A State-Transition Model of Trust Management and Access Control

Ajay Chander*
Computer Science Department
Stanford University
ajayc@cs.stanford.edu

Drew Dean†
Xerox PARC
ddean@parc.xerox.com

John C. Mitchell*
Computer Science Department
Stanford University
mitchell@cs.stanford.edu

Abstract

We use a state-transition approach to analyze and compare the core access control mechanisms that are characteristic of a variety of trust management, access control list, and capability-based systems. The framework, which characterizes the set of rights a subject has over an object after any sequence of actions, is based on abstract system states, state transitions, and logical deduction of access control judgments. We present abstract models representing the access control portion of trust management, access control lists, and two versions of capabilities, proving various correspondence and simulation relations between these models. The main results include an equivalence between access control lists (ACLs) and capabilities viewed as rows of the Lampson access matrix and the (proper) subsumption of a form of ACLs by an "unforgeable reference" form of capabilities. The access control mechanism at the heart of distributed trust management systems is formally shown to provide a tractable compromise between unrestricted capability passing from the capability models and easy revocation provided by access control lists. The underlying simulations show how trust management compares with more established access control mechanisms, independent of features such as local name spaces and certificate authorization hierarchies.

1. Introduction

Many approaches to discretionary access control have been proposed, studied, and implemented over the years [12, 22, 2]. Trust management is an emerging approach based on cryptographic keys, signed credentials, and policies [5, 6, 7]. As an access control mechanism, apart from cryptographic issues, a characteristic of trust management is *delegation*: a principal can delegate access to a resource, provided that the principal has been granted the ability to do so by the owner of the resource or by some chain of delegations. Our goal in this paper is to study the power of delegation, in comparison with other mechanisms such as transfer of capabilities in capability-based models, and prove precise relationships between trust management and other mechanisms. To this end, we present precise models of the abstract features of the access control portion of trust management, access control list, and capability systems, and prove the existence and nonexistence of simulation relations between these systems.

Lampson's access matrix [17], refined by Harrison, Ruzzo, and Ullman [14], has been used widely as the basis for comparing earlier access control mechanisms. While the access matrix is a useful model, it is not sufficiently expressive to account for properties of delegation in trust management, or the way that capabilities (when represented as unforgeable "tickets") can be passed from one user to another. We therefore propose a new model of access control, based on abstract system states, state transitions, and logical deduction of access control judgments. The main idea is simply to identify a set of abstract system states, each containing the kind of information that would be maintained in an access control system. The important property of each state is the set of access requests that will be allowed in this state, and the access requests that will be allowed after subsequent actions such as the transfer of a capability. The set of allowed access requests may be recorded directly in the state, as in access control lists, or derived from properties of the state by some form of logical inference. In this framework, we compare access control mechanisms by comparing the resulting "abstract state machines," using traditional forms of simulation relations from programming language and concurrency theory.

For each pair of access control mechanisms, we consider the ways that the actions of one mechanism can be simulated by actions of another. An interesting subtlety is whether the mapping between actions is one-to-one or one-to-many. While a one-to-many correspondence, in which one action in one model is equivalent in effect to a sequence of actions in another, may seem a suitable equivalence be-

*Supported in part by DARPA grant N66001-00-C-8015.
†Supported in part by DARPA grant N66001-00-1-8921.

tween models, there is the potential for the latter mechanism to be subject to attacks that cannot be carried out against the first model. Since this can be prevented by enforcing atomicity of the implementing sequence of actions in the second model, the distinction between one-to-one and one-to-many simulations may provide useful guidance in practice.

The main simulation results are summarized as a set of four diagrams, each showing the existence or impossibility of one-to-one and one-to-many simulations between access control models. In brief, the main results are that access control lists are equivalent to capabilities, when capabilities are regarded as rows of an access control matrix. However, when properties of the "unforgeable ticket" implementation of capabilities are taken into account, capabilities can weakly (one-to-many) simulate access control lists, but not conversely. Trust management systems, modeled here without keys or name spaces, can vary the delegation depth to strongly (one-to-one) simulate the other mechanisms, providing a tractable compromise between unrestricted capability passing from the capability model and easy revocation provided by access control lists. In this general case, trust management systems can provide feasible revocation, and we may identify trust credentials with history-dependent capabilities. More generally, our study suggests that trust management subsumes both earlier mechanisms in specific ways. These theorems depend on the exact nature of what is considered observable, and highlight some subtle differences in capability systems that might be easily overlooked.

The rest of this paper is organized as follows. Section 2 defines our model. Sections 3 through 5 formalize trust management, access control lists, and two models of capabilities. Section 6 presents a sequence of results capturing the relationships between these models, with an interpretation in Section 6.5. We make some concluding remarks in Section 7, and point out directions for further research. Some additional technical points and examples are presented in the appendices.

2. The State-Transition Model

An access control mechanism decides, given a configuration of the system, whether a subject is allowed a certain right on an object. For example, given a set of permissions at a web site, a web access control system must decide whether a specific employee of an organization can read a web page hosted within the corporate intranet. While authentication is required, we will not consider this part of the access control mechanism. Therefore, an access control mechanism comprises a set of system states, subjects, rights, objects, and an access algorithm. Along with a set of system states, an access control mechanism also provides a set of operations that change the system from one state to another. With these concepts in mind, our model of access control mechanisms has the following parts:

- A *world state*, which is the part of the system configuration that is relevant to the access control mechanism,
- A set of possible *actions*, each of which defines a transition mapping from world states to world states. Examples of actions include, create a resource, allow access, revoke access, and so on,
- An *access judgment*, which states when one object can access another. This may be specified in the form of logical inference, equivalent to some implementable algorithm. Given a world state WS, the judgment that subject s can access the object o with right r is written $WS \vdash s \to (o, r)$.

In the remainder of the paper, we will work with models of trust management, access control list, and capability systems consisting of these three parts. As suggested above, the world states will include subjects, rights, objects, and associations between them, actions will modify these associations, and access judgments will specify the accesses that are allowed in a given state.

Over time, differing versions of each access control mechanism have been proposed and implemented. For example, AFS [15] uses a fine grained form of access control lists, whereas classical Unix operating system permission bits can be seen as a restricted, coarse grained form of access control lists. Both systems add groups and define evaluation orders (see Example C.1 in Appendix C). Proposed capability-based systems [18] vary in a number of ways. The most significant for our study is the difference between systems that allow unrestricted passing of capabilities and systems that impose control through reference monitors and indirection. Trust Management systems [6, 8, 5] are quite recent, and have several variations. In order to give precise comparisons of some value, we have attempted to identify representative properties of each class of access control mechanism. In particular, we analyze and compare the well-understood centralized control and ease of revocation within access control lists with the unrestricted passing of capability systems, with newer trust management systems, which allow for bounded delegation.

This work follows in the tradition of Kain and Landwehr [16], although it provides a more precise and broader setting in which to reason about different access control mechanisms. Sandhu and Ganta have conducted related studies [23, 25, 24, 11] in the context of access matrix models and their variants; our focus has been on studying a wider range of different access control mechanisms in the same formal framework, and the development of general specification and comparison techniques.

3. Modeling Trust Management (M_{tm})

Trust Management [7] is a relatively recent proposal for access control, in which an access request is accompanied by a set of credentials which together (by transitive closure) constitute a proof as to why the access should be allowed. Resources may grant access, as well as the ability to further delegate that access a restricted number of times. We illustrate this by means of an example.

EXAMPLE 3.1 *The DoD Firewall Proxy*

A classic example in the trust management literature given in favor of delegating authority and reducing burden on a central omniscient directory of users is the DoD firewall access scenario [10]. Instead of enforcing strict control by a central database of users (access control list), the same end is sought by a small root access control list (containing say the president, ministry officials, and chiefs of staff) and carefully defined policies implemented by delegation. For example, private `joe` may be able to access file `foo` inside the firewall because the secretary of defense allowed a general to allow a captain to allow `joe` to access the file.

Lets denote the objects corresponding to the secretary of defense, the general, the captain, and `joe` by K_s, K_g, K_c, and K_j respectively (these could be their private keys, for example.) Then we can model the access control policy described above by the pseudo-credentials

$$\begin{aligned} \text{Allow}(\texttt{foo}, \texttt{read}) &\ni (K_s, 7) \\ \text{Delegate}(K_s, \texttt{foo}, \texttt{read}) &\ni (K_g, 5) \\ \text{Delegate}(K_g, \texttt{foo}, \texttt{read}) &\ni (K_c, 2) \\ \text{Delegate}(K_c, \texttt{foo}, \texttt{read}) &\ni (K_j, 0). \end{aligned}$$

Clearly, we may construct a proof of $\text{Allow\_Access}(K_j, \texttt{foo}, \texttt{read})$ by a transitive closure of the above credentials. Note that revocation of any of these credentials would cause the access to fail. We will formally compare this with capability revocation later in the paper. □

Current trust management systems [6, 8, 5] include other features like support for local name spaces and certificate authorization hierarchies. While these additions are useful, and it is important to understand them formally [1, 13, 19], they are orthogonal to the basic access control mechanism at the heart of trust management. Thus our model for trust management comprises of the following:

1. **World State** (WS): The world state contains a set O of objects, a set R of rights and two maps A (bounded RootACL) and D (bounded Delegate):

$$\begin{aligned} A &: O \times R \to \mathcal{P}(O \times \mathcal{N}) \\ D &: O \times R \times O \to \mathcal{P}(O \times \mathcal{N}) \end{aligned}$$

We do not specify a separate set of subjects; each object can act as a subject as well, requesting access rights on another object. Access to a right on an object in a trust management setting is either directly allowed by the (o, r) pair in question, or is propagated by someone who holds that right, and we model these two cases by the maps A and D respectively. In this sense, access propagation in these systems is a hybrid between the object-controlled access control list, and the subject-controlled capability.

