

A Posteriori Compliance Control *

Sandro Etalle
University of Twente
sandro.etalles@utwente.nl

William H. Winsborough
University of Texas at San Antonio
wwinsborough@acm.org

ABSTRACT

While preventative policy enforcement mechanisms can provide theoretical guarantees that policy is correctly enforced, they have limitations in practice. They are inflexible when unanticipated circumstances arise, and most are either inflexible with respect to the policies they can enforce or incapable of continuing to enforce policies on data objects as they move from one system to another. In this paper we propose an approach to enforcing policies not by preventing unauthorized use, but rather by deterring it. We believe this approach is complementary to preventative policy enforcement. We call our approach APPLE for A-Posteriori PoLicy Enforcement. We introduce APPLE Core, a logical framework for using logs to verify that actions taken by the system were authorized. A trust management system is used to ensure that data objects are provided only to users operating on auditable systems who are subject to penalty should they be found in violation. This combination of audit and accountability provides a deterrence that strongly encourages trustworthy behavior, thereby allowing a high level of assurance of end-to-end policy enforcement.

Categories and Subject Descriptors

K.6.5 [Management of Computing and Information Systems]: Security and Protection; D.4.6 [Operating Systems]: Security and Protection—*Access Controls Information Flow Controls*

General Terms

Security, Theory

Keywords

access control, trust management, policy enforcement

*This work was partially done during the first author's stay at the University of Trento (Italy) and was supported by the BSIK Freeband project I-Share, by the EU project SERENTY and by NSF awards CCR-0325951 and CCF-0524010.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SACMAT'07, June 20-22, 2007, Sophia Antipolis, France.
Copyright 2007 ACM 978-1-59593-745-2/07/0006 ...\$5.00.

1. INTRODUCTION

The problem of *policy enforcement*—guaranteeing that data is used according to established policies and is not disclosed to or corrupted by unauthorized users—is paramount to our IT dominated world. In particular, in inter-organizational cooperation, collaborating organizations often need to protect their intellectual property by enforcing policies that capture objectives such as “*this document may be seen and modified only by senior engineers working on project X*”

The approaches to enforcement of such policies most commonly used are *preventative*, in the sense that unauthorized actions are prevented from occurring. By far the most widely used method is *access control* (AC), wherein unauthorized access is prevented. There has been a very large amount of attention paid to AC (to scratch the surface, see [15, 23, 21, 5] and the survey by Abadi [2]). Another approach that is emerging is by *digital rights management* (DRM) [13, 12, 14, 24].

By their nature, when correctly deployed, preventative approaches provide guarantees that policies *as stated* will not be violated. Unanticipated circumstances (which in collaborative environments are rather common) cannot be met by circumventing preventative controls. The techniques we develop in this paper offer a very different interpretation of “policy enforcement,” by which we intend a usage of “enforcement” somewhat akin to that in “law enforcement,” and that can be employed in a manner complementary to preventative means. Preventative enforcement should be used only to prevent things that most assuredly must never happen. Based on deterrence, the approach we propose can be used to proscribe actions that are *normally* considered wrong. In it, unauthorized actions can be performed, but the perpetrators can be held accountable for such actions. Although in many environments it may not be possible to provide the level of deterrence required to make our approach workable, in many other environments, the deterrence provided by such accountability against violating policy can be very strong¹. In these environments, our approach can considerably augment the level of assurance provided by other methods, and it can be used with policies that are sufficiently strict that you would not want to make violating them utterly impossible. When unexpected, extenuating circumstances arise, system participants can rely

¹In the late '90's, a high-level IT manager at the New York Stock Exchange told one of the authors that access control on the trading floor was unnecessary because of auditing and the high opportunity cost of losing one's trader job.

on their own judgment as to whether an exception to standard policy should be made. To illustrate the principle with an example from the physical world, under normal circumstances, a school bus driver commandeering a bus for his own purposes was almost certainly committing a crime in New Orleans. However, doing so to help a neighborhood evacuate as Hurricane Katrina approached would probably have been recognized as heroic.

The setting we consider here is that of a number of organizations temporarily sharing resources and information on a collaborative project. The IT infrastructure of these organizations is heterogeneous: in particular their AC systems may be pairwise incompatible. Let us state the main requirements that the collaborative environments we aim to support impose on policy enforcement system.

1. Policy enforcement must be *end-to-end*. This means that information in documents is protected from the time it is entered into the system until the point at which it is deleted, no matter where the information may flow in the mean time. So when a document is released to a remote host or domain, the policy that governs it remains associated with it and enforced.
2. Users should be able to create, modify and merge documents using the programs they are accustomed to use for these purposes.
3. In dynamic cooperations, where documents are owned by different users or organizations, the owner of a document should be free to determine the (initial) policy that applies to it.
4. Policies should be able to refer to dynamic groups, and should govern both the use and release of the document(s) they refer to as well as the administration of the policy itself (e.g. “any manager may modify this document as well as restrict this policy”).
5. The environment is highly decentralized and does not offer compatible role or level assignment (as in RBAC or MAC): in dynamic cooperations we cannot expect participating organizations to find a way of merging their MAC or RBAC systems for the time required by the cooperation.
6. When they believe circumstances justify it, it should be possible for a user to elect to perform actions that are not normally permitted. In this case, the decision should be subject to inquiry and review. Only users that can be held accountable should have this option.

We are not aware of any preventative policy enforcement methods that meet all these requirements. In particular, requirement (6) implies a significant deviation from a strictly preventative approach. Yet this requirement is particularly likely to be important in the context of cooperative arrangements that are in a state of flux. Moreover, emergencies often justify unanticipated exceptions, and require improvisation. Even the first five requirements cannot be met by preventative methods of which we are aware, namely the various AC models or DRM. Mandatory Access Control achieves end-to-end enforcement, but does not allow policy administration and requires a fixed level infrastructure (MAC can deal with (1), (2) and the first part of (4), but not with (3), (5) and (6), and the second part of (4)); on the other hand, Discretionary Access Control deals well

with the administrative problems, but does not realize end-to-end policy enforcement, nor can it deal with unexpected situations. Finally, as currently implemented, DRM is too rigid for collaborative environments; updating and/or merging documents and retransmitting the result to other users is beyond the capabilities of DRM systems to support. Moreover current DRM systems do not allow policy administration.

We believe that it is possible to satisfy the above requirements by utilizing a *detective* approach. By ensuring that unauthorized redistribution of data is detected, individuals can be held accountable for their actions. So long as the individuals in question are liable or otherwise exposed to redress, we argue that *detering* illegitimate behavior can be almost as effective as preventing it, with the great advantage of being much more flexible. We call this approach a *posteriori policy enforcement*.

