

Remote Attestation of Attribute Updates and Information Flows in a UCON System

Mohammad Nauman¹, Masoom Alam¹, Xinwen Zhang², and Tamleek Ali¹

¹ Security Engineering Research Group,
Institute of Management Sciences, Peshawar, Pakistan
{nauman,masoom,tamleek}@imsociences.edu.pk

² Samsung Information Systems America, San José, USA
xinwen.z@samsung.com

Abstract. UCON is a highly flexible and expressive usage control model which allows an object owner to specify detailed usage control policies to be evaluated on a remote platform. Assurance of correct enforcement is mandatory for the establishment of trust on the remote platform claiming to implement UCON. Without such an assurance, there is no way of knowing whether the policies attached to the objects will be enforced as expected. Remote attestation, an important component of Trusted Computing, is highly suitable for establishing such an assurance. Existing approaches towards remote attestation work at a very coarse-grained level and mostly only measure binary hashes of the applications on the remote platform. Solutions at this level of abstraction cannot provide assurance to a challenger regarding behavior of a remote platform concerning enforcement of the owner's policies. In this paper, we provide a new remote attestation technique which allows a challenger to verify two important behaviors of a UCON system enforcing its policies. These two behaviors are the attribute update behavior and information flow behavior. Measuring, storing and reporting these behaviors in a trusted manner is described in detail and a mechanism for the verification of these behaviors against the original UCON policies is provided. The end result is a flexible and scalable technique for establishing trust on attribute updates and information flow behaviors of a remote UCON system.

Keywords: Information flow, remote attestation, usage control, security.

1 Introduction

Usage control deals with issues concerning usage of protected objects based on the policies of the object owner. While traditional *access control* models deal with authorization issues such as who may access an object, *usage control* models address issues concerning use of objects such as duration of each use, the number of usages and ability to re-distribute etc.

UCON [1] is a highly expressive usage control model which adds *continuity* of access decisions and *mutability* of attributes at the model level. Its major

strength lies in the ability to specify elaborate usage control policies to be evaluated on a remote platform. This strength of UCON to operate on a remote platform is also a source of concern. Since the owner of the object releases it to a remote platform, she has no way of ensuring that the policies attached to it will be enforced as specified. Trusted computing [2] proposes an innovative approach for establishing trust on a remote platform in such a scenario. This approach, called Remote Attestation, allows a *challenger* to verify that the behavior of a *target* platform is trustworthy. Existing approaches towards remote attestation include low-level techniques of presenting binary hashes of executables to the challenger [3,4], middle-level approaches of mapping system configurations to generic properties by a trusted third party [5] and a high level mechanism of measuring individual components of a policy model for the establishment of trust [6]. The low- and middle-level techniques allow a challenger to statically determine the identity of the applications running on the client and properties of the system in general. They do not enable measurement of dynamic behavior of a target application on the client. Moreover, it has been widely accepted that binary hashes of executables alone are insufficient for reasoning about trustworthiness of a platform [4,7]. Low-level binary hash based techniques are, therefore, not suitable for remote attestation of a UCON system.

Consider for example, a UCON policy, which specifies that, “a media file can only be played once by an individual in the public relations office for two minutes only and that each usage has to be logged”. Clearly, it is impossible to deduce, from the hash of an executable alone, that this policy will be enforced correctly by the application.

For deducing such intricate details of an application’s behavior, Alam et al. have proposed Model-based Behavioral Attestation [6] i.e. attestation of a policy model being followed by the target application for a specific purpose. This technique proposes the decomposition of the behavior of a policy model into its individual components and measuring these individual behaviors. If the behavior of each of the components can be attested by the challenger, the whole system can be deemed as trustworthy. Model-based Behavioral Attestation has specified three behaviors of a UCON policy model – active subject/object behavior, attribute update behavior and state transition behavior. We note that the procedure for the measurement of these behaviors is not a part of the Model-based Behavioral Attestation framework.

In this paper, we specify a technique for measurement, storage and reporting of the attribute update behavior and its verification against the challenger’s policies. Attribute updates are an integral part of the UCON model and heavily influence usage decisions [8]. Successful remote attestation of attribute update behavior would provide confidence to the challenger regarding the trustworthiness of a target platform.

We also identify another UCON model behavior – information flow behavior – which captures the possible information flows between objects in a UCON system. Attesting the trustworthiness of information flow behavior would provide assurance that no illegal information flows occurred on the client end during

the usage of the owner’s resources. Such assurance is critical in systems of a distributed nature [9]. Similar to the attribute update behavior, we provide a detailed mechanism for the measurement, storage, reporting and verification of the information flow behavior.

Contributions: Our contributions in this paper are as follows: 1) We describe a mechanism for recording arbitrary data structures in the TPM as opposed to binary hashes of executables only. 2) We detail a procedure for measuring the behaviors of attribute updates and information flows in a UCON system. 3) We provide a means of verifying the behavior tokens returned by the target application on the challenger side against the original UCON policies.

Outline: The rest of the paper is organized as follows: In Section 2, we provide background about the UCON model and describe the formal model used for our attestation purposes. Behavioral Attestation is introduced in Section 3. Our UCON system attestation is described at length in Section 4 with attribute updates and information flows covered in Sections 4.1 and 4.2 respectively. Previous work related to this paper is mentioned in Section 5. Finally, we conclude our work and present future directions in Section 6.

2 UCON

UCON [1] is a Usage CONTROL model, which builds heavily on traditional access control models. It incorporates dynamic usage of protected objects and changes in decisions to allow further access to these objects as a result of usage. This extension is achieved through the introduction of two novel features: access decision continuity and attribute mutability.