$A((o, r))$ specifies the set of objects which can access right r on object o, and the depth to which they can delegate that right. Thus if $(o_s, n) \in A((o, r))$, then o_s can access right r on object o, and can delegate that access right to another object o_d which can then delegate it to a maximum effective depth of $n - 1$. The delegation action would be modeled as $(o_d, n-1) \in D(o_s, r, o)$. In general if an object o_s delegates its access right on (o, r) to a delegatee object o_d with depth d, then the set of such delegations is captured by $D(o_s, o, r)$.

2. **Actions**: An action specifies a transition from one world state to another. For a given world state w, we denote the result of an action α that takes the vector of arguments $\vec{v}$ by $\alpha(\vec{v}; w)$. We assume that the world state w's components before the action are given by O, R, A, and D, and afterwards, by O', R', A' and D'. For each of the actions below, we informally state what the action is supposed to accomplish, define it by an equation, and then check to see if the equation matches our original intuition. The following actions are defined, together with the world state components that are modified (see table 1 in Appendix B for a summary of all actions):

• *Create* (object): Object o_c creates a new object o.

$$\text{Create}(o_c, o; w) = (O \cup \{o\}, R, A', D'), \text{ where}$$

$$\begin{aligned} A'(o, r) &= \begin{cases} (o_c, 1) & r = r_e \\ \emptyset & r \neq r_e \end{cases} \quad r \in R \\ D' &= D[(s, r, o) \mapsto \emptyset | s \in O, r \in R] \end{aligned}$$

The new object o is added to the set of objects O, and the creator o_c to the rootacl for (o, r_e). Since o_c created o, it is given the right r_e of editing o's rootacl, and the ability to delegate that right to anyone it wishes to. No one else holds any rights to o at this point.

• *Add* (to rootacl): Allow object o_s right r on object o, with delegation powers upto depth d.

$$\text{Add}(o, r, o_s, d; w) = (O, R, A', D), \text{ where}$$

$$A' = A[(o, r) \mapsto A((o, r)) \cup \{(o_s, d)\}]$$

The only state transformation caused by the *Add* action is the addition of the pair (o_s, d) to the rootacl of A. Since this newly obtained right has not yet been delegated, the map D and other state components remain the same.

• *Remove* (from rootacl): Remove object o_s's access to right r on object o.

$$\text{Remove}(o, r, o_s, d; w) = (O, R, A', D), \text{ where}$$

$$A' = A[(o,r) \mapsto A((o,r)) - \{(o_s,d)\}]$$

Again, the only change directly caused by the *Remove* action is the removal of (o_s, d) from the rootacl of (o, r), as reflected in the updated map A. Note that D has not changed after the action, even though accesses previously delegated through o_s will now fail.

• *Delegate* (access right): Object o_s delegates its right to access (o, r) to delegatee object o_d, and allows it to delegate it further to depth d.

$$\text{Delegate}(o_s, o, r, o_d, d; w) = (O, R, A, D'), \text{ where}$$

$$D' = D[(o_s, r, o) \mapsto D((o_s, r, o)) \cup \{(o_d, d)\}]$$

The only state component directly modified by the *Delegate* action, is the map D, which is updated to reflect the added delegation in the obvious way. All other components remain the same.

• *Revoke* (delegated access right): Object o_s revokes its previously delegated right to access (o, r) with delegation depth d from object o_d.

$$\text{Revoke}(o_s, o, r, o_d, d; w) = (O, R, A, D'), \text{ where}$$

$$D' = D[(o_s, r, o) \mapsto D((o_s, r, o)) - \{(o_d, d)\}]$$

Again, the *Revoke* action changes only the delegation map in the obvious way. All other components remain the same.

• *Delete* (object): Object o is deleted from the system.

$$\text{Delete}(o; w) = (O - \{o\}, R, A\mid_{(O-\{o\})}, D\mid_{(O-\{o\})})$$

The *Delete* action removes all instances of object o from the system, thereby removing it from the set of objects O, its rootacls from the map A, and all delegated access rights to it from the map D. In other words, the maps A and D are updated by restricting their domains to $(O - \{o\}) \times R$ and $(O - \{o\}) \times R \times (O - \{o\})$ respectively.

3. **Access Judgment**: We specify the access judgment as a logical judgment, given the following four inference rules. The component maps of the world state can be interpreted as set-membership predicates; $\text{ACL}(s, o, r, d)$ is true iff $(s, d) \in A((o, r))$, and $\text{Del}(s, o, r, r_s, d)$ is true iff $(r_s, d) \in D(s, o, r)$. Subject s can access the (o, r) pair *iff* it can produce a proof of $\text{Access}(s, o, r, d)$, for some d, from the predicate equivalent of the world state and the following four inference rules:

$$\begin{array}{lr}
(\textit{RootACL}) & \text{ACL}(A, B, r, d) \supset \text{Access}(A, B, r, d) \\
(\textit{Delegation}) & \text{Access}(A, B, r, d) \wedge \text{Del}(A, B, r, C, d-1) \\
 & \supset \text{Access}(C, B, r, d-1) \\
(\textit{Ord1}) & \text{Access}(A, B, d+1) \supset \text{Access}(A, B, d) \\
(\textit{Ord2}) & \text{Del}(A, B, r, C, d+1) \supset \text{Del}(A, B, r, c, d)
\end{array}$$

The first two rules capture the rootacl and delegation chain mechanisms of obtaining access, and the last two capture the intuitive order relationship between delegation depths.

We now model access control lists and two versions of capabilities within our framework, so that we may make comparisons between these models.

4. Modeling Access Control Lists (M_{acl})

Access control lists are a very commonly deployed mechanism for restricting access, useful when the "users" of the resource are known in advance. First introduced for controlling file system access in the operating system Multics [21] and in the Cambridge Titan Multi-Access System [4], access is moderated by checking for membership in the access control list (acl) associated with each object. Simply put, an entity S (subject) can access an entity O (object) if S appears in O's acl. While implementations of this mechanism may include other features like groups and priorities, they are orthogonal to the core access control decision. Thus our model for access control lists comprises of the following:

1. **World State** (WS): The World State contains a set O of objects, a set R of rights, a set S of subjects $(S \subseteq O)$, and an acl map A,

$$A : O \times R \rightarrow \mathcal{P}(S)$$

mapping each object/right pair (henceforth referred to as an (o, r) pair) uniquely to a set of subjects which are allowed that right on that object. This set of subjects thus forms the access control list (acl) for that (o, r) pair.

2. **Actions**: An action specifies a transition from one world state to another. We assume that the world state before the action, w, has components O, R, S, and A, and the corresponding components after the action are given by O', R', S' and A'. The following actions are defined, together with the world set components that are modified (see Table 1 in Appendix B for a summary of all actions):

• *Create* (object): Object s_c creates a new object o.

$$\begin{array}{rcl}
\text{Create}(s_c, o; w) & = & (O \cup \{o\}, R, S \cup \{s_c\}, A') \\
A'(o, r) & = & \left\{ \begin{array}{ll} s_c & r = r_e \\ \emptyset & r \neq r_e \end{array} \right.
\end{array}$$

The new object o is added to the set of objects O. Since o was created by s_c, we assume that s_c has the right r_e to edit o's acl; consequently s_c is added to the set of subjects S. A is updated to include an empty acl for each right r associated with o, except the r_e right.

• *Allow* (access): Allow subject s right r on object o.

$$\text{Allow}(s, o, r; w) = (O, R, S \cup \{s\}, A'), \text{ where}$$

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

The action *Allow* adds s to the set of subjects S, and updates the acl map A at (o, r) to include s, effectively adding

it to the (o, r) acl. The sets O and R are left unchanged.
• *Revoke* (access): Revoke subject s's right r on object o.

$$\text{Revoke}(s, o, r; w) = (O, R, S \ominus \{s\}, A'), \text{ where}$$

$$\begin{array}{rcl} S \ominus \{s\} & \stackrel{\text{def}}{=} & \begin{cases} S & |A^{-1}(\{s\})| \geq 2 \\ S - \{s\} & \text{otherwise} \end{cases} \\ A' & = & A[(o, r) \mapsto A((o, r)) - \{s\}] \end{array}$$

The acl map A is updated at (o, r) by removing s from that acl. The set of subjects S is modified to accurately capture the current set of objects which can access at least one (o, r) pair. We use A^{-1} to denote the natural extension of the usual inverse of map A to subsets of range(A).
• *Delete* (object): Delete object o from the system.

$$\text{Delete}(o; w) = (O - \{o\}, R, S - \{o\}, A\mid_{(O-\{o\}, R, S-\{o\})})$$

The deletion of o removes it from both the object and subject sets, while keeping R unchanged. (The set of rights R is assumed to contain all rights that may ever be associated with objects, and therefore it is retained even when no objects remain for a given right.) Clearly, everything in the system is unchanged, except that o disappears. Thus, the map A is updated by restricting its domain and co-domain to $(O - \{o\}) \times R$ and $S - \{o\}$ respectively. $A((o, r))$ is no longer defined for any $r \in R$, and o doesn't appear in any acl.
3. **Access Judgment**: Since $A((o, r))$ is the access control list associated with the object/right pair (o, r), s can access that right *iff* it belongs to the (o, r)-acl. Formally,

$$WS \vdash s \rightarrow (o, r) \stackrel{\text{def}}{=} s \in A((o, r)).$$

As an example application, we model UNIX file access control in Example C.1 (Appendix C).

4.1. A note on subjects

In the model above, we have maintained a distinction between the sets of objects and (active) subjects. Formally, we consider a subject to be an object which holds a certain right. In the case of access control lists, a subject is an object which appears on the acl of at least one object. The effect of the actions on the set of subjects should therefore be checked with respect to this definition of a subject. For example, the *Allow* action gives a right to an object, which is then added to the list of subjects as it now has an access right. For the same reason, when a *Revoke* action removes a subject from the access control list of an (o, r) pair, the set of subjects must be updated to possibly remove that object if it doesn't appear on any other acl. This is formally captured by the $\ominus$ operator. The actions for the models for capabilities are also specified to meet the above definition of subject.

Note that this definition of objects and subjects is in no way coupled with the modeling of other aspects of the access control mechanism—what is considered a subject can be defined independently by a system. Another choice could have been to stipulate that a subject is a subject irrespective of whether or not it currently has access to an (o, r) pair. In this case, we would identify the set S with the set of *active* subjects.