To advance our thesis, this paper presents *APPLE* (A Posteriori Policy Enforcement), a framework for policy specification and end-to-end policy enforcement in collaborative environments. In *APPLE*, the burden of preventing violations of established policies is not assigned to a trusted component, but *rests with the user*, who is not prevented from acting wrongly, but is held accountable for her actions. *Distributed auditing authorities* routinely check whether users have obtained and used their data in accordance with the applicable policies. When audited, a user shows her *log* to demonstrate that (a) she possessed the policies allowing her to carry out the actions she performed, (b) she fulfilled the obligations required by these policies and (c) she acquired the policies from trustworthy sources.

There are three critical components of a-posteriori policy enforcement: logging, auditing, and accountability. Logging records actions taken by users. A log is the history of user’s action as recorded by the infrastructure. Auditing is the process whereby those logs are interrogated to determine whether the logs are consistent with the policies associated with transmitted documents and with observations made by the auditing authority about the apparent relationships between transmitted documents and other documents available to the user. Accountability is the property of a user of being susceptible to penalty should misbehavior be detected. The penalty mechanism needs to exist, but its nature is outside the scope of the present work. Interestingly, such critical components are increasingly present in large organizations: controls over access and changes to (critical) data have gained in their importance with the introduction of regulations such as the Sarbanes-Oxley Act. Here, enforcing audit trails (recording automatically who did what when to what document and how) is an essential part of having such controls in place. Our system leverages this and provides end-to-end policy enforcement in collaborative environments. It does this not by merging AC systems of participating organizations, but by using logging and auditing infrastructure that is becoming increasingly standard as companies seek to comply with regulations, as well as some trusted communication systems (more about this later).

The concrete contribution of the paper is twofold:

- We present *APPLE Core*, a logical framework for *a-posteriori policy enforcement*. This framework combines trust management and elements of audit logic. For instance, it uses trust management techniques to provide reasonable assurance that trusted parties are

compliant with auditing and would be sufficiently accountable for their misdeeds to deter them.

- We discuss conditions under which such a framework could be deployed in practice. We suggest techniques that could be used to satisfy the security assumptions of the logical framework.

The significance of using a logical framework to define a-posteriori policy enforcement is threefold: doing so (1) defines compliance in a precise, unambiguous manner, (2) organizes the content that must be included in the log and (3) defines precisely what queries an auditing authority can expect answers to.

A-posteriori policy enforcement offers interoperability, flexibility and scalability, which are crucial features in collaborative environments. On the other hand, a-posteriori policy enforcement does not give 100% guarantee that the data will not be used illegitimately—only that transgressors can be held accountable for their actions and are deemed to have something to lose when this takes place. Therefore relying exclusively on a-posteriori policy enforcement is appropriate only when resource owners are willing to accept some (albeit small) risk in the interest of the aforementioned benefits. Thus, while a-posteriori enforcement offers flexibility not provided by preventative means, preventative enforcement may be more appropriate when it is important that a certain prohibition not be violated under any circumstances. The two approaches are mutually complementary. In our view, a-posteriori policy enforcement *should be integrated* with classical (preventive) access control methods to provide greater flexibility, particularly with respect to empowering users to handle exceptional situations by overriding the pre-defined authorizations with the understanding that they will have to answer for their actions.

The rest of this paper is organized as follows. Section 2 provides an outline of the APPLE system. Section 3 presents the formal system, APPLE Core, and illustrates it with an example, and discusses important extensions. Section 4 provides some discussion and concludes.

2. OUTLINE OF THE APPLE SYSTEM

To illustrate the pivotal elements of the system we now give an informal description of it. Throughout this paper, we assume that data is contained in unstructured documents, and that to each document, a single policy applies. Handling structured data objects—such as databases that are shared by multiple users and that contain records having heterogeneous policies—are left as future work.

In a nutshell, the system works as follows.

(1) Each document (*i.e.*, data object) is always accompanied by a (sticky) policy that labels the document, and by a list of *parent documents* (*i.e.*, the list of (log entries of) documents from which it derives). This policy determines who may possess, modify and distribute the document, and who may modify the policy and in what ways.

(2) Users may *receive*, *modify* and *redistribute* documents and policies so long as the applicable policy allows this and the users *log* these actions. (When receiving a document the recipient must also check that the source of the data is trustworthy, as this provides reasonable assurance of the integrity and authenticity of the associated policy.) The log is secure in the sense that users can log actions but cannot

modify an entry in the log.

(3) Trusted components are used to realize an *auditable system*, in which all communication is logged; *e.g.*, no-one can send an email without logging it.

(4) *Distributed auditing authorities* routinely check whether users have obtained and used their data in accordance with the applicable policies. When audited, a user shows her *log* to demonstrate that at the time she performed the action, (a) policies she believed to be in force at the time permitted her to carry out the actions she performed, (b) she fulfilled the requirements of these policies and (c) she acquired the policies from trustworthy sources.

We concentrate on monitoring the transmission of documents. It is in transmitting sensitive documents that the gravest policy violations occur. Happily, transmission also affords the greatest opportunity for detection of unauthorized activity, since it is possible to observe transmission without cooperation of the monitored user. Increasingly secure-boot technologies are also enabling remote authorities to gain assurance in the trustworthiness of at least some components of monitored systems. This trend is likely to enable effective auditing of actions that take place entirely within the local host. In this case, our system could be extended to monitor other actions, such as printing, playing, viewing, or even simply possessing data objects.

Policies. The policy label of a document determines (a) who is the owner of the document, (b) who may transfer the document and who may receive it, (c) who may modify the document, (d) who may modify the associated policy and in what ways, and (e) who may merge the document with other documents, and under what conditions. The essence (though not the syntax) of one simple policy would be: *“this document is owned by Bob; this document may be transmitted to engineers working on project X, and may be modified only by senior executives.”*

Trustworthiness \approx Deterrence = Auditability + Accountability. When a user receives, creates, modifies or transmits a document, he is required to *check* whether the policy allows for this action and *log* this action and the evidence that shows that these actions were authorized at the time they were performed. He is also obligated to archive all policy-labeled documents so they can be inspected later by the auditing authorities. Additionally, when a user *receives* a document from someone else, he is required to check the trustworthiness (auditability and accountability) of the source to establish reasonable assurance that the policy labeling the document is authentic. This obligation of the recipient is analogous to that which occurs in the real world, as illustrated by the following except from a web article cautioning consumers about their culpability for purchases of illegitimate goods [1]: *Happening lately in Italy: a Danish tourist buys a pair of supposed designer glasses for 10 euros—and has to pay a fine of 3.333 euros. The reason: the Italian inland-revenue office is fighting against counterfeit products. ... Punished will be [anyone] who buys or accepts goods which quality, way of selling or price arouse suspicion that the rules relating to origin, provenance and intellectual property have been ignored. In future, it will be advisable to get some information about the origin of the*

product before the purchase.

