initial, requesting, denied, accessing, revoked, end\}$ *and p is a policy instance of UCON model. Then,* S_{tates} *is a function that maps the policy p to a set of states T. Formally:*

$$S_{tates} : p \rightarrow \mathcal{P}(T)$$

For each state t_i *for usage session* (s, o, r) *where* $t_i \in T$ *and* $s \in S$, $o \in O$,$r \in R$, *the expected behavior* e_xb_i *of state* t_i *is equal to a set of symbols used to denote that trustworthiness is expected for the identified behaviors. Formally:*

$$e_xb_i = \mathcal{P}\{\mathcal{CR} \mid \mathcal{EN} \mid \mathcal{RK}, \rightarrow_e, \mathcal{AU}\}$$

$$t_i(s, o, r) \leftharpoonup e_xb_i$$

The above definition describes the expected behavior of a state corresponding to the subject s, object o and right r. The expected behaviors are associated with a state using the behavior association relation $\leftharpoonup$. The set of behaviors associated with different states of a UCON policy are collectively referred to as the expected behavior (E_xb – by Definition 2) of the corresponding policy.

Based on these definitions, we now define the *Behavior Association* (BA) function (cf. Fig 6) which associates

different behaviors with different states of a UCON policy. The function takes a set of states from a UCON policy as input and returns its expected behaviors. We assume that the input of a UCON policy is well-formed and the logical expressions of the corresponding policy are also well-formed. A UCON policy is well-formed if it contains only the states allowed by it decision timings and the attribute update actions required by its update timings.

Depending on the update timings of a policy p, the BA function (cf. Fig 6) constructs the set AUA (Attribute Update Actions) (line 6). The AUA set ensures that if update timing is 0, no attribute update behavior shall be associated with any state of the policy p. Afterwards, for each state t_i in S_{tates}, the BA function associates the attribute update behaviors (line 12-14), state transition behaviors and active subject/object behaviors (line 16-24) with state t_i. For example, if there is an update expected in attribute x_i within the state t_i, then the corresponding behavior is added to the set of behaviors $e_x b_i$ (line 13-14). Similarly, if t_i is *accessing*, then the ACM action *create* and state transition behavior are augmented to $e_x b_i$ (line 16-17). At the end, each state t_i is associated with the set of behaviors $e_x b_i$ using the behavior association relation (line 25). The sets of behaviors $e_x b_i$ are collected in an overall expected behavior $E_x b$ (line 26). The function[2] returns this expected behavior $E_x b$ of the policy p (line 28). The set of expected behaviors of a policy is also called its *behavior policy*.

Using the BA function, we now specify the behaviors of the example UCON policy presented in Section 2.

Example 2. Behavior Policy for Example 1

1) $requesting(s,medicalRecord,read) \leftarrow \{\mathcal{AU}_{s.NoOfTimesUsed}\}$
2) $\mathcal{AU}_{s.NoOfTimesUsed} \Rightarrow ((s.NoOfTimesUsed' =$
3) $\quad s.NoOfTimesUsed + 1)$
4) $\quad iff\ (s.NoOfTimesUsed.behavior = TRUSTED \wedge$
5) $\quad preupdate.behavior = TRUSTED))$
6) $accessing(s,medicalRecord,read) \leftarrow \{\mathcal{CR}, \rightarrow_e\}$
7) $\mathcal{CR} \Rightarrow medicalRecord \in O' \wedge s \in S' \wedge$
8) $\quad s \in A(medicalRecord, read)$
9) $\rightarrow_e \Rightarrow (s.NoOfTimesUsed \leq 5 \wedge s.designation = \text{`surgeon'})$
10) $\quad \wedge\ s.NoOfTimesUsed.behavior = TRUSTED$
11) $\quad \wedge\ s.designation.behavior = TRUSTED$

In the above example behavior policy, the only behavior associated with the state *requesting* is the attribute update behavior (line 1), as described in lines 2-5. The state *accessing* is associated with a set of behaviors (line 6). The first element in the set of behaviors designates that the ACM action *create* (line 7-8) is expected in state *accessing* for the current subject and object. The second element is the expected state transition behavior (line 9-11). The last element in the set of behaviors (line 10-11) describes the fact that the attributes involved in the logical expression e are expected to be in trusted state.