The distinction between the set of objects and the set of active subjects serves a technical purpose. Specifically, it provides us with a way to quantify the effects of actions in the models, especially revocation, by means of a counting argument based on the cardinality of the set of subjects. We use this to demonstrate the infeasibility of simulation in certain cases. While the same argument could be carried out if a different definition of the set of subjects were used, the definition we adopt simplifies the argument.

5. Modeling Capabilities

Capability based systems [18] provide a form of access control where the ability to access a resource is synonymous with the possession of an unforgeable *ticket* (or capability) to it. This idea can be realized in various ways. An operating system could manage all capabilities associated with a process, maintaining them in a separate store to prevent a user from forging them. Alternatively, systems such as the Communities.com E project [3] have proposed identifying capabilities with Java language pointers, relying on the Java type system to prevent users from forging capabilities.

Another view of capabilities is often used to describe a purported equivalence with access control lists. This view is based on the access matrix proposed by Lampson [17] and studied further by Harrison et al. [14]. The access matrix A is a two-dimensional matrix with object/right pairs as columns, subjects as rows, and the entry in the (i, j)th cell A^{ij} determining whether the subject in row i has access to the (o, r) pair in column j.

$$\begin{array}{c|c|c|c} & & (o,r)_i & \\ \hline & & & \\ \hline s_j & & A^{ij} & \\ \hline & & & \ddots \end{array}$$

The list of subjects in the column corresponding to (o, r) is called its access control list, and the list of (o, r) pairs in a row corresponding to a subject is its capability list. Although the capability as rows view integrates nicely with ACLs, the ticket model more accurately reflects the spirit of most capability-based proposals.

We will distinguish capabilities as tickets from the Lampson-matrix capabilities, giving a model of the former in Section 5.1 and the latter in Section 5.2. While different implementations of capability systems allow varying degrees of control over capability passing, we consider one extreme in one model and the opposite in the other. It is also possible to define other capability models in our framework, although for simplicity we do not do so in this paper.

5.1. Capabilities as unforgeable bit strings ($M_{C_{\text{ref}}}$)

Our model for this view of capabilities comprises of the following:

1. **World State** (WS): The world state comprises of a set O of objects, a set R of rights, a set S of subjects ($S \subseteq O$), a set C of capabilities, and the ticket and wallet maps T and W:

$$\begin{array}{rcl} T & : & O \times R \rightarrow \mathcal{P}(C) \\ W & : & S \rightarrow \mathcal{P}(C). \end{array}$$

$T(o, r)$ is the set of capabilities (or tickets) which can be used to access right r on object o. For a subject s, $W(s)$ denotes its capability-list [9]; s can access the (o, r) pairs for which it has a capability in this list.

A capability is intended to function as an unforgeable ticket to access a certain right on a certain object, and to that end it must be hard to fashion one given an (o, r) pair. Here, we assume the existence of a capability-generating function G. Since we may have more than one capability per object/right pair, a good capability generating function G must be collision resistant in addition to being one-way. Thus, $C \subseteq G(O \times R)$. Note that T may be identified with G on the intersection of their domains.

2. **Actions**: An action specifies a transition rule between two world sets. O, R, S, C, T, and W are the world state w's components before the action, and the primed versions are their counterparts afterwards. As before, we state what an action is supposed to do, define it in terms of a state transformation, and provide a justification that the two are equivalent.

• *Create* (object): Object s_c creates a new object o.

$$\begin{array}{rcl} \text{Create}(s_c, o; w) & = & (O \cup \{o\}, R, S', C \cup c^o_{gc}, T', W') \\ S' & = & S \cup \{s_c\} \\ T'(o,r) & = & \left\{ \begin{array}{ll} c^o_{gc} & r = r_{gc} \\ \emptyset & r \neq r_{gc} \end{array} \right. \quad r \in R \\ W'(s_c) & = & \left\{ \begin{array}{ll} \{c^o_{gc}\} & s_c \notin S \\ W(s_c) \cup \{c^o_{gc}\} & s_c \in S \end{array} \right. \end{array}$$

The creating object is given c^o_{gc}, the capability to generate any capability associated with the new object o (i.e., to call $G(o, r)$ for any $r \in R$), which can then be passed to other subjects. This parallels our justification for giving s_c the edit-acl right r_e in the acl model. Consequently, s_c is added to the set of subjects. At this point, no other objects hold this capability for o, and no other capabilities for o exist, which is reflected in the updated wallet and ticket maps W' and T' respectively.

• *Gen*erate (capability): The object o_g generates a new capability c for the object/right pair (o, r).

$$\begin{array}{rcl} \text{Gen}(o_g, c, o, r; w) & = & (O, R, S, C \cup \{c\}, T', W') \\ T' & = & T[(o, r) \mapsto T((o, r)) \cup \{c\}] \\ W' & = & W[o_g \mapsto W(o_g) \cup \{c\}] \end{array}$$

where $c = G((o, r))$. Since o_g was able to generate a capability for (o, r), it must have already had the c^o_{gc} capability, and is therefore already a subject. We assume that the ability to call G to generate a new ticket comes with the ability to cache the result, and hence c is added to o_g's wallet. Here onwards, this new capability maybe passed to other objects.

• *Pass* (capability): Subject s passes the capability $c \in W(s)$ for the object/right pair (o, r) to receiving subject r_s.

$$\text{Pass}(s, c, r_s; w) \quad = \quad (O, R, S \cup \{r_s\}, C, T, W')$$

$$W'(r_s) = \left\{ \begin{array}{ll} \{c\} & r_s \notin S \\ W(r_s) \cup \{c\} & r_s \in S \end{array} \right.$$

This action affects only the capability list (the map W) of the subject receiving the capability, r_s, which now contains the passed capability c.

• *Remove* (capability): The capability c corresponding to the object/right pair (o, r) is removed from the system.

$$\begin{array}{rcl} \text{Remove}(c, o, r; w) & = & (O, R, S', C - \{c\}, T', W') \\ T' & = & T[(o, r) \mapsto T((o, r)) - \{c\}] \\ W' & = & W[s \mapsto W(s) - \{c\}]_{s \in S'} \\ S' & = & S - \{s | W(s) = \{c\}\} \end{array}$$

The capability c is flushed out of the system, removing it from each of: the set of capabilities C, the tickets set associated with the (o, r) pair, and each subject's capability list. This operation may reduce some subjects' capability list to null, at which point they no longer should be considered subjects. (See Section 4.1.)

• *Delete* (object): The object o is deleted from the system.

$$\begin{array}{rcl} \text{Delete}(o; w) & = & (O - \{o\}, R, S - \{o\}, C', T', W') \\ C' & = & C - T((o, r)) \\ T' & = & T \mid_{(O - \{o\}, C')} \\ W' & = & W \mid_{(S - \{o\}, C')} \end{array}$$

The deleted object o is flushed from the world state, effectively resulting in retaining subsets of the component sets (O, S, C) and maps (T, W) which make no reference to it.

Note that we don't allow a *Revoke* action for capabilities. The nature of capability based systems makes it infeasible in general to implement revocation unless the *Pass* action is somehow monitored; this is formalized in Lemma 6.5.

3. **Access Judgment**: A subject s can access right r on o *iff* it possesses at least one of the tickets (capabilities) for that object/right pair. Formally,

$$WS \vdash s \rightarrow (o,r) \stackrel{\text{def}}{=} [W(s) \cap T(o,r) \neq \emptyset].$$

As an example application, we model capabilities in Amoeba [20] in Example C.2 (Appendix C).

5.2. Lampson matrix capabilities ($M_{C_{\text{row}}}$)

For purposes of comparison, we define a model for capabilities based on the rows of the Lampson access matrix. Our analysis will show that this view is *not* the same as capabilities as unforgeable bit strings. We model this view of capabilities as follows:

1. **World State** (WS): The world state comprises of a set O of objects, a set R of rights, a set S of subjects ($S \subseteq O$), and a map C

$$C : S \rightarrow \mathcal{P}(O \times R).$$

The map C associates a subject s with its capability list. However, in contrast with capabilities viewed as unforgeable bit strings, an element occurring in a capability list here is not a first class capability. More precisely, if $(o,r) \in C(s)$, then (s,o,r) is the capability, and not (o,r).

2. **Actions**: As usual, an action is a transition rule between two world states; the following actions are defined (see table 2, Appendix B for a summary). O, R, S, and C are the world state w's components before the action, and the primed versions are their counterparts afterwards.

• *Create* (object): Object s_c creates a new object o.

$$\begin{array}{rcl} \text{Create}(s_c, o; w) & = & (O \cup \{o\}, R, S \cup \{s_c\}, C') \\ C'(s_c) & = & \left\{ \begin{array}{ll} \{(o, r_e)\} & s_c \notin S \\ C(s_c) \cup \{(o, r_e)\} & s_c \in S \end{array} \right. \end{array}$$

Here r_e is the right to change capabilities for the entire object, which is given to the creating object s_c, effectively making it a subject. In other words, s_c can cause a *Grant* or *Revoke* of any capability of the form (s,o,r) for arbitrary $s \in S$ and $r \in R$.

• *Delete* (object): Object o is deleted from the system.

$$\text{Delete}(o; w) = (O - \{o\}, R, S - \{o\}, C \mid_{(S-\{o\}, O-\{o\})})$$

Flushing o from the system effectively results in retaining subsets of the world state component sets O, S and map C which make no reference to it. Restricting the domain of C to $S - \{o\}$ corresponds to removal of the o-row from the Lampson matrix, whereas restriction of the co-domain to $(O - \{o\}) \times R$ corresponds to removal of all (o,r)-columns.

• *Grant* (capability): Subject s is granted a capability to access right r on object o.

$$\text{Grant}(s, o, r; w) = (O, R, S \cup \{s\}, C[s \mapsto C(s) \cup \{(o,r)\}])$$

The addition of (o,r) to the capability list associated with s, corresponds to setting the bit in the (o,r)-column of the s-row of the Lampson matrix to 1.