The fact that consumers have some responsibility for questioning the legitimacy of the seller or the goods is a key factor in deterring commerce in stolen goods.

In our framework, we need a similar factor to deter the illegitimate diffusion of confidential documents. Suppose Alice collects in one document a number of medical records that she is not authorized to have. She attaches to the document a policy to the effect that “*this document is owned by Alice and may be disclosed to Bob.*” Alice then sells the policy-labeled document to Bob for \$4000. Just like in the case of the ersatz designer glasses, Bob needs to be cautious when accepting receipt of the illegitimate document. In our system, before accepting the document Bob must check whether Alice is auditable and accountable. If this is not the case, he will be unable to demonstrate to the auditing authority that he was sufficiently cautious when accepting the document.

Summarizing, APPLE is based on the premise that trustworthiness can be assessed indirectly by assessing deterrence from misbehavior. Deterrence in turn can be determined by assessing the chances misbehavior will be detected (auditability) and the likely consequences if that occurs (accountability).

Logging. Each user keeps a log of the security critical actions he executed, together with the evidence showing that he was allowed to perform them. There are four main types of log entries: *modification entries* (logging the creation and the modification of a document), *input entries* (logging the receipt of a document), *output entries* (logging the transmission of documents to other users), and *policy entries* (logging the policy rules that show actions were authorized at the time they were performed). In case of actions involving more than one agent (*i.e.*, communication), the log entry for these should contain a non-repudiable proof of origin and of receipt.

The essential requirement is that *all communication is logged*: the infrastructure must guarantee that no communication can take place without that the user creates an appropriate log of it. This could be achieved in a variety of ways. One is by using trusted software components on the hosts being monitored. To provide truly high assurance, such an approach would require secure-boot technology. Reasonable assurance can probably be obtained in environments where the user does not have administrator privileges and the installed software is tightly controlled by the IT provider. Another approach to increasing the level of assurance is to monitor network traffic.

If the communication is *encrypted*, then guaranteeing that all communication is properly logged provides certain challenges to APPLE. However, they can be managed. There are for instance two simple ways to allow APPLE to cope with encrypted communication: When sending/receiving a document, a user may either (a) log a non-encrypted version of it, together with a proof of the fact that the message that was sent/received was an encrypted version of the logged one, or (b) log the encrypted version, but then be ready to decrypt it when the AA asks him to do so (in this case the user has to store the keys used for encryption and decryption somewhere).

Auditing. Each system has one or more Auditing Authorities (AAs), in charge of checking that users behave correctly. Most of the AA operations can be carried out automatically; these are:

Entailment check The AA picks a document that it has become aware is (or should be) in the user’s possession, and checks if the policy belonging to it entails the policies of its parent documents.

Source check The AA picks a document that it has become aware is (or should be) in the user’s possession, and determines whether the user’s log contains evidence that the document is or was derived from a document received from another user. In this case, the AA determines whether the log contains satisfactory evidence that it was reasonable to trust the source with respect to the integrity and authenticity of the document’s policy label. It also determines whether that label permitted the communication.

Destination check The AA picks a document that it has become aware the user transmitted, and determines whether the user’s log contains evidence that the communication was permitted. This can include performing a source check to evaluate the circumstances in which the document came into the user’s possession.

Content check (this may eventually require human intervention) The AA picks a document that it has become aware is in the user’s possession, and analyzes the content of the document, looking for data whose provenance may not be correctly recorded, or its confidentiality, protected. The AA may in this and other checks enlist the assistance of other AAs that audit other users.

Why does it work. This system is clearly not watertight. Individual policy violations may go undetected. However the nontrivial risk of being caught and held accountable (for years) will, we believe, in many contexts deter frequent, significant violations. The principle is very similar to that used in criminal justice in open societies. Individuals are not controlled so tightly as to prevent all crime. However, society is not overwhelmed by criminality because of deterrence.

3. A FORMAL SYSTEM

This section present the APPLE system. It begins by briefly reviewing the simple trust management system that we use. Then it presents the APPLE Core—the model itself. An example then illustrates how the formal system is used. The section concludes by discussing some important extensions intended to increase the flexibility of the system.

Preliminaries: Trust Management and RT_0 . Trust management [7, 22, 10, 9, 11, 16, 17, 6, 19, 18, 25] is an approach to access control in decentralized distributed systems with access control decisions based on *credentials* issued by multiple principals. In trust management systems, credentials are maintained in a distributed manner and are often digitally signed to ensure their authenticity and integrity; such credentials are sometimes called *statements* or *certificates*. A trust management (TM) system is an essential component of APPLE, throughout the paper we shall employ the TM language RT_0 [19], however, we want to stress that APPLE would work in combination with *any* TM system. In the language RT_0 , a *principal* (*agent*) is a uniquely identi-

fied individual or process. Principals are denoted by names starting with an uppercase, typically, A, B, D . A principal can define a *role*, which is indicated by principal's name followed by the *role name*, separated by a dot. For instance $A.r$, and $UTSA.students$ are roles. A is the *owner* of $A.r$. We use names starting with a lowercase letter to indicate role names. A role denotes a set of principals (the principals that populate it, i.e., the members of the role). To indicate which principals populate a role, RT_0 allows the owning principal A (and only A) to issue (and revoke) four kind of *credentials*:

– *Simple Member*: $A.r \leftarrow D$. With this credential A asserts that D is a member of $A.r$.

– *Simple Inclusion*: $A.r \leftarrow B.r_1$. With this credential A asserts that $A.r$ includes (all members of) $B.r_1$. This represents a delegation from A to B , as B may add principals to become members of the role $A.r$ by issuing credentials defining (and extending) $B.r_1$.

– *Linking Inclusion*: $A.r \leftarrow A.r_1.r_2$. We call $A.r_1.r_2$ a *linked role*. With this credential A asserts that $A.r$ includes $B.r_2$ for every B that is a member of $A.r_1$. This represents a delegation from A to all the members of the role $A.r_1$.