In a UCON system, the user initiates a request for an object protected by the UCON system. The access decision depends on the policies and constraints for the particular *subject*, *object* and *right* combination, identified by (s, o, r) . The request can either be granted or denied. Even if the request is initially granted, the usage session does not end. The coupling of attribute mutability and access decision continuity means that due to the usage of the protected object, the attributes of the subject and/or object may change. As a result of this change, the decision to allow access might also be reversed. The usage session remains at state *accessing* as long as the constraints allow the continued use of the object. If the user ends the usage, the usage session moves to state *end*. If, however, the constraints lead to a denial of access after some time, the state moves to *revoked* and the user is no longer allowed access to the object. Figure 1 shows the states in the UCON usage session [1].

Zhang et al. [10] have formally specified the UCON model at a very abstract level. However, this formalization is not suitable for the purpose of information flow analysis and attestation of attribute updates. Instead, we use another formalization of UCON by Zhang et al. [11] which has been formulated for safety analysis of the UCON model. Safety analysis of UCON is essential for our behavior verification step. For the verification of behaviors collected on the target

platform, we create a benchmark on the challenger side, which requires the safety problem of UCON to be decidable. However, Zhang et al. [11] have shown that the safety problem in general UCON is undecidable. They have defined the safety problem of a subset of UCON which includes only authorization predicates. This subset is termed as $UCON_A$. A formal model of $UCON_A$ is defined in which a UCON system is composed of *subjects*, *objects*, *rights*, *permissions* *primitive actions* and *policies*. Sets of subjects, objects and rights are denoted by S , O and R respectively and $S \subseteq O$. A *permission* is a triple of (s, o, r) where $s \in S$, $o \in O$ and $r \in R$. An attribute a of an object o is denoted by $o.a$. The set of attributes is shared by all objects and is denoted by ATT . Attributes are mapped to their values using an assignment: $o.a = v$, where $v \in dom(a) \cup \{null\}$.

A UCON system state is a pair (O, σ) where O is the set of objects and $\sigma : O \times ATT \rightarrow dom(ATT) \cup \{null\}$ is a function which maps each attribute of each object to a value or **null**. The initial UCON state is denoted by (O_0, σ_0) .

A primitive action changes the system state. The three primitive actions in $UCON_A$ are *createObject*, *destroyObject* and *updateAttribute*. On the application of any of these actions, the state of a system is said to change from t to t' where t is the state before the application of the action and t' is the state after the action has been performed. In any given state t , the permission function ρ_t maps a pair (subject, object) to a set of rights according to their attribute values in state t .

A UCON policy consists of a name, two parameter objects (usually a subject and an object), an authorization rule and a sequence of primitive actions.

$$\begin{aligned} & policy_name(s, o) : \\ & p_1 \wedge p_2 \wedge \dots \wedge p_n \rightarrow permit(s, o, r) \\ & act_1; act_2; \dots; act_k \end{aligned}$$

If one of the primitive actions in a policy is a *createObject* action, the policy is called a creating policy. The set of policies in a system is denoted by C . Changes to a system state occur as a result of application of a policy. For two UCON system states, (O_t, σ_t) and $(O_{t'}, \sigma_{t'})$, $t \rightarrow_c t'$ denotes that there exists a pair of objects (o_1, o_2) where $o_1 \in O_t$ such that policy $c(o_1, o_2)$ can be applied to t and changes the state to t' . Moreover, $t \rightarrow_C t'$ if $\exists c \in C. t \rightarrow_c t'$ and $t \rightsquigarrow_C t'$ if there

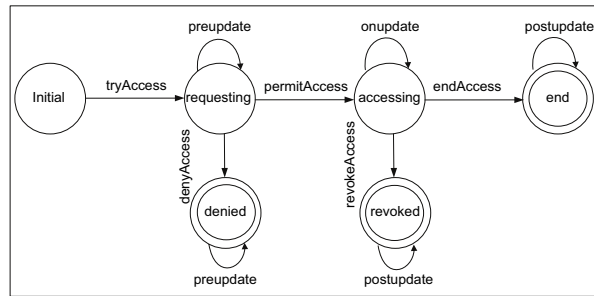


Fig. 1. The UCON Model States

exists a sequence of states t_1, t_2, \dots, t_n such that $t \twoheadrightarrow_C t_1 \twoheadrightarrow_C t_2 \dots \twoheadrightarrow_C t_n \twoheadrightarrow_C t'$. $t \rightsquigarrow_C t'$ or simply $t \rightsquigarrow t'$ is called the *transition history* from t to t' .

Zhang et al. have proven that the safety problem in this general model of UCON_A is undecidable [11]. In order to render the safety problem decidable, Zhang et al. propose some restrictions on the system. Different structures have been defined in the form of *ground policies*, *attribute update graph* and *attribute creation graph* to formalize these restrictions.

A set of *ground policies* generated from a UCON policy ‘ c ’ denotes all the evaluations of the policy c with possible attribute tuples of the object parameters which satisfy the predicates in the authorization rule of c . Assume $ATT = \{a\}$ and $\text{dom}(a) = \{1, 2, 3\}$ and the following UCON policy:

$$\begin{aligned} c(s, o) : \\ & s.a > o.a \rightarrow \text{permit}(s, o, r) \\ & \text{updateAttribute } o : o.a = o.a + 1 \end{aligned}$$

Grounding this policy generates the following three *ground policies*:

$$\begin{aligned} c(s : (a = 3), o : (a = 2)) : \\ & \text{true} \rightarrow \text{permit}(s, o, r) \\ & \text{updateAttributeTuple } o : (a = 2) \rightarrow (a = 3) \end{aligned}$$

$$\begin{aligned} c(s : (a = 3), o : (a = 1)) : \\ & \text{true} \rightarrow \text{permit}(s, o, r) \\ & \text{updateAttributeTuple } o : (a = 1) \rightarrow (a = 2) \end{aligned}$$

$$\begin{aligned} c(s : (a = 2), o : (a = 1)) : \\ & \text{true} \rightarrow \text{permit}(s, o, r) \\ & \text{updateAttributeTuple } o : (a = 1) \rightarrow (a = 2) \end{aligned}$$

Note that for attribute tuples for which the predicate is not true (e.g. $s : (a = 1), o : (a = 1)$), no ground policy is generated.