• *Revoke* (capability): The capability to access the right r on object o is revoked from subject s.

$$\begin{array}{rcl} \text{Revoke}(s, o, r; w) & = & (O, R, S', C') \\ S' & = & \left\{ \begin{array}{ll} S - \{s\} & C(s) = (o,r) \\ S & C(s) \neq (o,r) \end{array} \right. \\ C' & = & C[s \mapsto C(s) - \{(o,r)\}]_{s \in S'} \end{array}$$

The removal of (o,r) from s's capability list corresponds to setting the bit in the (o,r)-column of the s-row in the Lampson matrix to 0. S is modified to capture s's new status, depending on whether the action leaves it with a null capability list.

3. **Access Judgment**: A subject s is allowed access to the (o,r) pair *iff* (o,r) belongs in the capability list of s. Formally,

$$WS \vdash s \rightarrow (o,r) \stackrel{\text{def}}{=} ((o,r) \in C(s))$$

Note that the above access is allowed exactly when the bit in the (o,r)-column of the s-row of the equivalent Lampson protection matrix is set to 1.

6. Comparing Models via Simulation Relations

Each of the access control models has been presented as a labeled transition system. For those not familiar with the concept, Appendix A explains the general concept of a labeled transition system with states $\mathcal{Q}$, set *Act* of actions, and transition relation $\mathcal{T}$. In each of our models, $\mathcal{Q}$ is the set of possible world states, the set *Act* is the set of actions defined for that model, and $\mathcal{T}$ is the transition relation implied by the action definitions. This provides us with a natural way to compare these mechanisms, namely, simulation and bisimulation relations (Appendix A) between the various semantics.

In order to compare any pair of access control mechanisms, we will try to simulate each action in one mechanism by either a single action or a sequence of actions in another. More precisely, given access control models S_A, S_C, and S_T, we present *model functors*, which are maps between the world sets of S_A, S_C, and S_T, denoted by WS_A, WS_C and WS_T respectively. If $f_{A \rightarrow C}$ is one such functor, then our intention is that $f_{A \rightarrow C}(WS_A)$ be able to simulate the changes to WS_A, within the other model. This potentially yields two model functors for every pair of mechanisms, one in each direction.

The simulation theorems, and theorems stating the nonexistence of simulations, not only allow us to compare the expressive powers of these systems, but pinpoint implementation requirements that must be met if security policies expressed in one access control vernacular are to be accurately met within a system which uses a different access control mechanism. The existence of only a weak simulation between two models imposes atomicity constraints on the implementation, ensuring that the visible states of the implementation are the corresponding weakly similar states in the model. The infeasibility of simulating specific actions relies on counting arguments; we show that for these actions, any simulating sequence of actions must depend on the size of the world state, thus violating the requirements for weak simulation (see Appendix A.)

DEFINITION 6.1 (The Access-Containment relation) *Given two models for access control M_1 and M_2, and world states WS_{M_1} and WS_{M_2}, we say that WS_{M_1} is* access-contained *in WS_{M_2} if for any $s \in S_{WS_{M_1}}$, the access decisions $WS_{M_1} \vdash_{M_1} s \to (o,r)$ and $WS_{M_2} \vdash_{M_2} s \to (o,r)$ yield the same result. We denote this by $WS_{M_1} \subseteq_{\text{acc}} WS_{M_2}$. (Note that this implies that $S_{WS_{M_1}} \subseteq S_{WS_{M_2}}$.)*

DEFINITION 6.2 (Access equivalence) *We say that world states WS_{M_1} and WS_{M_2} in models M_1 and M_2 for access control are* access-equivalent *if WS_{M_1} is access-contained in WS_{M_2} and vice versa. Access equivalence implies that both world states have the same subject sets, and exactly the same accesses are allowed in either model.*

DEFINITION 6.3 (Strong and Weak model containment) *Given models M_1 and M_2, if the access containment relation between their models is a strong simulation (Appendix A), i.e.,*

$$WS_{M_1} \subseteq_{\text{acc}} WS_{M_2} \text{ and } WS_{M_1} \overset{a_{M_1}}{\to} WS'_{M_1} \Rightarrow$$
$$\exists WS'_{M_2}.WS_{M_2} \overset{a_{M_2}}{\to} WS'_{M_2} \text{ and } WS'_{M_1} \subseteq_{\text{acc}} WS'_{M_2},$$

then we say that model M_1 is strongly contained *in or* strongly simulated *by model M_2. We denote this by $M_1 \subseteq_s M_2$. If the access containment relation is a many-step simulation (Appendix A), i.e.,*

$$WS_{M_1} \subseteq_{\text{acc}} WS_{M_2} \text{ and } WS_{M_1} \overset{a_{M_1}}{\to} WS'_{M_1} \Rightarrow$$
$$\exists WS'_{M_2}.WS_{M_2} \overset{\vec{a}_{M_2}}{\to} WS'_{M_2} \text{ and } WS'_{M_1} \subseteq_{\text{acc}} WS'_{M_2},$$

then we say that model M_1 is weakly contained *in or* weakly simulated *by model M_2. We denote this by $M_1 \subseteq_w M_2$.*

DEFINITION 6.4 (Model Equivalence) *Two access control models M_1 and M_2 are defined to be equivalent ($M_1 \cong M_2$), when*

$$WS_{M_1} \vdash_{M_1} s \to (o,r) \quad \textbf{iff} \quad WS_{M_2} \vdash_{M_2} s \to (o,r)$$

for all pair of states WS_{M_1} and WS_{M_2} in the two models, which correspond to the same real world state. In other words, M_1 and M_2 are equivalent when subject s can access the (o,r) pair in a state of model M_1 iff it can access it in the corresponding state of model M_2.

6.1. Comparing access control lists and Lampson matrix capabilities

Consider a real world system objective S which is modeled as S_A using access control lists (M_{acl}), and as S_C using the capabilities as rows view ($M_{C_{\text{ref}}}$). For any given real world state of S, there will be world sets WS_A in the first model, and $WS_{C_{\text{row}}}$ in the second model, capturing the information of interest about S. In both cases, certain actions modify the world set, and hence the current world set can be considered to be the effect of a sequence of actions, starting from an initial world state. In other words, the real world system S started with an initial state S_i, which was modeled as WS^i_A and $WS^i_{C_{\text{row}}}$ (say) in the two systems. We assume that from an access control point of view, $WS^i_A \cong WS^i_{C_{\text{row}}}$. A sequence of real world actions took the system to state S, and the representation of these actions in the two systems took WS^i_A and $WS^i_{C_{\text{row}}}$ to WS_A and $WS_{C_{\text{row}}}$ respectively. We will construct maps from M_{acl} to $M_{C_{\text{row}}}$, and from $M_{C_{\text{row}}}$ to M_{acl}, to formally capture the relationship between these two world states. The maps will be defined by induction on the sequence of steps by which the world state was arrived at.

LEMMA 6.1 $M_{\text{acl}} \subseteq_s M_{\text{c}_{\text{row}}}$

Proof. We define a functor $f_{A \to C_{\text{row}}} : WS_A \to WS_{C_{\text{row}}}$ as follows (we abbreviate $f_{A \to C_{\text{row}}}$ by f and WS_A by w_A):

$$\begin{array}{lcl} f(WS^i_A) & = & WS^i_{C_{\text{row}}} \\ f(\text{Create}(s_c, o; w_A)) & = & \text{Create}(s_c, o; f(w_A)) \\ f(\text{Delete}(o; w_A)) & = & \text{Delete}(o; f(w_A)) \\ f(\text{Allow}(s, r, o; w_A)) & = & \text{Get}(s, o, r; f(w_A)) \\ f(\text{Revoke}(s, r, o; w_A)) & = & \text{Revoke}(s, o, r; f(w_A)) \end{array}$$

We claim that f is an access containment relation between the world states of M_{acl} and $M_{C_{\text{row}}}$.

CLAIM 6.1.1 $\forall w_A . w_A \subseteq_{\text{acc}} f(w_A)$

Proof Idea. Consider the last action in the evolution path of w_A, and show that the access-containment relation between the world states w_A and $f(w_A)$ holds after the action if it holds prior to it. Showing this for all possible actions of M_{acl} provides the different cases of this inductive proof. (See Appendix D for the complete proof.)

For each transition $w_A \overset{a_A}{\to} w'_A$ in M_{acl}, $f(w'_A)$ is defined in terms of exactly one action a_C and $f(w_A)$. For example, the $(\textbf{Allow})$ case of the definition has $a_A = \mathit{Allow}$

and $a_C = Get$. Hence f is a strong simulation, and hence $M_{\text{acl}} \subseteq_s M_{\text{c}_{\text{row}}}$. $\diamond$

LEMMA 6.2 $M_{\text{c}_{\text{row}}} \subseteq_s M_{\text{acl}}$

Proof. The map f created above is a bijection, and we can prove $\forall w_C.w_C \subseteq_{\text{acc}} f^{-1}(w_C)$ in an identical manner. The result follows. (See Appendix D for a definition of f^{-1}.) $\diamond$

THEOREM 6.1 $M_{\text{c}_{\text{row}}} \cong M_{\text{acl}}$

Proof. Follows directly from the definition for access containment, and the above two lemmas which show that world states of one model are access contained in the corresponding world states of the other. Thus, as mechanisms for access control, capabilities viewed as rows of the Lampson protection matrix and access control lists are equivalent, i.e., strongly bisimilar. ■

6.2. Comparing access control lists and capabilities as references

The fact that the *Remove* action in M_{acl} has no real counterpart in $M_{C_{\text{ref}}}$ leads to the following results.