– *Intersection Inclusion*: $A.r \leftarrow B_1.r_1 \cap B_2.r_2$. We call $B_1.r_1 \cap B_2.r_2$ an *intersection*. With this credential A asserts that $A.r$ includes every principal who is a member of both $B_1.r_1$ and $B_2.r_2$. This represents partial delegations from A to B_1 and to B_2 .

The definition of RT_0 given here is a slightly simplified (yet expressively equivalent) version of the one given in [19]. We use the term *role expression* to refer to an expression of any one of the forms $A, B.r, A.r_1.r_2$, or $B_1.r_1 \cap B_2.r_2$. At a given time, credentials currently issued, but not yet revoked by principals in the system are called *valid* credentials. The set of credentials that are currently valid is called the current *policy state*. We use \mathcal{P} to denote sets of valid credentials. For uniformity, we use the notation $\mathcal{P} \vdash B \in A.r$ to indicate that \mathcal{P} entails that B is a member of $A.r$. Below we report the semantics of RT_0 given in proof-system style (where for simplicity we omitted the rules for the logical operators and for \exists). It can be shown that this semantics is equivalent to the standard one i.e., that $\mathcal{P} \vdash B \in A.r$ iff $B \in [A.r]_{SP(\mathcal{P})}$ in the notation of [19].

$$\frac{}{\{A.r \leftarrow D\} \cup \mathcal{P} \vdash D \in A.r}$$

$$\frac{\{A.r \leftarrow A.s.t\} \cup \mathcal{P} \vdash \exists B, B \in A.s \wedge D \in B.t}{\{A.r \leftarrow A.s.t\} \cup \mathcal{P} \vdash D \in A.r}$$

$$\frac{\{A.r \leftarrow B.s\} \cup \mathcal{P} \vdash D \in B.s}{\{A.r \leftarrow B.s\} \cup \mathcal{P} \vdash D \in A.r}$$

$$\frac{\{A.r \leftarrow B.s \cap C.t\} \cup \mathcal{P} \vdash D \in B.s \wedge D \in C.t}{\{A.r \leftarrow B.s \cap C.t\} \cup \mathcal{P} \vdash D \in A.r}$$

3.1 APPLE Core

Now we introduce APPLE's formal framework. For the sake of simplicity, we assume that each user has exactly one host (we identify users and hosts). This restriction helps

the exposition and can simply be removed by referring to the combination user/host rather than just the user.

Policies & Documents. *Sticky policies* are a pivotal element of APPLE: each document is equipped with a policy label that governs its distribution and modification. Policy labels are constructed using *policy predicates*: $\{owner, maymodify, mayrefine, maytell\}$; we call the atomic formulas over these predicates *policy atoms*:

- $owner(A)$: “agent A owns this document”
- $maymodify(A)$: “agent A may modify this document”
- $maytell(A, B)$: means that the document may be sent from agent A to agent B . In practice, this has two meanings, according to the agent who is using it: (1): “agent A may send this document agent B ” and (2) “agent B may receive this document from A .”
- $mayrefine(A)$: “agent A may refine (make more restrictive) the policy of this document”

In general, these predicates may be applied to RT role expressions. For instance, we consider $owner(A.r)$ to be a policy atom stating that “any member of $A.r$ owns this document”. Technically, $owner(A.r)$ is just a shorthand for the first-order formula $\forall x(x \in A.r \rightarrow owner(x))$. Clearly, in this case, to establish the semantics of a policy we need to refer to the state (\mathcal{P}) of the trust management system, which determines the semantics of $A.r$. This forms the link between sticky policies and trust management in APPLE. For instance, the statement $\mathcal{P} \vdash owner(A.r) \rightarrow owner(B)$ indicates that the formula $\forall x(x \in A.r \rightarrow owner(x)) \rightarrow owner(B)$ is true in the state \mathcal{P} , i.e. that $\mathcal{P} \vdash B \in A.r$. Due to space constraints, we elide presentation of the proof rules that derive formulas involving policy predicates applied to general RT role expressions. A *policy label* is a conjunction of policy atoms.

Definition 3.1 A *document* is a triple $\langle id, L, \phi \rangle$, where id is an *identifier* that uniquely identifies the document, L is a list of identifiers representing the ancestor documents, and ϕ is the policy label associated with the document.

Example 3.2 A policy might contain the following atomic formula if anyone possessing the document can give it to anyone who is trusted by principal D : $maytell(x, D.trusted)$. It would be natural for D to define $D.trusted$ to include anyone who is considered compliant with auditing by an auditing authority that is trusted by D and who D considers accountable for their actions, for instance because they have something to lose if auditing finds out they are not adhering to policy. These objectives are met by the following two RT statements:

$$D.trusted \leftarrow D.auditable \cap D.accountable$$

$$D.auditable \leftarrow D.taa.compliant$$

Here *taa* is intended to be a role containing trusted auditing authorities and we assume such principals define their *compliant* roles to include agents who have been complying with auditing requirements of the auditing authority and are scheduled for further audit in the future. If the recipient is also only allowed to receive the document from principals that are trusted by a trusted auditing authority, the formula would become $maytell(D.trusted, D.trusted)$.

Logs & Compliance. There are two kinds of actions that are always logged: (1) creating a document and (2) transferring a document. The term $action(create(A, id))$ indicates the action in which user A has created the document with id id ; on the other hand $action(send(A, B, doc))$ indicates the action in which document doc was transferred from A to B .

Definition 3.3 A *log* is a sequence of entries, each of which is either an action or a set of credentials documenting that at a certain time, certain role memberships held, as required to satisfy the premise of certain inference rules used to prove compliance. Each log is associated with a user.

Logs are modeled as lists with most recent entries listed first. We use an infix dot for the list constructor, which takes an element as its left argument and a list as its right. The intended usage of logs is that when an auditor contacts an agent to audit it, the agent must provide its current log at that time to show he is compliant. Let A be an agent and l , its log. When the auditing authority (AA) audits A , it can do one of two things:

- The authority asks A to show A is (or was) authorized to have a document doc that A is observed to possess (or have possessed). In this case, A has to prove that $l \vdash_A have(doc)$.
- The authority asks A to show the legitimacy of the action act that is found in A 's log. In this case A has to prove that $l_{prev} \vdash_A act$. Here, l_{prev} is the prefix of l containing all entries created before act was performed.

Now let us see how \vdash_A is constructed. This is done by using a number of rules, shown in Figure 1. Note that \mathcal{P} is a set of credentials of the trust management system that are valid at the time the action is performed and that are needed to evaluate policies referring to trust management roles. In the rules, we suppress the details of how it is shown that credentials were valid at the time when the action was performed. This can be handled straightforwardly by using credential validity periods and timestamps.