6. BEHAVIOR ATTESTATION

In this section, we present the formal semantics of the framework for behavior attestation of UCON. *Behavior attestation* refers to a mechanism, which verifies that the distinct components of a target model perform their observable actions in expected manner. One of the biggest advantage of our approach is that, it is not tied to any specific software or hardware platform, thus attestation can be realized for a variety of softwares and hardware platforms. We assume that, the target platform sends enforcement behavior of a UCON policy. The enforcement behavior of a UCON policy is organized according to its states. Each state is associated with a set of proofs of the fulfillments of different expected behaviors. We define enforcement behavior as follows:

Definition 11. *Given a state t, an enforcement behavior $e_n b$ is a mapping from state t to a set of proofs of expected behaviors. Formally:*

$$e_n b : t \rightarrow \mathcal{P}\{\phi(\mathcal{CR}) \mid \phi(\mathcal{EN}) \mid \phi(\mathcal{RK}), \phi(\rightarrow_e), \phi(\mathcal{AU})\}$$

where ϕ is a function – called proof function, that takes an expected behavior as input and returns its trusted proof. For example, $\phi(\mathcal{CR})$ may return a proof of the expected ACM action *create* in the form of a trusted certificate, that advocates the current status of active subjects and objects in a particular state. The implementation of the proof function requires that the target platform should support the low-level techniques required for that individual proof. This implementation is left unspecified in our framework. The reason is that, we do not include such implementation issues following our practice of keeping the model and attestation mechanisms distinct.

In order to compare enforcement behaviors with expected behaviors, we define four *satisfies* relations. These relations are based on the enforcement behaviors of a state, its expected behaviors and the trustworthiness of corresponding expected behaviors. In general, an enforcement behavior of a state can only satisfy its expected behavior if and only if the corresponding expected behavior is trustworthy.

Definition 12. *Given a state t, an enforcement behavior $e_n b$ satisfies an expected ACM action iff ACM is trustworthy in state t. Formally:*

$$e_n b(t) \models_a \mathcal{CR} \text{ iff } o \in O' \wedge s \in S' \wedge s \in A'(o,r)$$
$$e_n b(t) \models_a \mathcal{RK} \text{ iff } S' = S \ominus \{s\} \wedge O' = O \ominus \{o\} \wedge A'(o,r) = A(o,r) - \{s\}$$
$$e_n b(t) \models_a \mathcal{EN} \text{ iff } S' = S \ominus \{s\} \wedge O' = O \ominus \{o\} \wedge A'(o,r) = A(o,r) - \{s\}$$

We have already defined the trustworthiness of the UCON ACM in Definition 5. The above definition uses Definition 5 to set a condition for the comparison between an expected ACM action behavior and its enforcement behavior. For example, an enforcement behavior can only satisfy the expected ACM action *create* behavior if it contains a proof that, subject s is added to the set of active subjects, object o is added to the set of active objects and the ACM itself contains an entry (o,r) such that $s \in A(o,r)$.

Definition 13. *Given a state t, an enforcement behavior $e_n b$ satisfies an expected state transition behavior iff state transition behavior is trustworthy in state t. Formally:*

$$e_n b(t) \models_e \rightarrow_e \text{ iff } \forall i.\ attr_i.behavior = TRUSTED \wedge e = TRUE, \text{ where } i=1,\ldots,n.$$

In other words, the above definition means that, in order to satisfy an expected state transition behavior, the enforcement behavior must contain a proof that a state

[2]The algorithm provided in Figure 6 has linear time and space complexity in the size of the policy.

transition has occurred only when the associated logical expression was *true* and all attributes present in the expression were trusted.