LEMMA 6.3 $M_{\text{acl}} \subseteq_w M_{C_{\text{ref}}}$

Proof. We define a correspondence functor $f_{A \to C_{\text{ref}}} : WS_A \to WS_{\text{C}_{\text{ref}}}$ as follows (we abbreviate $f_{A \to C_{\text{ref}}}$ by f and WS_A by w_A):

$$\begin{array}{lcl} f(WS^i_A) & = & WS^i_{\text{C}_{\text{ref}}} \\ f(\text{Create}(s_c, o; w_A)) & = & \text{Create}(s_c, o; f(w_A)) \\ f(\text{Delete}(o; w_A)) & = & \text{Delete}(o; f(w_A)) \\ f(\text{Allow}(s, r, o; w_A)) & = & \text{Pass}(\mathbf{o_g}, c, s; \\ & & \quad \text{Gen}(\mathbf{o_g}, c, o, r; f(w_A))) \\ f(\text{Revoke}(s, r, o; w_A)) & = & \text{Remove}(c_s, o, r; f(w_A)) \end{array}$$

where c_s is the capability that s has to the (o, r) pair ($c_s = W(s) \cap T((o, r))$). We claim that f is an access containment relation between the world states of M_{acl} and $M_{C_{\text{ref}}}$.

CLAIM 6.3.1 $\forall w_A.w_A \subseteq_{\text{acc}} f(w_A)$

Proof Sketch. Again, we consider the last transition in the evolution path of w_A, and show that for all possible actions, the access-containment relation holds between w_A and $f(w_A)$ after the action if it holds before. Hence, by induction, we are done.

Since the *Allow* action of M_{acl} requires two $M_{C_{\text{ref}}}$ actions to simulate it, $M_{\text{acl}} \not\subseteq_s M_{C_{\text{ref}}}$. This can also be inferred by considering the actions needed in $M_{C_{\text{ref}}}$ to simulate the first subject that is given a (non edit-acl) right to an object in M_{acl}.

Note that the equivalent capability system should hand out fresh capabilities for each *Allow* action authorized by the object in M_{acl}, as specified in the (**Allow**) case above. Failing that, it would be hard to model acls with a capability implementation because of the infeasibility of determining which subjects a revoked capability corresponds to. $\diamond$

LEMMA 6.4 $M_{\text{c}_{\text{ref}}} \not\subseteq_s M_{\text{acl}}$

In other words, $\exists a \in Act_{M_{\text{C}_{\text{ref}}}}$ such that

$$w_C \subseteq_{\text{acc}} f(w_C) \text{ and } w_C \xrightarrow{a} w'_C \text{ but } \nexists a'.f(w_C) \xrightarrow{a'} f(w'_C).$$

where f is the correspondence functor between $M_{C_{\text{ref}}}$ and M_{acl}.
Proof. Consider the following world state w_C in $M_{C_{\text{ref}}}$.

$$w_C = \\ (\{o, o_s, s_1, s_2, s_3\}, \{r, r_{gc}\}, \{o_s, s_1, s_2, s_3\}, \{c^o_{gc}, c_a, c_b\}, \\ \begin{bmatrix} (o, r_{gc}) & \mapsto & c^0_{gc} \\ (o, r) & \mapsto & \{c_a, c_b\} \\ & \vdots & \end{bmatrix}, \begin{bmatrix} o_s & \mapsto & \{c^o_{gc}, c_a, c_b\} \\ s_1 & \mapsto & \{c_a\} \\ s_2 & \mapsto & \{c_a\} \\ s_3 & \mapsto & \{c_b\} \end{bmatrix})$$

One may imagine that this state was the result of the superuser o_s generating capabilities c_a and c_b to object o and handing them out to subjects s_1 and s_3 respectively. Subsequently, subject s_1 passed ticket c_a to subject s_2. Clearly, the state w_A of M_{acl} which corresponds to w_C is given by

$$w_A = (\{o, o_s, s_1, s_2, s_3\}, \{r, r_e\}, \{o_s, s_1, s_2, s_3\}, \\ \begin{bmatrix} (o, r_e) & \mapsto & o_s \\ (o, r) & \mapsto & \{s_1, s_2, s_3\} \\ & \vdots & \end{bmatrix}).$$

Removing capability c_a from w_C (by the $\text{Remove}(c_a, o, r)$ action) results in a state w'_C whose corresponding state in M_{acl}, w'_A (say), cannot be reached from w_A by any single action of M_{acl}. Thus w'_A is reached by the actions $\text{Revoke}(s_1, r, o)$ and $\text{Revoke}(s_2, r, o)$ in any order. $\diamond$

LEMMA 6.5 $M_{\text{c}_{\text{ref}}} \not\subseteq_w M_{\text{acl}}$

Proof. We consider the "cost" associated with carrying out an action in either model, and show that in order to reach a corresponding state in M_{acl}, after a *Remove* action in $M_{C_{\text{ref}}}$, requires a number of actions proportional to the size of the set of objects. Consider the following state in $M_{C_{\text{ref}}}$:

$$w_C = \\ (\{o, o_s, s_1, \ldots, s_n\}, \{r, r_{gc}\}, \{o_s, s_1, \ldots, s_n\}, \{c^o_{gc}, c\}, \\ \begin{bmatrix} (o, r_{gc}) & \mapsto & c^0_{gc} \\ (o, r) & \mapsto & \{c\} \\ & \vdots & \end{bmatrix}, \begin{bmatrix} o_s & \mapsto & \{c^o_{gc}, c\} \\ s_1 & \mapsto & \{c\} \\ & \vdots & \\ s_n & \mapsto & \{c\} \end{bmatrix})$$

Here the capability c to (o, r) pair is held by subjects $s_1, \ldots, s_n$, which hold no other capabilities. A $\text{Remove}(c, o, r | w_C)$ action reduces $S_{\mathrm{C_{ref}}}$ to $\{o_s\}$, a reduction in size by $\Theta(|S|)$. Since each of the actions in M_{acl} changes the set of subjects by at most one, the above action needs $\Theta(|S|)$ actions in M_{acl} to simulate it. Thus any simulating sequence necessarily depends on w_C, and fails to be a witness for a weak simulation. Hence, $M_{\mathrm{C_{ref}}} \not\subseteq_w M_{\mathrm{acl}}$. $\diamond$

6.3. Access control lists and Trust Management

It is not possible to simulate the delegation feature of trust management in a way that allows for controlled revocation, leading to an asymmetric relationship between M_{acl} and M_{tm}. The following results express this formally.

LEMMA 6.6 $M_{\mathrm{acl}} \subseteq_s M_{\mathrm{tm}}$

Proof. We define a correspondence functor $f_{A \to T} : WS_A \to WS_T$ as follows (we abbreviate $f_{A \to T}$ by f and WS_A by w_A):

$$
\begin{array}{lcl}
f(WS_A^i) & = & WS_T^i \\
f(\text{Create}(s_c, o; w_A)) & = & \text{Create}((s_c, o; f(w_A)) \\
f(\text{Delete}(o; w_A)) & = & \text{Delete}((o; f(w_A)) \\
f(\text{Allow}(s, o, r; w_A)) & = & \text{Add}(o, r, s, 0; f(w_A)) \\
f(\text{Revoke}(s, o, r; w_A)) & = & \text{Remove}(o, r, s, 0; f(w_A))
\end{array}
$$

In other words, by setting the delegation depth to 0, thereby rendering any delegation actions ineffective, we can embed M_{acl} into M_{tm}. We claim that f is an access containment relation between the world states of M_{acl} and M_{tm}.

CLAIM 6.6.1 $\forall w_A . w_A \subseteq_{\mathrm{acc}} f(w_A)$

Proof Idea. The proof strategy is identical to that of Claim 6.1.1, and considers the last action in the evolution path of w_A.

For each transition $w_A \xrightarrow{a_A} w'_A$ in M_{acl}, $f(w'_A)$ is defined in terms of exactly one action $a_T \in M_{\mathrm{tm}}$ and $f(w_A)$. Hence f is a strong simulation, and $M_{\mathrm{acl}} \subseteq_s M_{\mathrm{tm}}$. $\diamond$

LEMMA 6.7 $M_{\mathrm{tm}} \not\subseteq_s M_{\mathrm{acl}}$

Proof. Consider the following world state w_T in M_{tm}.

$$
w_T = (\{o, o_s, s_1, s_2\}, \{r_e, r\}, \begin{bmatrix} (o, r_e) & \mapsto & (o_s, 1) \\ (o, r) & \mapsto & (o_s, 2) \\ & \vdots & \end{bmatrix},
\begin{bmatrix} (o_s, r, o) & \mapsto & \{(s_1, 1)\} \\ (s_1, r, o) & \mapsto & \{(s_2, 0)\} \\ & \vdots & \end{bmatrix})
$$

One may imagine that this state was the result of a superuser o_s delegating its right r on object o to subject s_1, who further delegated it to s_2. The state w_A in M_{acl} which corresponds to w_T is given by:

$$
w_A = (\{o, o_s, s_1, s_2\}, \{r_e, r\}, \{o_s, s_1, s_2\},
\begin{bmatrix} (o, r_e) & \mapsto & o_s \\ (o, r) & \mapsto & \{o_s, s_1, s_2\} \\ & \vdots & \end{bmatrix})
$$

The action $\text{Remove}(o, r, o_s, 2; w_T)$ in M_{tm} cannot be simulated by any single action of M_{acl}, but requires both $\text{Revoke}(s_1, o, r)$ and $\text{Revoke}(s_2, o, r)$. Intuitively, the removal of an object from a rootacl (or the revocation of a delegation) renders several previously allowed accesses void, and identifying these denied accesses can take $\Theta(|S_{M_{\mathrm{acl}}}|)$ actions in the worst case. The next results states this formally. $\diamond$

LEMMA 6.8 $M_{\mathrm{tm}} \not\subseteq_w M_{\mathrm{acl}}$

Proof. As we did in Lemma 6.5, we consider the cost associated with actions in the two models, and show that the Remove action of M_{tm} can require upto $\Theta(|S_{M_{\mathrm{acl}}}|)$ actions in M_{acl} to reach an access equivalent state. This may be seen by generalizing the world state in the last lemma to contain a delegation chain of depth n. As a result, any candidate sequence of actions for simulating the Remove action depends on the corresponding state in M_{acl}, and thus we are done. $\diamond$

6.4. Comparing Trust Management and capabilities as references

Delegation in a trust management style of access control provides for bounds on propagation of access rights, a property which doesn't hold true for capabilities. In addition, it is possible to meaningfully revoke access anywhere in a delegation chain for trust management, in contrast to its infeasibility for capabilities. We formalize this intuition below.