A *create ground policy* is a ground policy, which contains a *createObject* action in its body. In such a policy, the attribute tuple of the first parameter object is termed as *create-parent attribute tuple* and that of the second is termed as *create-child attribute tuple*. An *Attribute Creation Graph (ACG)* is a directed graph with nodes all possible attribute tuples and an edge from create-parent attribute tuple to a create-child attribute tuple if there exists a corresponding ground policy for these tuples.

Similarly, in a ground policy which updates an attribute tuple, the old attribute tuple is called the *update-parent attribute tuple* and the updated tuple is called the *update-child attribute tuple*. An *Attribute Update Graph (AUG)* is a directed graph with nodes all possible attribute tuples and edges from update-parent attribute tuple to update-child attribute tuple if there exists a corresponding ground policy for these tuples.

Using these structures, Zhang et al. [11] have shown that a $UCON_A$ system with finite attribute domains is decidable if the ACG is acyclic, the AUG has no cycles containing a create-parent attribute tuple and in each creating ground policy, the attribute tuples of both the parent and child are updated. Usefulness of $UCON_A$ systems with these restrictions has been shown. For a detailed discussion and proofs of these statements, we refer the reader to [11].

In the next sections, we describe how a $UCON_A$ system with these restrictions can be remotely attested using dynamic behaviors of the system recorded during enforcement of policies.

3 Behavioral Attestation

Traditional attestation techniques [3,4,5] rely solely on the binary hashes of applications running on the client. A chain of trust is established from the core root of trust (i.e. the Trusted Platform Module) to the application. However, all of these techniques measure the target application statically without considering its inner working [6]. A recent technique, Model-based Behavioral Attestation (MBA) [6], proposes a high-level framework for measuring the internal working of the target application based on the dynamic behaviors of the different components of the application. We note that the MBA framework relies on the existence of a small *monitor* module in the target application as part of the Trusted Computing Base (TCB)¹.

The monitor, being a part of the TCB, can measure the dynamic behavior of the rest of the application in a trusted manner. During an attestation request, the monitor sends these measurements to the challenger where they can be verified. If the behavior depicted by these measurements is compliant with the object owner's policy, the challenger can be assured that the security policy is indeed being enforced as expected. For the dynamic behaviors reported by the monitor to be trusted, there are two requirements.

1. The monitor module has to be verified for correctness using formal methods. While formal verification of large systems is a complex procedure and quickly becomes infeasible [13], verification of small components is easier and can yield many benefits. The monitor is a relatively small component and its formal verification adds significantly to the confidence in the correctness of the functionality and subsequently to its reported measurements.
2. Its hash has to be attested using traditional attestation techniques such as IMA [3] or PRIMA [4]. In other words, this dynamic attestation technique is not exclusive of traditional attestation mechanisms but supports them by providing an added level of confidence through attestation of internal working of the application and its dynamic behavior.

The rest of the paper describes details of implementation of this monitor in a target application enforcing UCON policies. We discuss the measurements to

¹ TCB is the collection of software and hardware components which are responsible for enforcing security policies on a platform [12].

be made for the dynamic behavior of attribute updates and information flows in the application and the mechanism for reporting these changes to the challenger in a trusted manner. We also describe how the reported behavior can be verified against the challenger’s policy to ensure that the information flows and attribute updates in the target UCON application are occurring as expected.

4 UCON System Attestation

UCON is primarily concerned with usage of an object after it is released to a remote platform. The owner of the object may not have control over the usage of the object. It is therefore imperative that she be able to establish trust on the remote platform. Without the assurance of trustworthiness of the UCON system on the remote platform, there is no way of ensuring that the UCON policy attached to the object will indeed be enforced as expected [6].

We focus on two aspects of a UCON system implementation in this contribution:

1. Attributes play an important role in a UCON system. Attribute mutability is a core feature of UCON which lends the model its flexibility and expressive power. The challenger needs to be able to verify remotely that attribute updates occurring on the client are compliant with the policies.
2. To ensure that no information leakage can occur, the challenger needs a mechanism for remotely attesting possible information flows on the target. Information flows not allowed by the policies of the challenger may lead to a leakage of information to unauthorized parties. By having the client report all possible information flow to the challenger in a trustworthy manner, possible information leakage can be successfully detected.²

To formulate a framework for these two requirements, we define two *behaviors*. The first requirement is captured by the *attribute update behavior* (\mathcal{AU}) and the second is captured by the *information flow behavior* (\mathcal{IF}). Each of these behaviors is monitored by the Behavior Manager (BM) which is a part of the UCON engine on the client end (cf. Section 3). The BM captures dynamic behavior of attribute updates and possible information flows and is capable of communicating these behaviors to the challenger in a trustworthy manner.

Figure 2 depicts the architecture of remote attestation of a UCON system. When a target application on the client requests an object, the server, upon successful authorization of the client, attaches a UCON policy to the object and releases the *protected object* to the client. The object is registered with the UCON decision engine on the client. During the usage of the object, usage authorization decisions and any updates which need to be performed are communicated to the Behavior Manager. The Attribute Update Manager records proofs for the attribute update behavior \mathcal{AU} and the Information Flow Manager records proofs for the information flow behavior \mathcal{IF} . During an attestation challenge,

² In this contribution, we focus on explicit information flows. Implicit information flows, such as those through covert channels, are not addressed.