Creation: When an agent creates a document, he becomes its owner (rule CREA).

Policy Change: A document's owner may change its policy arbitrarily (rule CHG). A is an owner if policy ϕ implies $owner(A)$ when evaluated in the current state \mathcal{P} : $\mathcal{P} \vdash \phi \rightarrow owner(A)$. \mathcal{P} is added to the log to record the fact that the valid policy at the time authorized A making the change to ϕ . This enables the validity of the inference to be checked subsequently by the AA. Note that \mathcal{P} need only include the relatively small number of credentials needed to support the derivation.

Refinement: When allowed A may refine (*i.e.* make more restrictive) the policy of a document (rule REF). Notice that this operation is rather powerful: for instance it allows A to remove a principal from the list of owners. The latitude for policy change permitted by refinement can be made more limited by adding an argument to $mayrefine$ indicating which predicates may be refined and which policy predicates may not be changed; in this case the above proof rule has to be split into a number of proof rules (one for each policy predicate) to keep the system at first order level. For the sake of simplicity, we leave this as it is.

Modification: When allowed A may modify a document (rule MOD).

Join: When allowed by the policies, A may join two documents (rule JOIN). Notice that we require the three policies ϕ_i to be equivalent to each other in the current state \mathcal{P} . Of course, if A is the owner (or may refine) one of the documents, he can adjust the policies before doing the merge.

Send: When allowed by the policy, A may send a document to B (rule SEND). When a document $\langle id, L, \phi \rangle$ is sent from A to B , A 's log contains $send(A, B, \langle id, L, \phi \rangle)$, while B 's log contains $send(A, B, \langle id', \{A : id\}, \phi \rangle)$. The parent document, identified by $A : id$, is identified as id within A 's namespace. This allows the recipient B to rename the document, while allowing the auditing authority to query A 's log for the provenance of the document.

Receive: B may keep a document received from A provided that the policy allows for it and that A is a trustworthy user. Note that we suppress the details of checking timestamps to ensure that the send occurred before B is authorized to possess the document.

Log Extension: A log that justifies the presence of a document can be extended arbitrarily to obtain a log that also justifies the presence of the document (rule EXT).

3.2 Example

Consider the following example. We have two companies: *CITA* (in Italy) and *CUS* (in the US). *CITA* has a hierarchical structure, while *CUS*, on the other hand, has a flatter structure

<i>CITA.partner</i>	\leftarrow	<i>Antonio</i>
<i>CITA.manager</i>	\leftarrow	<i>Luca</i>
<i>CITA.programmer</i>	\leftarrow	<i>Sandro</i>
<i>CITA.all</i>	\leftarrow	<i>CITA.partner</i>
<i>CITA.all</i>	\leftarrow	<i>CITA.manager</i>
<i>CITA.all</i>	\leftarrow	<i>CITA.programmer</i>
<i>CUS.ceo</i>	\leftarrow	<i>Bob</i>
<i>CUS.employee</i>	\leftarrow	<i>John</i>
<i>CUS.employee</i>	\leftarrow	<i>David</i>
<i>CUS.all</i>	\leftarrow	<i>CUS.ceo</i>
<i>CUS.all</i>	\leftarrow	<i>CUS.employee</i>

In both companies there is an agreement that employees may trusted all the sources that are trusted by the *partner*(resp. *ceo*) (*n.b.*, employees may be allowed to trust other sources as well).

<i>Luca.trusted</i>	\leftarrow	<i>CITA.partner.trusted</i>
<i>Sandro.trusted</i>	\leftarrow	<i>CITA.partner.trusted</i>
<i>John.trusted</i>	\leftarrow	<i>CUS.ceo.trusted</i>
<i>David.trusted</i>	\leftarrow	<i>CUS.ceo.trusted</i>

CITA and *CUS* decide to join forces on *projX*, and they agree that most of the documents developed in *projX* should be accessible only to people working at the project, and that some particularly confidential documents should circulate only among the senior personnel. To implement this, the two companies agree to employ the role names *projX* and *seniorprojX*. In *CITA*, the partner decides who participates to projectX, and decides (in agreement with *CUS*) that the managers of *CITA* should be considered senior people, while in *CUS*, the ceo delegates to *John* the definition of the projectX team as well as of the senior people in it.

$\frac{}{create(A, id).l \vdash_A have(\langle id, \{id\}, owner(A) \rangle)}$	CREA	$\frac{l \vdash_A have(\langle id, L, \phi \rangle) \quad \mathcal{P} \vdash \phi \rightarrow owner(A)}{\mathcal{P}.l \vdash_A have(\langle id, L, \phi' \rangle)}$	CHG	
$l \vdash_A have(\langle id, L, \phi \rangle) \quad \mathcal{P} \vdash \phi \rightarrow maymodify(A)$	MOD	$\frac{l \vdash_A have(\langle id, L, \phi \rangle)}{x.l \vdash_A have(\langle id, L, \phi \rangle)}$	EXT	
$l \vdash_A have(\langle id, L, \phi \rangle) \quad \mathcal{P} \vdash \phi \rightarrow mayrefine(A) \wedge \phi' \rightarrow \phi$		$\mathcal{P}.l \vdash_A have(\langle id, L, \phi' \rangle)$		REF
$l \vdash_A have(\langle id_1, L_1, \phi_1 \rangle) \quad l \vdash_A have(\langle id_2, L_2, \phi_2 \rangle) \quad \mathcal{P} \vdash \phi_1 \leftrightarrow \phi_2 \wedge \phi_1 \leftrightarrow \phi_3 \wedge \phi_1 \rightarrow mayjoin(A)$		$\mathcal{P}.l \vdash_A have(\langle id_3, \{id_1, id_2\} \cup L_1 \cup L_2, \phi_3 \rangle)$		JOIN
$l \vdash_A have(\langle id, L, \phi \rangle) \quad \mathcal{P} \vdash \phi \rightarrow maytell(A, B)$		$\mathcal{P}.send(A, B, \langle id, L, \phi \rangle).l \vdash_A action(send(A, B, \langle id, L, \phi \rangle))$		SEND
$\mathcal{P} \vdash \phi \rightarrow maytell(A, B) \quad \mathcal{P} \vdash A \in B.trusted$		$\mathcal{P}.send(A, B, \langle id', \{A : id\}, \phi \rangle).l \vdash_B have(\langle id', \{A : id\}, \phi \rangle)$		RCV

Figure 1: Rules of APPLE

Finally, *CITA* and *CUS* trust each other's definitions of (senior) people working at projectX.