LEMMA 6.9 $M_{\mathrm{C_{ref}}} \subseteq_s M_{\mathrm{tm}}$

Proof. We define a functor $f_{C_{\mathrm{ref}} \to T} : WS_{\mathrm{C_{ref}}} \to WS_T$ as follows (we abbreviate $f_{C_{\mathrm{ref}} \to T}$ by f and $WS_{\mathrm{C_{ref}}}$ by w_C):

$$
\begin{array}{lcl}
f(WS_{\mathrm{C_{ref}}}^i) & = & WS_T^i \\
f(\text{Create}(s_c, o; w_C)) & = & \text{Create}(s_c, o; f(w_C)) \\
f(\text{Gen}(o_g, c, o, r; w_C)) & = & \text{Add}(o, r, o_g, \infty; f(w_C)) \\
f(\text{Pass}(s, c, r_s; w_C)) & = & \text{Delegate}(s, o, r, r_s, \infty; \\
& & \hfill f(w_C)) \\
f(\text{Remove}(c, o, r; w_C)) & = & \text{Remove}(o, r, s_c, \infty; \\
& & \hfill f(w_C)) \\
f(\text{Delete}(o; w_C)) & = & \text{Delete}(o; f(w_C))
\end{array}
$$

We claim that f is an access containment relation between the world states of $M_{C_{\mathrm{ref}}}$ and M_{tm}.

CLAIM 6.9.1 $\forall w_C . w_C \subseteq_{\text{acc}} f(w_C)$

Proof Idea. As before, we consider the last action in the evolution path of w_C, and show that the access-containment relation between the world states w_C and $f(w_C)$ holds after the action if it holds prior to it. Showing this for all possible actions of $M_{C_{\text{ref}}}$ provides the different cases of this inductive proof.

Since each action in $M_{C_{\text{ref}}}$ is simulated by exactly one action of M_{tm}, $M_{\text{c}_{\text{ref}}} \subseteq_s M_{\text{tm}}$. $\diamond$

LEMMA 6.10 $M_{\text{tm}} \not\subseteq_w M_{\text{c}_{\text{ref}}}$

Proof. Consider the following world state w_T of M_{tm}.

$$w_T =$$
$$(\{o, o_s, s_1, \ldots, s_n\}, \{r, r_e\}, \left[\begin{array}{ccc} (o, r_e) & \mapsto & \{(o_s, 1)\} \\ & \vdots & \end{array}\right],$$
$$\left[\begin{array}{ccc} (o_s, r, o) & \mapsto & (s_1, n-1) \\ (s_1, r, o) & \mapsto & (s_2, n-2) \\ & \vdots & \\ (s_{n-1}, r, o) & \mapsto & (s_n, 0) \end{array}\right])$$

The following world state w_C of $M_{C_{\text{ref}}}$ is access equivalent to the above.

$$w_C =$$
$$(\{o, o_s, s_1, \ldots, s_n\}, \{r, r_e\}, \{o_s, s_1, \ldots, s_n\}, \{c^o_{gc}, c\},$$
$$\left[\begin{array}{ccc} (o, r_{gc}) & \mapsto & c^o_{gc} \\ (o, r) & \mapsto & c \end{array}\right], \left[\begin{array}{ccc} o_s & \mapsto & \{c^o_{gc}, c\} \\ s_1 & \mapsto & \{c\} \\ & \vdots & \\ s_n & \mapsto & \{c\} \end{array}\right])$$

In order to simulate the action $\text{Revoke}(o_s, o, r, s_{n-1}, 1 | w_T)$ within $M_{C_{\text{ref}}}$, the capability c must be removed from s_n, and s_n only. This requires $\Theta(|S_{\text{M}_{\text{tm}}}|)$ *Pass* actions to propagate the new capability to the subjects $s_1, \ldots, s_{n-1}$, making any candidate simulating sequence dependent on w_C. Hence, $M_{\text{tm}} \not\subseteq_w M_{\text{c}_{\text{ref}}}$. $\diamond$

6.5. Interpretation of results

The results of Sections 6.1–6.4 place on a formal footing our expectations about these access control mechanisms. The models we consider have actions for creating new objects, granting access to an object, delegating or transferring access, and revoking access to an object. Considering all of these actions, access control lists are equivalent to capabilities, when capabilities are regarded as rows of an access control matrix. This is intuitively reasonable, as acls are just the columns of the matrix. However, when properties of the "unforgeable ticket" implementation of capabilities are taken into account, capabilities can weakly (one-to-many) simulate access control lists, but not conversely. Trust management, modeled here without keys or name spaces, can strongly (one-to-one) simulate the other mechanisms, providing a tractable compromise between unrestricted capability passing from the capability model and easy revocation provided by access control lists. This comparison is summarized in Figure 1.

The difference between strong (one-to-one) and weak (one-to-many) simulations is essentially atomicity of transactions. In a strong simulation, one action is simulated by one visible action whereas in a weak simulation, one action may be simulated by more than one visible action. If multiple visible actions are used to achieve the same end as achieved by a single visible action in another model, then an adversary interacting with the system may be able to interleave some of its own actions. While we have not investigated any potential attacks, we believe that when only weak simulation is possible (as proved in several cases), some form of forced atomicity is required to achieve equivalence. In common terms, if the functionality of access control lists is desired within a capability-based system, for example, then some locking mechanism must be added to the capability system in order to accomplish some actions. This may be feasible if the system is centralized or implemented on a sequential processor, or infeasible in a distributed setting.

The key actions that distinguish these three mechanisms are revocation and delegation. Each mechanism operates in the context of a system configuration which determines the feasibility of these actions. The model for access control lists provides centralized control, thereby making revocation trivial, and delegation illegal. Capability systems modeled as unforgeable references present the other extreme, where delegation is trivial, and revocation is infeasible. The trust management model is able to simulate both these systems by setting the delegation depth to one of two extremes: 0 or ∞. In the general case, trust management systems provide a feasible revocation mechanism, since an access request is tagged with all the nodes along the delegation chain. Our specification of the access judgment in this model (Section 3) assumes that the delegation map D is available globally, so that the effect of local revocations are reflected in this global data structure. In practice, this points towards the need to ensure "freshness" of credentials, by means such as leases for example. A resource may also check for recent revocations, with all the nodes along a delegation chain specified in an access request. To simulate this behavior in a capability system, one would have to tag each *Pass* action with the identity of the sender, or otherwise enforce that an access request to a resource came back to it through the same *Pass* chain that gave the subject the capability. We may thus view a delegation credential in a trust management system to be the creation of a new *history-dependent capability*, created by the delegator, and

$$
\begin{tikzcd}
M_{C_{ref}} \arrow[r, shift left, "s"] \arrow[d, shift left, "w", "/" marking] & M_{tm} \arrow[l, shift left, "w", "/" marking] \arrow[d, shift left, "w", "/" marking] \\
M_{acl} \arrow[u, shift left, "w"] \arrow[r, shift left, "s"] & M_{C_{row}} \arrow[l, shift left, "s"] \arrow[u, shift left, "s"]
\end{tikzcd}
$$

(1)

$$
\begin{tikzcd}
M_{C_{ref}} \arrow[r, shift left, "s"] \arrow[d, shift left, "s"] & M_{tm} \arrow[l, shift left, "w", "/" marking] \arrow[d, shift left, "w", "/" marking] \\
M_{acl} \arrow[u, shift left, "w"] \arrow[r, shift left, "s"] & M_{C_{row}} \arrow[l, shift left, "s"] \arrow[u, shift left, "s"]
\end{tikzcd}
$$

(2)

$$
\begin{tikzcd}
M_{C_{ref}} \arrow[r, shift left, "w", "/" marking] \arrow[d, shift left, "w", "/" marking] & M_{tm} \arrow[l, shift left, "w"] \arrow[d, shift left, "s"] \\
M_{acl} \arrow[u, shift left, "w"] \arrow[r, shift left, "s"] & M_{C_{row}} \arrow[l, shift left, "s"] \arrow[u, shift left, "s"]
\end{tikzcd}
$$

(3)

$$
\begin{tikzcd}
M_{C_{ref}} \arrow[r, shift left, "s"] \arrow[d, shift left, "s"] & M_{tm} \arrow[l, shift left, "w"] \arrow[d, shift left, "s"] \\
M_{acl} \arrow[u, shift left, "w"] \arrow[r, shift left, "s"] & M_{C_{row}} \arrow[l, shift left, "s"] \arrow[u, shift left, "s"]
\end{tikzcd}
$$

(4)

Comparing access control mechanisms
Figure (1): All actions
Figure (2): Without capability passing/revocation
Figure (3): Without delegation
Figure (4): Without revocation or delegation

usable *only* by the delegatee. The access judgment for trust management may now be viewed as the judgment for simple capability systems, with an additional forward temporal consistency check to see if current beliefs of nodes in the chain match the ones existing at the time the capability was issued.

Three additional figures show how the relationships change if we focus on specific subsets of actions. Ignoring revocation, the access control list and capabilities-as-rows models can strongly simulate capabilities-as-tickets, with other relationships unchanged. Without delegation, trust management becomes equivalent to access control lists and capabilities-as-rows, and weakly simulable by, but not equivalent to, capabilities-as-tickets, with other relationships unchanged. Finally, if revocation and delegation are ignored, then all models become equivalent as simple mechanisms for granting and checking access to objects.

7. Conclusions and Further Work

Using a framework based on abstract system states, state transitions, and logical deduction of access control judgments, we compare four approaches to access control: access control lists, two forms of capability mechanisms, and trust management. A general conclusion is that, in a formal sense, trust management combines the strong points of access control lists and capability systems. Intuitively, this is because trust management allows subjects to delegate rights to objects in a revocable manner.