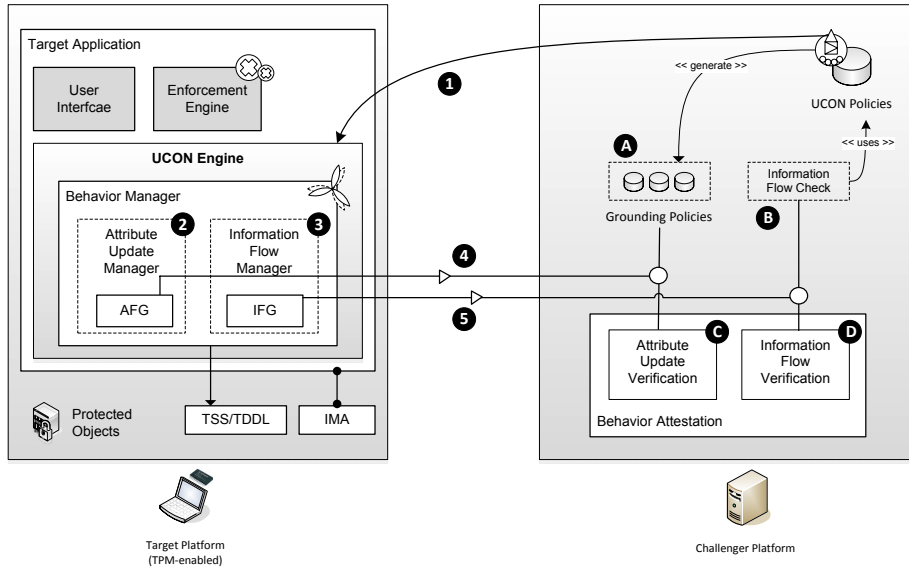


Fig. 2. Remote Attestation of a UCON System

the Behavior Manager collects these behavior proofs and reports them to the challenger.

Upon receipt of these two behaviors, the challenger performs two behavior verification procedures. The attribute update verification module utilizes the original UCON policies to generate ground policies (cf. Section 2) which are used to verify the trustworthiness of the attribute update behavior (cf. Section 4.1). The same set of UCON policies are utilized by the information flow attestation mechanism to verify the information flow behavior using an information flow check algorithm (cf. Section 4.2).

The details of storing and reporting the two behaviors in a trusted manner on the target platform and the verification mechanisms utilized on the challenger side are described below.

4.1 Attribute Update Behavior

For capturing the attribute update behavior, the BM implements one or more *Attribute Update Procedures (AUPs)* which are responsible for updating attributes on the client end. The procedures take two inputs, either of which can be an attribute of an object or a constant.

Individual calls to an attribute update procedure and subsequent attribute updates are recorded through a graph structure called the *attribute flow graph*. This structure stores the relationship between updated attributes and the attributes used as inputs for this updation. Formally:

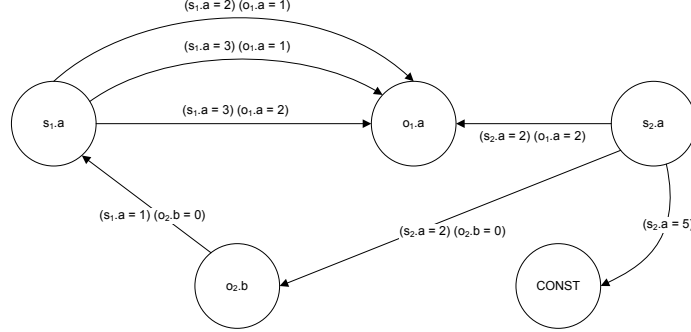


Fig. 3. Attribute Flow Graph Example

Definition 1 (Attribute Flow Graph). *The Attribute Flow Graph (AFG) is a directed multi-graph (G, V) where G is a set of nodes representing object attributes or a constant and V is a set of edges representing attribute updates. An edge directed from $o_i.a$ to $o_j.b$ denotes an update of $o_i.a$ and is labeled with $(o_i.a = \text{dom}(ATT)), (o_j.b = \text{dom}(ATT))$. The label captures the values of $o_i.a$ and $o_j.b$ before the update takes place. The special node called $CONST$ is used to denote all constants.*

Figure 3 shows a graphical representation of the AFG. Note that there may be more than one attribute updates involving the same set of object attributes but with different values. An attribute update involving a constant is represented by an edge from the target attribute to the $CONST$ node.

To capture the AFG in a trustworthy manner, we employ the constructs of Trusted Computing. The initial value of the the AFG (i.e. null) and any subsequent changes to it are stored in an *Attribute Update Log (AUL)*. At startup, the BM initializes the AUL with an initialization token $INIT$. It monitors all calls to the *AUPs* and whenever a call is received, it creates an *entry* in the AUL. Any change to the AUL is stored in a Platform Configuration Register (PCR) of the TPM by taking a hash of the entry and extending the PCR through *pcr_extend* (cf. Figure 4). The hash of the update procedure (AUP_x) responsible for performing the specific update is also recorded in the PCR through *pcr_extend*. The new value of the PCR after an update is calculated as:

$$PCR_{AUL_\epsilon} = \text{SHA-1}(\text{SHA-1}(PCR_{AUL_{\epsilon-1}} \parallel \text{SHA-1}(AUL_\epsilon)) \parallel \text{SHA-1}(AUP_x))$$