<i>CITA.projX</i>	← <i>Antonio.projX</i>
<i>CITA.seniorprojX</i>	← <i>CITA.partner</i>
<i>CITA.seniorprojX</i>	← <i>CITA.projX</i> ∩ <i>CITA.manager</i>
<i>Antonio.projX</i>	← <i>Luca</i>
<i>Antonio.projX</i>	← <i>Sandro</i>
<i>CITA.projX</i>	← <i>CUS.projX</i>
<i>CITA.seniorprojX</i>	← <i>CUS.seniorprojX</i>
<i>CUS.projX</i>	← <i>John.projX</i>
<i>CUS.seniorprojX</i>	← <i>CUS.ceo</i>
<i>CUS.seniorprojX</i>	← <i>John.seniorprojX</i>
<i>John.seniorprojX</i>	← <i>John</i>
<i>John.projX</i>	← <i>John</i>
<i>John.projX</i>	← <i>David</i>
<i>CUS.projX</i>	← <i>CITA.projX</i>
<i>CUS.seniorprojX</i>	← <i>CITA.seniorprojX</i>

Policies. Having fixed the credentials of the current policy state, let us consider some example policy labels. The label of a document that may be seen and modified only by the senior members of projectX is

$$owner(CITA.seniorprojX) \wedge maymodify(CITA.seniorprojX) \wedge maytell(CITA.seniorprojX, CITA.seniorprojX)$$

The document labeled by the following policy can be seen by non-senior people, provided that it is given to them by a senior person

$$maytell(CITA.seniorprojX, CITA.projX) \wedge \dots$$

Let us now consider a document-lifecycle scenario. *Luca* (a manager at *CITA*) creates a document with id id_1 . *Luca*'s log is now $create(Luca, id_1)$, and the document he possesses $\langle id_1, \{id_1\}, owner(Luca) \rangle$. *Luca* immediately modifies the policy label to be

$$\phi_1 : owner(CITA.seniorprojX) \wedge maymodify(CITA.projX) \wedge mayrefine(CITA.projX) \wedge maytell(CITA.projX, CITA.projX)$$

Notice that no credentials need to be used to show that *Luca* is authorized to make this change to the policy. *Luca* next sends the document to *David*. At this point, *Luca*'s host contains $doc_2 = \langle id_1, \{id_1\}, \phi_1 \rangle$, and *Luca*'s log is $\mathcal{P}_2.send(Luca, David, doc_2). \mathcal{P}_1.create(Luca, id_1)$, in which \mathcal{P}_1 is the empty set and

$$\mathcal{P}_2 = \left\{ \begin{array}{ll} CITA.projX & \leftarrow Antonio.projX \\ CITA.projX & \leftarrow CUS.projX \\ Antonio.projX & \leftarrow Luca \\ CUS.projX & \leftarrow John.projX \\ John.projX & \leftarrow David \end{array} \right\}$$

At this point *David*'s log is $\mathcal{P}_3.send(Luca, David, doc_3)$, where $doc_3 = \langle id_2, \{Luca : id_1\}, \phi_1 \rangle$ (notice that in *David*'s system the document has been renamed) and \mathcal{P}_3 consists of credentials that show *David* has reason to trust *Luca*.

If now the auditing authority asks *Luca* to justify the possession of this document, *Luca* is able to prove that $log_{Luca} \vdash_{Luca} have(doc_2)$ (This proof is reported in Table 1 (top)). Furthermore, if the AA asks *Luca* to account for sending doc_2 to *David*, he also has to show that $log_{Luca} \vdash_{Luca} action(send(Luca, David, doc_2))$. This latter proof (which encompasses the first one) is reported in Table 1 (middle).

After receiving the document, *David* plans to modify its content. As the planned modification would add information that must be approved by managers before being released, he begins by refining the policy label to be more restrictive about where the document can be sent:

$$\phi_2 : owner(CITA.seniorprojX) \wedge maymodify(CITA.projX) \wedge mayrefine(CITA.projX) \wedge maytell(CITA.projX, CITA.seniorprojX)$$

Now, *David*'s host has the document $doc_4 = \langle id_2, \{Luca : id_1\}, \phi_2 \rangle$, while his log has been extended with a set of credentials \mathcal{P}_4 that show he was authorized to refine the policy. Next *David* modifies the document, obtaining $doc_5 = \langle id_3, \{id_2, Luca : id_1\}, \phi_2 \rangle$, and sends doc_5 to his boss, *John*. After this operation *David*'s log contains

$\frac{\frac{\text{create}(Luca, id_1) \vdash_{Luca} \text{have}(doc_1)}{\mathcal{P}_1.\text{create}(Luca, id_1) \vdash_{Luca} \text{have}(doc_2)}}{\text{CREA}} \quad \mathcal{P}_1 \vdash \phi_1 \rightarrow \text{owner}(Luca)$ CHG	$\frac{\frac{\frac{\text{create}(Luca, id_1) \vdash_{Luca} \text{have}(doc_1)}{\text{CREA}} \quad \mathcal{P}_1 \vdash \phi_1 \rightarrow \text{owner}(Luca)}{\text{CHG}}}{\text{CHG}} \quad \mathcal{P}_1.\text{create}(Luca, id_1) \vdash_{Luca} \text{have}(doc_2)$
$\frac{\frac{\frac{\text{create}(Luca, id_1) \vdash_{Luca} \text{have}(doc_1)}{\text{CREA}} \quad \mathcal{P}_1 \vdash \phi_1 \rightarrow \text{owner}(Luca)}{\text{CHG}} \quad \mathcal{P}_2 \vdash \phi_1 \rightarrow \text{magfell}(Luca, David)}{\text{SEND}} \quad \mathcal{P}_1.\text{create}(Luca, id_1) \vdash_{Luca} \text{have}(doc_2)$ $\mathcal{P}_2.\text{send}(Luca, David, doc_2) \cdot \mathcal{P}_1.\text{create}(Luca, id_1) \vdash_{Luca} \text{action}(a_2)$ <p style="text-align: center;">where $a_2 = \text{send}(Luca, David, doc_2)$ and $a_1 = \text{create}(Luca, id_1)$</p>	
$\mathcal{P}_3 \vdash \phi \rightarrow \text{magfell}(Luca, David) \quad \mathcal{P}_3 \vdash_{Luca \in \text{David.trusted}}$ $\frac{\mathcal{P}_3.\alpha_1 \vdash_{\text{David}} \text{have}(doc_3)}{\text{CHG}} \quad \mathcal{P}_4 \vdash \phi_1 \rightarrow \text{magrefine}(\text{David}) \wedge \phi_2 \rightarrow \phi_1$ $\frac{\mathcal{P}_4.\mathcal{P}_3.\alpha_1 \vdash_{\text{David}} \text{have}(doc_4)}{\text{CHG}} \quad \mathcal{P}_5 \vdash \phi_2 \rightarrow \text{magmodify}(\text{David})$ $\frac{\mathcal{P}_5.\mathcal{P}_4.\mathcal{P}_3.\alpha_1 \vdash_{\text{David}} \text{have}(doc_5)}{\text{CHG}} \quad \mathcal{P}_6 \vdash \phi_2 \rightarrow \text{magfell}(\text{David}, \text{John})$	$\mathcal{P}_6.\alpha_2 \cdot \mathcal{P}_5 \cdot \mathcal{P}_4 \cdot \mathcal{P}_3.\alpha_1 \vdash_{\text{David}} \text{action}(a_2)$
<p>Here we have omitted the rule labels (from top to bottom: SEND, MOD, REF, RCV). Also $a_1 = \text{send}(Luca, David, doc_3)$, and $a_2 = \text{send}(David, John, doc_5)$. The \mathcal{P}_i are minimal sets of credentials valid at the appropriate times and sufficient to prove the associated formulas.</p>	
$\begin{aligned} doc_1 &= \langle id_1, \{id_1\}, \text{owner}(Luca) \rangle \\ doc_2 &= \langle id_1, \{id_1\}, \phi_1 \rangle \\ doc_3 &= \langle id_2, \{Luca : id_1\}, \phi_1 \rangle \\ doc_4 &= \langle id_2, \{Luca : id_1\}, \phi_2 \rangle \\ doc_5 &= \langle id_3, \{id_2, Luca : id_1\}, \phi_2 \rangle \end{aligned}$	