The framework and comparison techniques used are general enough to analyze a variety of other access control mechanisms; we hope that they will be useful in evaluating new mechanisms, especially hybrids drawing on the strengths of pre-existing schemes. The analysis of these mechanisms with only some active actions allows us to isolate and better understand the contribution of a certain feature to the overall strengths and weaknesses of a scheme. A distinction between one-to-one and one-to-many simulations between these mechanisms point to (and, hopefully, help avoid) possible pitfalls and security loopholes in retrofitting a particular security policy not originally meant for a particular security mechanism.

In particular, we have used our model to define and clarify the equivalence between access control lists and capabilities, showing how capabilities viewed as rows of the Lampson access matrix, and the more honest capabilities-as-tickets view, differ in their relation to each other and to access control lists. Our specification of trust management systems shows, in a formal manner, how the depth of delegation can be varied to capture both the behavior of access control lists and capabilities. In the general case, trust management systems can provide feasible revocation, and we may identify trust credentials with history-dependent capabilities.

There are a number of promising directions for further investigation. One particular area of interest is to incorporate naming into the comparison. Proposed trust management systems include hierarchical and local namespaces. The functional behavior of these features could be evaluated, in comparison with other mechanisms, using the general approach suggested in this paper. In a forthcoming paper, we model the naming aspects of distributed trust management systems in a manner that composes well with our analysis of the core access control mechanism here. Another issue is the reliance on an external authentication mechanism. Access control lists, for example, list subjects that are allowed access and therefore rely on some authentication mechanism to determine the identity of a subject requesting access. Trust management and capability-as-ticket systems use alternate mechanisms which do not rely on the same form of external authentication mechanism. Perhaps incorporating these issues will provide further insight into the relative strengths and possible shortcomings of emerging trust management systems.

Acknowledgments

We thank Peter Neumann and the anonymous referees for helpful comments on this paper.

References

[1] M. Abadi. On SDSI's linked local name spaces. *Journal of Computer Security*, 6:3–21, 1998.

[2] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *TOPLAS*, 15(4):706–734, Sept. 1993.

[3] D. Barnes, C. Morningstar, D. Bornstein, G. Freeman, and E. Messick. Original-E. `http://www.erights.org`.

[4] D. W. Barron, A. G. Fraser, D. F. Hartley, B. Landy, and R. M. Needham. File handling at Cambridge University. In *AFIPS Conference Proceedings, Volume 30, 1967 Spring Joint Computer Conference*, pages 163–167, Washington, DC, USA, 1967. Thompson Books.

[5] Blaze, Feigenbaum, and Strauss. Compliance checking in the PolicyMaker trust management system. In *FC: International Conference on Financial Cryptography*. LNCS, Springer-Verlag, 1998.

[6] M. Blaze, J. Feigenbaum, and A. D. Keromytis. KeyNote: Trust management for public-key infrastructures. *Lecture Notes in Computer Science*, 1550:59–63, 1999.

[7] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 1996. IEEE Computer Society, Technical Committee on Security and Privacy, IEEE Computer Society Press.

[8] Y.-H. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss. REFEREE: Trust management for Web applications. *Computer Networks and ISDN Systems*, 29(8–13):953–964, Sept. 1997.

[9] J. B. Dennis and E. C. Van Horn. Programming semantics for multiprogrammed computations. *Communications of the ACM*, 9(3):143–155, Mar. 1966. Originally presented at the Proceedings of the ACM Programming Language and Pragmatics Conference, August 8–12, 1965.

[10] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI certificate theory. Available at `http://www.clark.net/pub/cme/theory.txt`.

[11] S. Ganta. *Expressive Power of Access Control Models Based on Propagation of Rights*. PhD thesis, George Mason University, 1996.

[12] L. Gong. A secure identity-based capability system. In *Proc. 1989 IEEE Symposium on Research in Security and Privacy, IEEE Computer Society Press.*, pages 56–63, May 1989.

[13] J. Halpern and R. van der Meyden. A logic for SDSI's linked local name spaces. *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, pages 111–122, 1999.

[14] M. Harrison, W. Ruzzo, and J. Ullman. Protection in operating systems. In *Communications of the ACM*, pages 461–471. ACM, Aug. 1976.

[15] J. H. Howard. An overview of the Andrew file system. In *Proceedings of the USENIX Winter 1988 Technical Conference*, pages 23–26, Berkeley, CA, 1988. USENIX Association.

[16] R. Y. Kain and C. E. Landwehr. On access checking in capability-based systems. *IEEE Transactions on Software Engineering*, 13(2):202–207, February 1987.

[17] B. Lampson. Protection. In *Proceedings of the 5th Annual Princeton Conference on Information Sciences and Systems*, pages 437–443, Princeton University, 1971.

[18] H. M. Levy. *Capability-Based Computer Systems*. Digital Press, 1984.

[19] N. Li. Local names in SPKI/SDSI. *Proceedings of the 13th Computer Security Foundations Workshop*, pages 2–15, July 2000.

[20] S. J. Mullender and A. S. Tanenbaum. The design of a capability-based distributed operating system. *The Computer Journal*, 29(4):289–299, Aug. 1986.

[21] E. I. Organick. *The Multics System: An Examination of Its Structure*. MIT Press, Cambridge, MA, USA, 1972.

[22] R. Sandhu. The typed access matrix model. In *Proc. of the 11th IEEE Symp. on Security and Privacy.*, pages 122–136, 1992.

[23] R. Sandhu and S. Ganta. On testing for absence of rights in access control models. In *Proc. of the 6th IEEE Computer Security Foundations Workshop*, pages 109–118, Franconia, NH, 1993.

[24] R. Sandhu and S. Ganta. On the expressive power of the unary transformation model. In D. Gollmann, editor, *Computer Security - ESORICS 94: Third European Symposium on Research in Computer Security*, volume 875 of *Lecture Notes in Computer Science*, Brighton, UK, November 1994. Springer-Verlag.

[25] R. Sandhu and S. Ganta. On the minimality of testing for rights in transformation models. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, pages 230–241, Oakland, CA, May 1994.

A. Simulation and Bisimulation relations

A Labeled Transition System (LTS) over a set of actions *Act* is a pair $(\mathcal{Q}, \mathcal{T})$ consisting of

1. A set of states $\mathcal{Q}$, and
2. A ternary relation $\mathcal{T} \subseteq (\mathcal{Q} \times Act \times \mathcal{Q})$ called the transition relation.

Elements (p, α, p') of the transition relation are also denoted by $p \stackrel{\alpha}{\to} p'$.

DEFINITION A.1 *Strong (one-step) simulation and bisimulation*

Let $(\mathcal{Q}, \mathcal{T})$ be an LTS over a set of actions *Act* ($\alpha \in Act$), and let S be a binary relation over $\mathcal{Q}$. Then S is called a strong simulation over $(\mathcal{Q}, \mathcal{T})$ if, whenever pSq,

$$\text{if } p \stackrel{\alpha}{\to} p', \text{ then there exists } q' \in \mathcal{Q} \text{ such that } q \stackrel{\alpha}{\to} q' \text{ and } p'Sq'.$$

We say that q strongly simulates p if there exists a strong simulation S such that pSq.

A binary relation S over $\mathcal{Q}$ is said to be a strong bisimulation over the LTS $(\mathcal{Q}, \mathcal{T})$ if both S and its converse are strong simulations. We say that p and q are strongly bisimilar or strongly equivalent, $p \sim q$, if there exists a strong bisimulation S such that pSq.

DEFINITION A.2 *Weak (many-step) simulation and bisimulation*

Let $(\mathcal{Q}, \mathcal{T})$ be an LTS over a set of actions *Act* ($\alpha \in Act$, $\vec{\alpha} \in Act^\star$), and let S be a binary relation over $\mathcal{Q}$. Then S is called a many-step simulation over $(\mathcal{Q}, \mathcal{T})$ if, whenever pSq,

$$\text{if } p \stackrel{\alpha}{\to} p', \text{ then there exists } q' \in \mathcal{Q} \text{ such that } q \stackrel{\vec{\alpha}}{\to} q' \text{ and } p'Sq'.$$

We say that q simulates p in many steps if there exists a many-step simulation S such that pSq. It is assumed that $\vec{\alpha}$ only depends on α and is independent of p and q. In other words, the action α in the first LTS is always simulated by the sequence of actions $\vec{\alpha}$ in the second LTS.

B. Models for access control mechanisms

Tables 1 to 4 summarize the models for ACLs, two versions of capabilities, and trust management. Note that only changes to the world state components are specified; a $|$ in a table entry denotes restriction of the corresponding component in the obvious way.

C. Examples

EXAMPLE C.1 (Access Control Lists) *Unix File Access*

Classical Unix operating systems use a restricted, coarse grained form of access control lists to regulate access to various system resources. For example, each Unix file is associated with an *owner* and a *group*. (The group associated with the file may be different from the group the owner of the file belongs to.) Everyone else belongs in a category called *other*. Access to a certain right associated with the file (read (r), write (w), or execute(x)) is moderated via an access control list expressed as a vector. A "-" in the vector indicates no access, whereas "r", "w", or "x" implies access to the corresponding right. For example, user `joe` can read and write the file `foo` below, but not execute it.

$$\texttt{-}\ \underbrace{\texttt{rw-}}_{\substack{\texttt{joe}\text{'s}\\ \text{rights}}}\ \underbrace{\texttt{rw-}}_{\substack{\texttt{mail}\\ \text{group's}\\ \text{rights}}}\ \underbrace{\texttt{---}}_{\substack{\text{for}\\ \text{everyone}\\ \text{else}}}\quad \texttt{joe}\ \ \texttt{mail}\ \ \texttt{foo}$$

Thus, a Unix file is associated with a vector of nine bits (for read, write, and execute rights) for its owner, group, and for everyone else.

$$f : \underbrace{b_r^o b_w^o b_x^o}_{\text{owner}} \underbrace{b_r^g b_w^g b_x^g}_{\text{group}} \underbrace{b_r^e b_w^e b_x^e}_{\substack{\text{everyone}\\ \text{else}}}$$

Here o denotes the owner of the file f, g is the group associated with the file, and e stands for everyone else. Without loss of generality from the point of view of this modeling, we consider the read right on Unix files.