During an attestation challenge, the BM receives a nonce from the challenger and submits the nonce to the TPM through a Trusted Software Stack (TSS) [14,15,16]. It requests the TPM to perform a quote over the given PCR and nonce. The quoted value of the PCR is sent to the challenger along with the AUL for verification.³

³ The interested reader may refer to [17] for a detailed description of the quote operation over a PCR.

```

INIT // initialize the AUL
s1.a:o1.a:s1.a=2:o1.a=1::AUP1 // update by AUP1 involving s1.a and o1.a
s1.a:o2.b:s1.a=1:o2.b=0::AUP2 // update by AUP2 involving s1.a and o2.b
s1.a:o1.a:s1.a=3:o1.a=1::AUP1 // ...
s2.a:CONST:s2.a=5::AUPY // update by AUPY involving s2.a and a constant
s1.a:o1.a:s1.a=2:o1.a=2::AUPX // ...
s2.a:o1.a:s2.a=2:o1.a=2::AUP2 // ...
s2.a:o2.b:s2.a=2:o2.b=0::AUP1 // ...

```

Fig. 4. Sample Attribute Update Log

Capturing the dynamic behavior of updates is a relatively simple task. Once the attribute update log is received by the challenger, it has to be verified against the policy to ensure that all attribute updates occurring on the client comply with the policies of the challenger. The challenger utilizes the grounding procedure, defined in Section 2, for this compliance checking.

In order for the attribute updates occurring on the client to be considered as trustworthy, the challenger needs to be able to verify that, for each update, there exists a ground policy (generated as a result of grounding of the policies sent to the client), which requires the update performed at the client end. It also requires the hash of the update procedure responsible for performing the update to be trusted. The first step for the verification of attribute update behavior is the verification of the signature performed by the client's TPM on the PCR value. This ensures that the PCR values can be trusted to be signed by a genuine TPM and not by a software masquerading as a TPM. The second step is to verify the Attribute Update Log (AUL) against the PCR value returned. This is a similar operation to the verification procedure used by the Integrity Measurement Architecture [3]. Hashes of entries in the AUL and those of the update procedures are concatenated in sequence to give the final value of the PCR. For each entry AUL_ϵ in the AUL, the PCR value at AUL_ϵ is given by:

$$PCR_{AUL_\epsilon} = \text{SHA-1}(\text{SHA-1}(PCR_{AUL_{\epsilon-1}} \parallel \text{SHA-1}(AUL_\epsilon)) \parallel \text{SHA-1}(AUP_x))$$

where AUP_x is the procedure performing the update recorded in AUL_ϵ . If the final value of the computation matches the value of the PCR returned by the target's TPM, the challenger can be assured that the AUL has not been tampered with and can be used for verification of the target's behavior.

The next step in the attribute update behavior verification is to verify each attribute update operation against the ground policies to ensure that no illegal attribute updates have occurred on the target platform and that the hash of the update procedure responsible for performing the updates is a known good one. For each attribute update, represented by edges in the AFG, there must exist a ground policy which updates the target (object, attribute) pair using the source (object, attribute) pair in the AFG. Attribute updates involving constants must be verified against the *CONST* node against the values required by the ground policies. Formally:

$$\begin{aligned}
&\forall v \in V. \exists c_n \in C_n. \exists uo \in c_n. \text{target}(uo) = \text{target}(v) \\
&\quad \wedge \exists s \in \text{sources}(uo). s = \text{source}(v) \\
&\quad \wedge \forall o.a \in v. \text{value}(o.a) = \text{avalue}(o.a, uo)
\end{aligned}$$

where uo is an update operation in a ground policy c_n , $target(uo)$ is the output of the update operation, $sources(uo)$ are the inputs to the update operation uo , $value(o.a)$ returns the value of the attribute a of object o during the update procedure and $avalue(o.a, uo)$ returns the attribute value of $o.a$ from the attribute tuple of uo .

If the above condition is satisfied by the complete AFG, the challenger can be assured that all attribute update operations performed on the client have been in compliance with the UCON policies. We define the trustworthiness of the attribute update behavior \mathcal{AU} as:

$$\begin{aligned} \mathcal{AU}.behavior = \text{trusted} \text{ iff} \\ \forall v \in V. \exists c_n \in C_n. \exists uo \in c_n. target(uo) = target(v) \\ \wedge \exists s \in sources(uo). s = source(v) \\ \wedge \forall o.a \in v. value(o.a) = avalue(o.a, uo) \\ \wedge \forall a \in AUP_x. a.behavior = \text{trusted} \end{aligned}$$

In essence, attribute update behavior is trusted if and only if 1) all attribute updates taking place on the target machine are allowed by some ground policies generated from the original usage policies of the challenger and 2) all the procedures responsible for performing attribute updates on the target are also trusted.

4.2 Information Flow Behavior

For the measurement of the information flow behavior, the Behavior Manager utilizes an Information Flow Manager. This component of the UCON implementation is responsible for maintaining a structure called the Access Rights Graph (ARG). The ARG records information about which objects have been granted access rights to other objects. Formally:

Definition 2 (Access Rights Graph). *An Access Rights Graph (ARG) is a directed graph (H, W) where H is a set of nodes representing the objects and W is a set of edges representing rights. An edge from h_1 to h_2 labeled r denotes the rights r assigned to h_1 on h_2 at some point in the usage history where $h_1, h_2 \in H$, $r \in 2^R$ and R is the set of rights.*

Figure 5 shows a graphical depiction of the ARG. To store and later report this structure in a trusted manner, the BM utilizes a technique similar to that used for capturing the AFG. The initial (empty) value of the ARG is stored in a *Access Rights Log (ARL)*. The ARL is initialized as empty by setting it to the value *INIT*. Any decisions by the UCON decision module are captured by the ARL. If an access is granted to a subject s on an object o for right r , nodes s and o are added to the ARG, if they are not already present. An entry is made in the ARL for recording the addition of the nodes. An edge, directed from s