Table 1: *Luca's* proofs (the first above, the second in the middle) and *David's* proof (below)

$\mathcal{P}_6.send(David, John, doc_5). \mathcal{P}_5. \mathcal{P}_4. \mathcal{P}_3.send(Luca, David, doc_3)$

where \mathcal{P}_5 shows *David* was authorized to modify the document and \mathcal{P}_6 shows he was authorized to send it to *John*. At this stage, if the AA asks *David* to account for sending the document to *John*, then *David* has to prove that $log_{David} \vdash_{David} action(send(David, John, doc_5))$. For this, he has to combine rules SEND, MOD, REF and RCV, to show that he was allowed to receive, refine, modify, and send the document (the complete proof is reported Table 1).

Finally, after receiving the document from *David*, *John* may decide that this document may be seen by all participants to projectX; since *John* is an owner of the document (*John* is in *CITA.seniorprojX*), he can change the policy of the document from ϕ_2 to ϕ_1 .

Note that if there had been an unanticipated emergency in which all members of the project had to see the document right away, but *John* could not be reached, *David* could go ahead and send out the document. The AA would detect this in the next audit, since *David* was not technically authorized to release the document to all project members. The AA would then contact the document owners and inform them of the infraction. A review of *David's* actions could then be conducted. It may be that *David's* ability to override the authorization policy (but then have to take responsibility for this decision) would then be found to have been essential to the success of the project. (It also may be that *David* will be found to have acted inappropriately, in which case appropriate steps can be taken because the document owners are made aware of the situation.)

4. DISCUSSION AND CONCLUSIONS

Although historically access control has been the primary means of enforcing policy, there has recently been an increase in the amount of logging required, particularly in business systems. Recent legislation, such as the Sarbanes-Oxley Act, have increased the amount of monitoring of how data is collected and used. Thus, much of the infrastructure needed to enable a posteriori policy enforcement may already be finding its way into contemporary organizations.

Intended Environment. We now briefly discuss several assumptions concerning the environment in which APPLE is proposed for use. We assume that all policy-relevant actions are audited securely. Although this is not happening yet in practice, there is a clear tendency towards increased auditability. Industries that deal heavily with confidential data already do a great deal of logging. In addition, these and many other industries do not usually allow their employees to install their own applications. In this case, the logging could take place at the operating system level, and the logged material be temporarily cached on secure storage and later transferred to some log depository.

Naturally, the requirement that all communication be logged raises scalability concerns. There are several approaches one might to mitigate this potential problem. For instance, assuming that the user is not allowed to modify applications and the operating system installed, one could allow certain trusted programs that cannot be used to communicate documents to be used without logging. For instance, one could allow the use of restricted versions of VOIP programs and web browser, where the restriction would be that they would not allow sending a document, and that they would use a special protected cache. Thus, logging would be necessary

only when these programs would be used to send or receive documents, or if a cached document should be moved to the user area. A related issue is the control of information hidden within other data, for instance, by use of steganography. Our proposal is to disallow the use of the controlled computer system for transmission of data in which other information can easily be hidden. Similarly, we propose to disallow transmission of encrypted material from these hosts.

Technically, the crucial point is that the audit log should be rich enough to enable the auditor subsequently to determine whether an operation was authorized or not. This is virtually impossible if the logged material consist e.g., of TCP/IP packets, because given a number of logged packets it is nearly impossible to establish which is the document and which is the policy that the packed is about. On the other hand, in the situation like the above in which applications cannot be installed by the user, we believe it would be possible to realize a logging at a higher level than the Network Layer, in such a way that the logged material is sufficiently expressive to determine documents and policies involved with the communication. Once this is possible, our results show that it is possible to determine whether the operation was allowed or not.

Notice also that our system can be used by the very user also to determine whether a certain operation is allowed or not. Indeed, since policies are coupled with the documents they protect, a user can simply employ a reference monitor based on the proof system we present to check whether e.g., sending a document to a given person would be legitimate.

Data Tracking. A nice side-feature of the system, is that (assuming that users behave correctly) it is always possible to check who has received the information that was contained in a specific document. This might be valuable when it becomes necessary to retract or correct some information. For instance, suppose that Bob sends around a document containing some confidential information that is meant only for the senior people; if Bob later finds out that the information was not correct, and should be revised, he may be able to exploit the system to inform everyone who received the document (or a derivative of it) of the erroneous information. In fact, also if we do not assume that the users behave correctly, it is to some extent possible to check what has happened to a specific piece of information. This is easier if the piece of information is short and structured, like an email address, and can be done by following the chain downwards, from the source to the destinations, with the additional condition that at each step data mining and pattern matching techniques are applied to see if the info has not spuriously entered a document that was not enlisted as child of the original one.