DEFINITION 14. *Given a state t, an enforcement behavior $e_n b$ satisfies an expected attribute update behavior iff attribute update behavior is trustworthy in state t. Formally:*

$$e_n b(t) \models_{au} \mathcal{AU}_{x_i} \text{ iff } x_i.behavior = TRUSTED \wedge update.behavior = TRUSTED \text{ where } i = 1, ..., n$$

```
1)  Function: BV
2)  Input: Enforcement Behavior E_n b, Expected Behavior E_x b
3)  Output: Boolean
4)    if (|E_x b| != |E_n b|) then return false;
5)    // if the number of states isn't the same in expected
6)    // and enforced behavior, return false
7)    foreach e_x b in E_x b do
8)      t_i = getState(e_x b)
9)      if ((→_e ∈ e_x b(t_i)) ∧ !(e_n b(t_i) ⊨_e →_e)) then return false;
10)     if ((→_e ∉ e_x b(t_i)) ∧ (e_n b(t_i) ⊨_e →_e)) then return false;
11)     if ((CR ∈ e_x b(t_i)) ∧ !(e_n b(t_i) ⊨_a CR)) then return false;
12)     if ((CR ∉ e_x b(t_i)) ∧ (e_n b(t_i) ⊨_a CR)) then return false;
13)     if ((EN ∈ e_x b(t_i)) ∧ !(e_n b(t_i) ⊨_a EN)) then return false;
14)     if ((EN ∉ e_x b(t_i)) ∧ (e_n b(t_i) ⊨_a EN)) then return false;
15)     if ((RK ∈ e_x b(t_i)) ∧ !(e_n b(t_i) ⊨_a RK)) then return false;
16)     if ((RK ∉ e_x b(t_i)) ∧ (e_n b(t_i) ⊨_a RK)) then return false;
17)     if ((AU_{x_i} ∈ e_x b(t_i)) ∧ !(e_n b(t_i) ⊨_au AU_{x_i})) then return false;
18)     if ((AU_{x_i} ∉ e_x b(t_i)) ∧ (e_n b(t_i) ⊨_au AU_{x_i})) then return false;
19) end foreach
20) return true
```

Figure 7: BV (Behavior Verification) Function

Based on these definitions, we now specify the *Behavior Verification* (BV) function (cf. Fig 7) which verifies the expected behaviors of a policy against its enforcement behaviors. The function takes the enforcement behavior and expected behaviors as input, and returns a Boolean value. In Behavior Verification, the first check is related to the number of states in expected and enforcement behaviors (line 4). If the number of component behaviors (and thus states) in the two behavior sets are not the same, verification fails. If the number of states match, the behavior of each component is then verified (line 7). For each expected behavior $e_x b$, the function first fetches the state t_i associated with $e_x b$ (line 8). Afterwards, using a series of `if` statement pairs, different enforcement behaviors are verified. For example, if the behavior $\rightarrow_e$ is expected in an $e_x b$, it has to be satisfied by the enforcement behavior (line 9). Also, if it is not expected, then it should *not* be satisfied by the enforcement behavior (line 10). In other words, there should be a one-to-one correspondence between the expected and enforcement behaviors. The rest of the behaviors follow the same pattern of reasoning (lines 11-18). If all matches are successful and no discrepancy is found between expected and enforcement behaviors, the function[3] returns *true* (line 20).

Behavior verification is summarized by the following definition:

DEFINITION 15. *A state t_i of a policy model $\mathcal{M}$ is trustworthy iff all the behaviors associated with it are trustworthy. Further, a policy model $\mathcal{M}$ is trustworthy iff all its states are trustworthy.*

[3]The algorithm provided in Figure 7 has linear time and space complexity in the size of the policy.

Based on the above definition, we conclude this section with the following theorem.

THEOREM 1. *A model $\mathcal{M}$ is trustworthy if its first state t_0 is trustworthy.*

Proof: We prove this with mathematical induction. If t_0 is trustworthy, then all behaviors associated with t_0 are trustworthy. Now, we assume that t_n is trustworthy and prove that t_{n+1} is trustworthy.

- **State Transition Behaviors:** State t_n is trustworthy, which means that $e_n b(t_n) \models_e \rightarrow_e$ holds (by Definition 13). Now, suppose if the behavior of the reference monitor responsible for making state transition decisions is measured, e.g., through signed-hash values, then it is trusted for all reachable states. Trivially, if $t_{n-1} \rightarrow_e t_n$ then $t_n \rightarrow_{e'} t_{n+1}$, which further means that if $e_n b_n(t_n) \models_e \rightarrow_e$ holds, and then $e_n b(t_{n+1}) \models_e \rightarrow_e$ also holds.