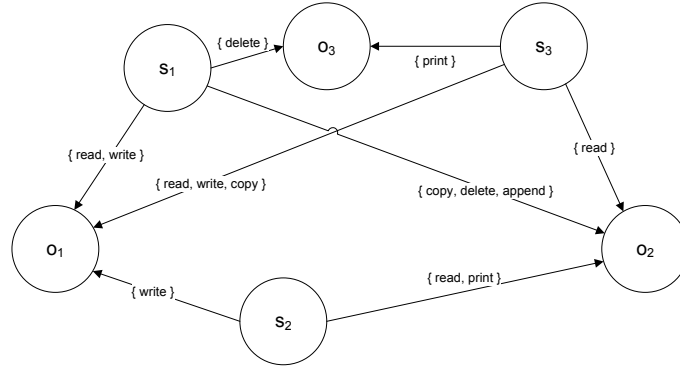


Fig. 5. Access Rights Graph Example

to o is added to the ARG and labeled $\{r\}$ if such an edge doesn't already exist. If the edge already exists, right r is added to the set of rights on the edge. An entry is made in the ARL corresponding to the addition of the right for s on o . Figure 6 shows an example ARL created as a result of different usage decisions.

Whenever an entry is appended to the ARL, its hash is calculated by the Information Flow Manager and stored in the PCR through *pcr_extend*. During an attestation challenge, the ARL and this PCR value is returned to the challenger where verification of these structures against the challenger's UCON policies takes place.

For the verification of the ARL on the challenger side, we utilize an *information flow check* algorithm which utilizes the same semantics as the UCON *safety check* algorithm presented by Zhang et al. [10]. The first step, as in the verification of the ARL, is to verify the signature by the TPM to ensure that the PCR values returned are from a genuine TPM and that the ARL is trusted. Every entry ARL_ϵ in the ARL is concatenated in sequence to give the final value of the PCR as:

$$PCR_{ARL_\epsilon} = \text{SHA-1}(PCR_{ARL_{\epsilon-1}} \parallel \text{SHA-1}(ARL_\epsilon))$$

Verification of the entries in the ARL provides assurance to the challenger that the ARL has not been tampered with. The individual entries in the ARL are

INIT	// initialize the ARL
ADD s1	// add a new subject s1
ADD o3	// add a new object o3
ASSIGN s1:o3:delete	// assign right delete to s1 on o3
ADD o1	// ...
ASSIGN s1:o1:read	// ...
ADD o2	// ...
ASSIGN s1:o2:append	// ...
...	

Fig. 6. Sample Access Rights Log



Fig. 7. Information Flow Graph corresponding to ARG in Figure 5

used to re-generate the ARG on the challenger side. After this re-generation, the challenger creates an Information Flow Graph (IFG). The IFG depicts possible information flows implied by the Access Rights Graph. Formally:

Definition 3 (Information Flow Graph). *An Information Flow Graph (IFG) is a directed graph (I, U) where I is a set of nodes representing the objects and U is a set of edges representing possible information flow in the direction of the edge. An edge from i_1 to i_2 denotes that information may have flown from i_1 to i_2 where $i_1, i_2 \in I$.*

To construct the IFG from the ARG, we first define all rights as *read-like*, *write-like*, *read-write-like* or *no-impact* [18]. No-impact operations are those which cannot play a part in information flow (such as *print*) and are discarded immediately. Afterwards, all objects in the ARG are represented in the IFG. For each subject s in the ARG, an edge is created from o_1 to o_2 if a read-like (or read-write-like) operation is granted to s on o_1 and a write-like (or read-write-like) operation is granted to s on o_2 . Afterwards, orphan nodes are removed from the IFG. Figure 7 shows an IFG corresponding to the ARG of Figure 5.

For the verification of possible information flows as depicted by the IFG the following procedure is adopted. For each edge on the IFG, Algorithm 1 is applied to ensure that the information flow is compliant with the policies of the challenger. The algorithm takes an initial UCON state and a set of ground policies as inputs. A finite automaton (FA) is created which maps changes to the UCON state as a result of applying non-creating ground policies (line 2). For each state in the resulting FA, a few operations are performed. First, all subjects which have been assigned a read-like (or read-write-like) right on o_1 are added to the set *reading* (lines 4,6). If one of the subjects was previously granted a write-like operation on o_2 , the algorithm immediately returns *true* (line 7) as the subject would have been able to cause information to flow from o_1 to o_2 .

A similar procedure is followed for write-like operations (line 9). All subjects which have been assigned a write-like operation (or read-write-like operation) on o_2 are added to the set *writing* (lines 9,11) and if one of them was previously assigned a read-like operation on o_1 , the algorithm returns *true* immediately (line 12).⁴

Finally, creating ground policies are applied (line 15) to extend the UCON system with new objects and `InfoFlowCheck()` algorithm is called recursively (line 20) to check for possible information flows in this expanded space.

⁴ Note that the algorithm only checks for information flow *from* o_1 to o_2 and not in the other direction.