Unix specifies that a user s can read file f if it is either the owner and the owner has read permission, or it belongs to the group g, and the group has read permission, or if everyone has read permission on the file, in that order. Formally, read access is the value of the expression:

$$\begin{array}{lll} \text{if} & s = o & \text{then} \quad b_r^o \\ & & \text{else} \\ \text{if} & s \in g & \text{then} \quad b_r^g \\ & & \text{else} \quad b_r^e \end{array}$$

which is equivalent to $(s = o \wedge b_r^o) \vee (s \neq o \wedge ((s \in g \wedge b_r^g) \vee (s \notin g \wedge b_r^e)))$.

Any complete model of Unix will include constructs (users, groups, locks, system bits) over and above those in

	O	R	S	A
Create(s_c, o)	$\cup\{o\}$		$\cup\{s_c\}$	$(o, r_e) \mapsto s_c$ $(o, r) \mapsto \emptyset$
Allow(s, o, r)			$\cup\{s\}$	$(o, r) \mapsto A((o, r)) \cup \{s\}$
Revoke(s, o, r)			$\ominus\{s\}$	$(o, r) \mapsto A((o, r)) - \{s\}$
Delete(o)	$-\{o\}$		$-\{o\}$	$\vert$

Table 1. Access Control Lists

	O	R	S	C
Create(s_c, o)	$\cup\{o\}$		$\cup\{s_c\}$	$s_c \mapsto \begin{cases} \{(o, r_e)\} & s_c \notin S \\ C(s_c) \cup \{(o, r_e)\} & s_c \in S \end{cases}$
Grant(s, o, r)			$\cup\{s\}$	$s \mapsto C(s) \cup \{(o, r)\}$
Revoke(s, o, r)			$\ominus\{s\}$	$s \mapsto C(s) - \{(o, r)\}$
Delete(o)	$-\{o\}$		$-\{o\}$	$\vert$

Table 2. Lampson matrix capabilities

	O	R	S	C	T	W
Create(s_c, o)	$\cup\{o\}$		$\cup\{s_c\}$	$\cup\{c^o_{gc}\}$	$(o, r_{gc}) \mapsto c^o_{gc}$ $(o, r) \mapsto \emptyset$	$s_c \mapsto \begin{cases} \{c^o_{gc}\} & s_c \notin S \\ W(s_c) \cup \{c^o_{gc}\} & s_c \in S \end{cases}$
Gen(o_g, c, o, r)				$\cup\{c\}$	$(o, r) \mapsto T((o, r)) \cup \{c\}$	$o_g \mapsto W(o_g) \cup \{c\}$
Pass(s, c, r_s)			$\cup\{r_s\}$			$r_s \mapsto \begin{cases} \{c\} & r_s \notin S \\ W(r_s) \cup \{c\} & r_s \in S \end{cases}$
Remove(c, o, r)			$\ominus\{s\}$		$(o, r) \mapsto T((o, r)) - \{c\}$	$-\{c\}$
Delete(o)	$-\{o\}$		$-\{o\}$	$-T((o, r))$	$\vert$	$\vert$

Table 3. Capabilities as unforgeable bit strings

	O	R	A	D
Create(o_c, o)	$\cup\{o\}$		$(o, r_e) \mapsto (o_c, 1)$ $(o, r) \mapsto \emptyset$	$(s, r, o) \mapsto \emptyset$
Add(o, r, o_s, d)			$(o, r) \mapsto A((o, r)) \cup \{o_s, d\}$	
Remove(o, r, o_s, d)			$(o, r) \mapsto A((o, r)) - \{o_s, d\}$	
Delegate(o_s, o, r, o_d, d)				$(o_s, r, o) \mapsto D((o_s, r, o)) \cup \{o_d, d\}$
Revoke(o_s, o, r, o_d, d)				$(o_s, r, o) \mapsto D((o_s, r, o)) - \{o_d, d\}$
Delete(o)	$-\{o\}$		$\vert$	$\vert$

Table 4. Trust Management

our model for access control. We assume the existence of the partial maps

$$\text{Owner} : O \to O, \text{ and}$$

$$G(\text{Group}) : O \to \mathcal{P}(O),$$

and the set $E(\subseteq O)$ (for "everyone else"). The intention is that Owner(f) be the object corresponding to the owner of file f, and $G(f)$ be the set of objects in the group associated with the file f. The file f belongs in the set of objects, and the above maps are partial because they make sense only for files (actually other Unix entities as well, but certainly not all of them).

Note that the mechanism for Unix file access, like all access control list implementations, separates the access control question into

- mapping the subject s to the subjects of ACL entries (in the case of Unix, determining whether $s = \text{Owner}(f)$, or $s \in G(f)$), and
- determining the precedence of the ACL entries. In Unix, there is an if-then-else ordering of tests on the access bits. Thus, if $s = o$ and $b_r^o = 0$, then access should be denied even if $s \in g$ and $b_r^g = 1$.

The first of these two is modeled with access control list maps for each of the three bits, A^o, A^g and A^e. Clearly,

$$\begin{array}{rcrcl} A^o(f,r) & = & \{\text{Owner}(f)\} & \wedge & b_r^o \\ A^g(f,r) & = & G(f) & \wedge & b_r^g \\ A^e(f,r) & = & E & \wedge & b_r^e \end{array}$$

where conjunction is interpreted as the entire set or nothing depending on the access control bit. Combining this with the if-then-else construct, we get the formal expression for when a Unix subject s can read a file f:

$$\begin{array}{l} (s = o \wedge s \in A^o(f,r)) \vee \\ \quad (s \neq o \wedge ((s \in g \wedge s \in A^g(f,r)) \vee \\ \qquad\qquad (s \notin g \wedge s \in A^e(f,r))\ )). \end{array}$$

While our model for acls is powerful enough to formally model the access control mechanism, as demonstrated above, any real system will need to be compiled into this description. For an example of a file system which uses fine grained access control lists, see AFS [15]. □

EXAMPLE C.2 (Capabilities as unforgeable bit strings)
Sparse capabilities in Amoeba

The distributed operating system Amoeba [20] uses one-way functions to compute capabilities for objects. Each object can be assumed to be managed by a server, which makes the port for accessing that object public. Clients (subjects) communicate with the object by sending it messages containing the necessary capability, i.e., a bit sequence containing the port number b_p, the object name b_o, the set of rights b_r that the capability corresponds to, and a random number b_c generated by the server managing the object. For example, to create a file `foo`, user `joe` uses his account-login capability to login, directory-write capability to create a file, and possesses the capabilities to modify this file at the end of this sequence of operations.

This situation can be modeled in a straightforward manner by using G as the server's one-way function and $c = b_p b_o b_r b_c$ as the capability. The ticket map T and set of capabilities C is stored disjointly at each of the servers, and the wallet W resides in each client's own space. □

D. Sample Proofs

Proof. (Lemma 6.1.1) We prove this by induction on the evolution path of w_A. If

$$p = w_A^i, a^1, w_A^1, a^2, w_A^2, \ldots, w_A^{n-1}, a, w_A$$

is a path, and $w_A^{n-1} \subseteq_{\text{acc}} f(w_A^{n-1})$, then we show that $w_A \subseteq_{\text{acc}} f(w_A)$ for all possible actions a. The different cases to consider (based on the last action a) are:

1. (Base) The congruence assumption $WS_A^i \cong WS_{C_{\text{row}}}^i$ implies the lemma for this case.

2. (Create) The only new access that is valid in w_A over w_A^{n-1} is $s_c \to (o, r_{acl})$. Hence we only need to check if

   $$f(w_A) = \text{Create}(s_c, o; f(w_A^{n-1})) \vdash_{C_{\text{row}}} s_c \to (o, r_e)$$

   or, equivalently, if

   $$\text{Create}(s_c, o; f(w_A^{n-1})) \vdash_{C_{\text{row}}} (o, r_e) \in C(s_c).$$

   But this is true by the definition of *Create*(Table 2). Also, since this action does not revoke any previous allowed accesses in the C_{row} model, we are done.

3. (Delete) The accesses allowed in w_A are the accesses allowed in w_A^{n-1} which do not refer to o. Hence we need to check that exactly the same accesses are denied in $f(w_A)$. This follows directly from the definition of *Delete* in Table 2, as all capabilities to o are removed from the system, and everything else is untouched.

4. (Allow) Again, the only new access valid in w_A over w_A^{n-1} is $s \to (o, r)$. Correspondingly, in the C_{row} model,

   $$f(w_A) = \text{Get}(s, o, r; f(w_A^{n-1})) \vdash_{C_{\text{row}}} s \to (o, r)$$

   since $(o, r) \in C(s)$. As no accesses are revoked, we are done.

5. (Revoke) The accesses allowed in w_A are the accesses of w_A^{n-1} except $s \rightarrow (o,r)$. Hence we need to show that the accesses of $f(w_A)$ are the accesses of $f(w_A^{n-1})$ except $s \rightarrow (o,r)$. But this follows directly from the definition of *Revoke*(Table 2). $\diamond$

Proof. (Lemma 6.2) The following functor $f^{-1}_{C_{\text{row}} \rightarrow A} : WS_{C_{\text{row}}} \rightarrow WS_A$ acts as an access containment relation between the world states of $M_{C_{\text{row}}}$ and M_{acl}.

$$
\begin{array}{lcl}
f^{-1}(WS^i_{C_{\text{row}}}) & = & WS^i_A \\
f^{-1}(\text{Create}(s_c, o; w_C)) & = & \text{Create}(s_c, o; f^{-1}(w_C)) \\
f^{-1}(\text{Grant}(s, o, r; w_C)) & = & \text{Allow}(s, o, r; f^{-1}(w_C)) \\
f^{-1}(\text{Revoke}(s, o, r; w_C)) & = & \text{Revoke}(s, o, r; f^{-1}(w_C)) \\
f^{-1}(\text{Delete}(o; w_C)) & = & \text{Delete}(o; f^{-1}(w_C))
\end{array}
$$

It can be shown by induction that

$$\forall w_C . w_C \subseteq_{\text{acc}} f^{-1}(w_C)$$

in a manner similar to the proof of Lemma 6.1.1. $\diamond$