Related work. As we previously mentioned, policy enforcement is presently carried out by *access control* (e.g., [15, 23, 21, 5, 2]), *digital right management* (e.g., [13, 12, 14, 24]), or *trust management* (e.g., [7, 22, 10, 9, 11, 16, 17, 6, 19, 18, 25]); these three areas have a large body of literature. However, APPLE is radically different from these approaches in that it does not provide a preventive method for compliance checking. To mention the papers with some affinity: in Originator Control [20], the original owner of the data can always change the access rights on the data, while the current owner

of the data can not do so. Cassandra [4], is a role based trust-management designed to implement privacy policies of health records in an electronic health record (EHR) system. Under certain conditions, Cassandra allows to “break the seal” of certain documents, and this is reminiscent of APPLE’s feature of allowing unauthorized actions. Related our auditing by means of proofs, Appel and Felten [3] propose the Proof-Carrying Authentication framework for the authorization of clients to web servers, by using distributed policies. Related to the logic of APPLE is [8], where Cederquist et al. outline a logic for accountability, with core notions of data and agent accountability. APPLE however is also fundamentally different from [8], in the fact that APPLE’s incorporation of trust management greatly facilitates its application in decentralized environments, in particular, the logic in [8], does not allow one to express and refer to dynamic groups, making it too restrictive for the collaborative environment we focus on. Cederquist et al. also did not address practical concerns regarding how the system can be deployed.

Conclusions. In this paper we make a case for a posteriori policy enforcement. First, we argue that—particularly for collaborative environments—detective approaches to policy enforcement may present great advantages to the usual preventative ones such as AC or DRM. In particular, detective approaches can be integrated with preventive approaches to form a second line of defence which is particularly useful when data has to move across organizations and when there is the need to cope flexibly and efficiently with unexpected situations. Secondly, we provide a theoretical framework for a-posteriori policy enforcement. The framework is substantially different than any other logic for access control (with the exception of our previous work [8], discussed above), in that it relies on the log of the system. This framework combines a logic for accountability with trust management to provide a flexible system in which policies include administration rights and one can define groups in a simple and effective way. Finally, we discuss the main (non-trivial) practical issues that need to be solved to realize this form of policy enforcement. The salient feature of this approach is that it does not prevent illegitimate behavior, but rather deters it. Thus it does not preclude transgressions, but it reduces their impact to a *manageable risk*.

5. REFERENCES

- [1] European consumer centre. <http://www.euroconsumatori.org/16856v16856d24338.html>.
- [2] M. Abadi. Logic in access control. In *8th IEEE Symposium on Logic in Computer Science (LICS)*, pages 228–233. IEEE Computer Society Press, 2003.
- [3] A. W. Appel and E. W. Felten. Proof-carrying authentication. In G. Tsodik, editor, *Proc of the 6th Conference on Computer and Communications Security*. ACM Press, 1999.
- [4] M. Y. Becker and P. Sewell. Cassandra: flexible trust management, applied to electronic health records. In *IEEE Computer Security Foundations Workshop (CSFW)*, pages 139–154. IEEE Computer Society Press, 2004.
- [5] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A logical framework for reasoning about access control models. *ACM Transactions on Information and System Security (TISSEC)*, pages 71–127, 2003.
- [6] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote trust-management system, version 2. IETF RFC 2704, September 1999.
- [7] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In IEEE Computer Society Press, editor, *Proc. 1996 IEEE Symposium on Security and Privacy*, pages 164–173, 1996.
- [8] J. Cederquist, R. Corin, M. A. C. Dekker, S. Etalle, and J. I. den Hartog. An audit logic for accountability. In A. Sahai and W. Winsborough, editors, *6th Int. Workshop on Policies for Distributed Systems & Networks (POLICY)*, pages 34–43. IEEE Computer Society Press, 2005.
- [9] D. Clarke, J-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R. Rivest. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(4):285–322, 2001.
- [10] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI certificate theory. IETF RFC 2693, September 1999.
- [11] C. Gunter and T. Jim. Policy-directed certificate retrieval. *Software: Practice & Experience*, 30(15):1609–1640, September 2000.
- [12] C. Gunter, S. Weeks, and A. Wright. Models and languages for digital rights. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)*, pages 4034–4038. IEEE Computer Society Press, 2001.
- [13] H. Guo. Digital rights management (DRM) using XrML. In *T-110.501 Seminar on Network Security 2001*, page Poster paper 4, 2001.
- [14] R. Iannella. Open digital rights management. DRM Workshop, Position Paper 23, 2001. <http://www.w3.org/2000/12/drm-ws/pp/>.
- [15] S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino. A unified framework for enforcing multiple access control policies. In J. Peckham, editor, *Proc. International Conference on Management of Data (SIGMOD)*, pages 474–485. ACM Press, 1997.
- [16] T. Jim. SD3: A trust management system with certified evaluation. In *IEEE Symposium on Security and Privacy*, pages 106–115. IEEE Computer Society Press, 2001.
- [17] N. Li, B. Grosz, and J. Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM Trans. Information System Security*, 6(1):128–171, 2003.
- [18] N. Li, J. Mitchell, and W. Winsborough. Design of a role-based trust-management framework. In *IEEE Symposium on Research in Security and Privacy*, pages 114–130. IEEE Computer Society Press, 2002.
- [19] N. Li, W. Winsborough, and J. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security*, 11(1):35–86, 2003.
- [20] J. Park and R. Sandhu. Originator control in usage control. In *IEEE Int. Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 60–66. IEEE Computer Society Press, 2002.
- [21] J. Park and R. Sandhu. Towards usage control models: Beyond traditional access control. In *ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 57–64. ACM Press, 2002.
- [22] R. Rivest and B. Lampson. SDSI — a simple distributed security infrastructure, 1996. <http://theory.lcs.mit.edu/~rivest/sdsi11.html>.
- [23] R. Sandhu and P. Samarati. Access control: Principles and practice. *IEEE Communications Magazine*, 32(9):40–48, 1994.
- [24] X. Wang, G. Lao, T. De Martini, H. Reddy, M. Nguyen, and E. Valenzuela. XrML: eXtensible rights markup language. In M. Kudo, editor, *Proc. 2002 ACM workshop on XML security (XMLSEC-02)*, pages 71–79. ACM Press, 2002.
- [25] S. Weeks. Understanding trust management systems. In *IEEE Symposium on Security and Privacy*, pages 94–105. IEEE Computer Society Press, 2001.