Algorithm 1. Information Flow Check Algorithm

Input: $UCON_A$ system with initial state $t_0 = (O_0, \sigma_0)$, a finite set of ground policies and two objects, o_1 and o_2 **Output:** A boolean value which is true only if information can flow from o_1 to o_2

```

1) InfoFlowCheck( $O_0, t_0$ )
2) Construct a finite state automaton  $\mathcal{FA}$  with objects  $O_0$  and the set of non-creating
   ground policies as in [11]
3) foreach  $t_0 \rightsquigarrow t \in \mathcal{FA}$  do
4)   collect  $\varsigma = \{x | r \in \rho_t(x, o_1) \wedge r = \text{'read'}\}$ 
5)   foreach  $s \in \varsigma$  do
6)      $reading := reading \cup \{s\}$ 
       // maintain a set of subjects which have been allowed to read from  $o_1$ 
7)     if  $s \in writing$  return true;
8)   end for
9)   collect  $\varsigma = \{x | r \in \rho_t(x, o_2) \wedge r = \text{'write'}\}$ 
10)  foreach  $s \in \varsigma$  do
11)     $writing := writing \cup \{s\}$ 
       // maintain a set of subjects which have been allowed to write to  $o_2$ 
12)    if  $s \in reading$  return true;
13)  end for
14)  foreach subject  $s$  in  $t$  do
15)    foreach creating ground policy  $c(s : \tau_s, o : \tau_o)$ , where  $\tau_s(a) = \sigma_s(o.a)$  do
16)      enforce  $c(s : \tau_s, o : \tau_o)$ ;
17)      create object  $o$  and update its attribute tuple to  $\tau'_o$ ;
18)      update  $s$ 's attribute tuple to  $\tau'_s$ ;
19)      the system state changes to  $t'$  with new object  $o$  and update attributes of
        $s$  and  $o$ ;
20)       $InfoFlowCheck(O_0 \cup \{o\}, t')$ 
21)    end for
22)  end for
23) end for

```

The trustworthiness of the information flow behavior \mathcal{IF} is defined as:

$$\mathcal{IF}.behavior = \text{trusted iff } \forall u \in U. InfoFlowCheck(u) = \text{true}$$

where U is the set of edges in the IFG.

Concisely, information flow behavior is trusted if and only if all possible paths of information flow on the client comply with the challenger's usage policies.

5 Related Work

One of the earliest and most significant works analyzing information flow models is by Denning [19] in which mechanisms for information flow are formalized using a lattice structure of labels and classes of objects. JFlow [20] is a security-typed language providing "mostly-static" information flow control by assigning labels to objects within the source code and ensuring that information flows

comply with the security policy of the programmer. JFlow relies on a specialized compiler and information flow controlling virtual machine for enforcement of information flow control. Haldar et al. [21] have devised a mechanism for implementing mandatory access control (MAC) mechanisms in virtual machines for controlling information flows. They propose the use of run-time policy enforcement as opposed to the mostly-static compile time checks [20] for enforcing MAC policies. Nair et al. [22] have presented an information flow control system which addresses the issue of implicit information flows. The resulting framework is capable of dynamically assigning labels to objects and propagating these labels based on information and control flow.

All of these models and mechanisms address either information flow control or audit but do not deal with remote attestation of information flows, the underlying environment or the target application. However, as can be seen, some of them deal with implicit information flows as well as explicit ones and can, therefore, help in future extensions of this work.

From the aspect of remote attestation, several works have been proposed. These include the Integrity Measurement Architecture [3], which allows a remote party to verify the trustworthiness of a target platform based on the load-time integrity of binaries on the target platform. Policy Reduced Integrity Measurement Architecture (PRIMA) [4] targets a specific application by analyzing the information flow to and from the target application but still does not address internal structures and semantics of the application. LKIM [7] is one of the few approaches, which target the dynamic behavior of a system. It verifies the integrity of a Linux kernel by measuring and reporting the target's dynamic state [23]. It has been shown to detect malicious code, which could not be detected using hashes of static code.

Gu et al. [24] have described a new approach for measuring the behavior of an application using static analysis of the source code and verification of *program execution* against this benchmark.

The attestation technique described in our contribution is significantly different from both these approaches in two respects. Firstly, we utilize the TPM hardware for *trusted* storage and reporting of measurements. Secondly, our approach utilizes the owner's *policies* for the creation of a baseline. This allows for the integrity verification of a specific application for a particular purpose, thus greatly reducing the complexity of attestation.

Semantic Remote Attestation [25] is closest to the approach described in this paper. It proposes the use of a Trusted Virtual Machine, which is established as trusted and is then expected to enforce the policies at the VM level. However, trust on the correct enforcement of the policies is implied and no mechanism for measuring the correctness of the enforcement is provided. Our technique builds on this approach and describes a detailed architecture for using run-time measurements of the behavior in a trusted manner for dynamic behavioral attestation of a target application. To the best of our knowledge, no work has been done for the attestation of attribute updates and information flows in a UCON system at this level of detail.

6 Conclusion and Future Work

Remote attestation is an integral part of Trusted Computing. It allows a challenger to establish the trustworthiness of a remote platform depending on its behavior. Recent advances in remote attestation have led us to believe that measuring the hashes of executables on the remote platform is insufficient for the establishment of trust. It is necessary to verify the dynamic behavior and internal functioning of a target application. In this paper, we have proposed a mechanism for attesting the dynamic behavior of UCON – a highly expressive usage control model. Two important aspects of UCON, attribute updates and information flow, have been described. We have presented details regarding measurement, storage and verification of these behaviors in a trustworthy manner. The model of UCON under consideration is UCON_A with certain restrictions, which has previously been shown to be useful in practical scenarios. Establishment of trust on a remote party implementing this model will provide confidence to the challenger that her policies will indeed be enforced on the remote end as dictated.

This paper has introduced the novel concept and semantics of using a small ‘behavior manager’ component on the remote platform for collecting trust tokens used during attestation. This concept has been applied to collect and verify attribute update behavior and information flow behavior. The same technique can, with slight modifications, be applied for collecting various other types of trust tokens, such as information flows to and from other applications, system calls and input/output to storage devices, for an even more detailed inspection of the dynamic behavior of the target application. These and other behaviors are being considered for attestation of UCON and even generalized applications not following the UCON model. These form the basis of ongoing work in this research.