  Now, for the attributes involved in the logical expression:

  1) If there is no update in state t_n, and attributes involved in a usage control session are monitored, e.g., through attribute certificates, then they are also trustworthy in state t_{n+1}.

  2) If there is an attribute update in state t_n to a subject or object attribute a. Then, the new attribute value a' is only trustworthy iff a is trustworthy and the procedure responsible for updating is trustworthy (by Definition 9). We assume that behavior of the attribute update actions responsible for update is measured, e.g., through signed-hash method, then the behavior remains trusted for all set of states. Thus, a' is also trustworthy in state t_{n+1}.

  Thus, $e_n b(t_{n+1}) \models_e \rightarrow_e$ holds.

- **Active Subject/Object Behaviors:** State t_n is trustworthy, which means that if an ACM action is associated with state t_n, then the corresponding ACM action is also trustworthy (by Definition 15).

  1) If t_n=*requesting*, then t_{n+1} is either *accessing* or *denied*. Since the state *denied* is not within the domain of UCON ACM, we take t_{n+1}=*accessing*. Now, if subject s_1 requests for right r_1 on object o_1 in state t_n, then the ACM is trustworthy in state t_{n+1} iff $o_1 \notin O' \wedge s_1 \notin S' \wedge s_1 \notin A'(o_1, r_1)$ (by Definition 5). However, if subject s_1 is exercising r_2 on o_1, then the ACM is trustworthy iff $o_1 \in O' \wedge s_1 \in S' \wedge s_1 \in A'(o_1, r_2)$ in state t_{n+1} (by Definition 5). If these conditions hold then $e_n b(t_{n+1}) \models_a \mathcal{CR}$ holds, and correspondingly ACM is trustworthy.

  2) If t_n = *accessing*, then t_{n+1} is either *revoked* or *end*. Since both the states *end* and *revoked* have the same ACM behaviors, therefore by proving one of them is trustworthy, the ACM is trustworthy. Suppose, if s_1 is currently exercising right r_1 on o_1, then the ACM is trustworthy iff $o_1 \in O' \wedge s_1 \in S' \wedge s_1 \in A'(o_1, r_1)$ in t_n, and t_{n+1} is trustworthy iff $S' = S \ominus \{s_1\} \wedge O' = O \ominus \{o_1\} \wedge A'(o_1, r_1) = A(o_1, r_1) - \{s_1\}$ (by Definition 12). If these conditions hold, then $e_n b(t_{n+1}) \models_a \mathcal{EN}$, and $e_n b(t_{n+1}) \models_a \mathcal{RK}$ also holds.

- **Attribute Update Behaviors:** State t_n is trustworthy, which means that $e_n b(t_n) \models_{au} \mathcal{AU}_x$ holds for attribute x (by Definition 14). Suppose, if the behavior of the attribute update action responsible for updating x is measured, e.g., through signed-hash method, then the behavior remains constant for all states. On the other hand, if attribute x involved in a particular usage control session is monitored, using e.g., attribute certificate, then it means that attribute is also trusted for all states. Thus, if $e_n b(t_n) \models_{au} \mathcal{AU}_x$ holds, then $e_n b(t_{n+1}) \models_{au} \mathcal{AU}_x$ also holds.

Thus, all three behaviors are trustworthy in state t_{n+1} if they are trustworthy in state t_n which completes the proof of the theorem that a model $\mathcal{M}$ is trustworthy iff t_0 is trustworthy. □

We conclude that if a policy model is trustworthy in its initial state, a challenger can consider the target platform as trustworthy and can assume that the associated instance policy of the model will indeed be enforced.

7. CONCLUSION AND FUTURE WORK

In this paper, we have presented a high-level behavior specification and attestation framework, which can be built on top of various low-level attestation techniques. In our approach, the behavior of a model is attested rather than a hardware or software platform. We selected UCON as an example target model in this paper because UCON is an ideal model to capture continuous access control in distributed and dynamic environments. Thus, usage control is an prime candidate for remote attestation. (In Appendex A, we provide another example in the form of behavior specification of the RBAC model.)

The product of our work presented here is a high-level framework which can utilize different low-level techniques in a supporting manner for the purpose of attesting a remote platform. Behavior is associated with different components of a policy model. The behavior of each of these components can be attested separately at runtime and the aggregate of these behaviors can be used to measure the trustworthiness of the remote platform. Trust is thus associated with the dynamic behavior of a policy model instead of static measures such as hardware or software configurations or properties of the remote platform.

MBA opens a new dimension in the form of behavior identification, specification and verification of access control and usage control models. Currently, behavior specification is not automatic and requires manual intervention. A fully automated behavior specification framework will simplify behavioral attestation. The details of this aspect will be explored in our future work.

Another important aspect is the transformation of high-level behavior specifications to some low-level policies such as XACML. We conjecture that, such a transformation framework will help service providers specify their behavior expectations in the form of a concrete policy.

8. ACKNOWLEDGEMENT

This work has been supported by the Institute of Management Sciences reseach grant number RG-CS-07001 to Security Engineering Research Group, Peshawar, Pakistan.

9. REFERENCES

[1] Security-Enhanced Linux (SELinux). http://www.nsa.gov/selinux/.

[2] Trusted Computing Group (TCG). https://www.trustedcomputinggroup.org/.

[3] TCG Specification Architecture Overview v1.2, page 11-12. Technical report, TCG, April 2004.

[4] Liqun Chen, Rainer Landfermann, Hans Löhr, Markus Rohe, Ahmad-Reza Sadeghi, and Christian Stüble. A Protocol for Property-based Attestation. In *STC '06: Proceedings of the first ACM workshop on Scalable trusted computing*, pages 7–16, New York, NY, USA, 2006. ACM Press.

[5] Vivek Haldar, Deepak Chandra, and Michael Franz. Semantic remote attestation – a virtual machine directed approach to trusted computing. *In. Proc. of the Third Virtual Macine Research and Technology Symposium USENIX 2004.*

[6] Trent Jaeger, Reiner Sailer, and Umesh Shankar. PRIMA: Policy-Reduced Integrity Measurement Architecture. In *SACMAT '06: Proceedings of the eleventh ACM symposium on Access control models and technologies*, pages 19–28, New York, NY, USA, 2006. ACM Press.

[7] Xiao-Yong Li, Chang xiang Shen, and Xiao-Dong Zuo. An Efficient Attestation for Trustworthiness of Computing Platform. In *IIH-MSP*, pages 625–630, 2006.

[8] Jaehong Park and Ravi Sandhu. Towards Usage Control Models: Beyond Traditional Access Control. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 57–64, New York, NY, USA, 2002. ACM Press.

[9] Ahmad-Reza Sadeghi and Christian Stüble. Property-based Attestation for Computing Platforms: Caring about Properties, not Mechanisms. In *NSPW '04: Proceedings of the 2004 Workshop on New Security Paradigms*, pages 67–77, 2004. ACM Press.

[10] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, pages 16–16, Berkeley, CA, USA, 2004. USENIX Association.

[11] Ravi Sandhu. Rationale for the RBAC96 Family of Access Control Models. In *RBAC '95: Proceedings of the first ACM Workshop on Role-based access control*, page 9, New York, NY, USA, 1996. ACM Press.

[12] Ravi S. Sandhu. Lattice-Based Access Control Models. *Computer*, 26(11):9–19, 1993.

[13] Elaine Shi, Adrian Perrig, and Leendert Van Doorn. BIND: A Fine-Grained Attestation Service for Secure Distributed Systems. In *SP '05: Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 154–168, 2005.

[14] XACML 2.0 Specification Set. Available at: http://www.oasis-open.org/ committees/ tc_home.php? wg_abbrev=xacml.

[15] Xinwen Zhang, Francesco Parisi-Presicce, Ravi Sandhu, and Jaehong Park. Formal Model and Policy Specification of Usage Control. *ACM Trans. Inf. Syst. Secur.*, 8(4):351–387, 2005.