Acknowledgements

This research work has been supported by Grant No. ICTRDF/TR&D/2008/45 from the National ICT R&D Fund, Pakistan to Security Engineering Research Group, Institute of Management Sciences, Peshawar.

References

1. Park, J., Sandhu, R.: Towards Usage Control Models: Beyond Traditional Access Control. In: SACMAT 2002: Proceedings of the seventh ACM Symposium on Access Control Models and Technologies, pp. 57–64. ACM Press, New York (2002)
2. Trusted Computing Group, <http://www.trustedcomputinggroup.org/>
3. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and Implementation of a TCG-based Integrity Measurement Architecture. In: SSYM 2004: Proceedings of the 13th conference on USENIX Security Symposium, Berkeley, CA, USA, USENIX Association (2004)

4. Jaeger, T., Sailer, R., Shankar, U.: PRIMA: Policy-Reduced Integrity Measurement Architecture. In: SACMAT 2006: Proceedings of the eleventh ACM Symposium on Access Control Models and Technologies, pp. 19–28. ACM Press, New York (2006)
5. Sadeghi, A.R., Stübke, C.: Property-based Attestation for Computing Platforms: Caring about Properties, not Mechanisms. In: NSPW 2004: Proceedings of the 2004 Workshop on New Security Paradigms, pp. 67–77. ACM Press, New York (2004)
6. Alam, M., Zhang, X., Nauman, M., Ali, T., Seifert, J.P.: Model-based Behavioral Attestation. In: SACMAT 2008: Proceedings of the thirteenth ACM symposium on Access control models and technologies. ACM Press, New York (2008)
7. Loscocco, P.A., Wilson, P.W., Pendergrass, J.A., McDonell, C.D.: Linux Kernel Integrity Measurement Using Contextual Inspection. In: STC 2007: Proceedings of the 2007 ACM Workshop on Scalable Trusted Computing, pp. 21–29. ACM, New York (2007)
8. Zhang, X., Nakae, M., Covington, M.J., Sandhu, R.S.: Toward a Usage-Based Security Framework for Collaborative Computing Systems. *ACM Trans. Inf. Syst. Secur.* 11(1) (2008)
9. Srivatsa, M., Balfe, S.: Trust Management For Secure Information Flows. In: CCS 2008: Proceedings of the 15th ACM Conference on Computer and Communications Security, pp. 175–187. ACM, New York (2008)
10. Zhang, X., Parisi-Presicce, F., Sandhu, R., Park, J.: Formal Model and Policy Specification of Usage Control. *ACM Trans. Inf. Syst. Secur.* 8(4), 351–387 (2005)
11. Zhang, X., Sandhu, R., Parisi-Presicce, F.: Safety Analysis of Usage Control Authorization Models. In: ASIACCS 2006: Proceedings of the 2006 ACM Symposium on Information, computer and communications security, pp. 243–254. ACM, New York (2006)
12. Kanerva, P.: Anonymous Authorization in Networked Systems: An Implementation of Physical Access Control System. Masters Thesis. Helsinki University of Technology (March 2001)
13. Bella, G., Paulson, L.C., Massacci, F.: The Verification of an Industrial Payment Protocol: the SET Purchase Phase. In: CCS 2002: Proceedings of the 9th ACM Conference on Computer and Communications Security, pp. 12–20. ACM, New York (2002)
14. TCG Software Stack (TSS) Specifications,
<https://www.trustedcomputinggroup.org/specs/TSS/>
15. Trusted Computing for the Java(tm) Platform,
<http://trustedjava.sourceforge.net/>
16. Java Community Process. JSR321: Trusted Computing API for Java,
<http://jcp.org/en/jsr/detail?id=321>
17. Alam, M., Zhang, X., Nauman, M., Ali, T.: Behavioral Attestation for Web Services (BA4WS). In: SWS 2008: Proceedings of the ACM Workshop on Secure Web Services (SWS) located at 15th ACM Conference on Computer and Communications Security (CCS-15). ACM Press, New York (2008)
18. Guttman, J.: Verifying Information Flow Goals in Security-Enhanced Linux. *Journal of Computer Security* 13(1), 115–134 (2005)
19. Denning, D.E., Denning, P.J.: Certification of programs for secure information flow. *Commun. ACM* 20(7), 504–513 (1977)
20. Myers, A.C.: JFlow: Practical Mostly-static Information Flow Control. In: POPL 1999: Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 228–241. ACM, New York (1999)

21. Haldar, V., Chandra, D., Franz, M.: Practical, Dynamic Information-flow for Virtual Machines, www.vivekhaldar.com/pubs/plid2005.pdf
22. Nair, S., Simpson, P., Crispo, B., Tanenbaum, A.: A Virtual Machine Based Information Flow Control System for Policy Enforcement. *Electronic Notes in Theoretical Computer Science* 197(1), 3–16 (2008)
23. Thober, M., Pendergrass, J.A., McDonell, C.D.: Improving Coherency of Runtime Integrity Measurement. In: *STC 2008: Proceedings of the 2008 ACM Workshop on Scalable Trusted Computing*. ACM, New York (2008)
24. Gu, L., Ding, X., Deng, R., Xie, B., Mei, H.: Remote Attestation on Program Execution. In: *STC 2008: Proceedings of the 2008 ACM Workshop on Scalable Trusted Computing*. ACM, New York (2008)
25. Haldar, V., Chandra, D., Franz, M.: Semantic Remote Attestation – A Virtual Machine directed approach to Trusted Computing In. *Proc. of the Third Virtual Machine Research and Technology Symposium USENIX (2004)*