70

A Calculus for Access Control in Distributed Systems

M. Abadi, M. Burrows, B. Lampson, G. Plotkin

February 28, 1991, revised August 28, 1991

A Calculus for Access Control in Distributed Systems

M. Abadi, M. Burrows, B. Lampson, G. Plotkin

February 28, 1991, revised August 28, 1991

M. Abadi, M. Burrows, and B. Lampson are at Digital Equipment Corporation, Systems Research Center. 130 Lytton Avenue, Palo Alto, California 94301, USA.

G. Plotkin is at the Department of Computer Science, University of Edinburgh, King's Buildings, Edinburgh EH9 3JZ, UK.

Authors' Abstract

We study some of the concepts, protocols, and algorithms for access control in distributed systems, from a logical perspective. We account for how a principal may come to believe that another principal is making a request, either on his own or on someone else's behalf. We also provide a logical language for access control lists, and theories for deciding whether requests should be granted.

Contents

1 The Problem

At least three ingredients are essential for security in computing systems:

- A trusted computing base: the hardware and systems software should be capable of preserving the secrecy and integrity of data.
- Authentication: it should be possible to determine who made a statement; for example, a user should be able to request that his files be deleted and to prove that the command is his, and not that of an intruder.
- Authorization, or access control: access control consists in deciding whether the agent that makes a statement is trusted on this statement; for example, a user may be trusted (hence obeyed) when he says that his files should be deleted.

These ingredients are fairly well understood in centralized systems.

Distributed systems pose new problems. Scale becomes an issue, as ultimately one would want to envision a global distributed system, with a global name space and global security. In addition, there are difficulties with communication, booting, loading, authentication, and authorization.

The computing base of a distributed system need not reside in a single location, under a single management. This implies, in particular, that secure communication cannot be taken for granted. Since it is hard to provide physically secure communication lines, some form of encryption is typically required. In what follows, we assume that shared-key encryption (e.g., [8]) and public-key encryption (e.g., [9, 22]) are available where needed, but we use them frugally. In addition, there are problems of secure booting and secure loading of software. For instance, the operating system that a host runs may be obtained from a repository across the network; a mechanism is needed to guarantee that this software is an authentic release, free of viruses. This mechanism will inevitably rely on authentication and access control, as it is necessary to restrict who can make a release.

The nodes of a distributed system may act on their own behalf, or on behalf of users. Users and nodes trust one another to different extents. Moreover, a user may be trusted only when he is working at certain nodes; he may have more rights at his office than when working from a remote public terminal. Therefore, both users and nodes need to be authenticated, and their identities considered in access control decisions.

These issues give rise to a change in the nature of authentication and access control. The basic questions of authentication and access control are, always, “who is speaking?” and “who is trusted?” Typically the answer is the name of a simple principal (a user or a host). In a distributed environment, these questions can receive a variety of answers; the notion of principal can be extended, to include:

- Users and machines.
- Channels, such as input devices and cryptographic channels. There is no formal reason to distinguish channels from users and machines, and it is advantageous not to. Cryptographic channels are identified by keys, and so we may write that the key K says s when s is asserted in a message decrypted with K.
- Conjunctions of principals, of the form $A \wedge B$. If A and B make the same statement s, then $A \wedge B$ says s as well. It often happens that $A \wedge B$ is trusted on s, but neither A nor B is trusted by himself—a “joint signature” is required.
- Groups. It is often inconvenient to list explicitly all of the principals that are trusted in some respect, both because the list would be long and because it may change too often. Groups provide an indirection mechanism. The use of groups implies the need for a scheme for deciding whether a principal is a member of a group. This is not always straightforward, for example when the registrar for a group is remote; group membership certificates then become necessary.
- Principals in roles, of the form A *as* R. The principal A may adopt the role R and act under the name A *as* R when he wants to diminish his powers, in particular as a protection against blunders. For example, a system manager may act in the role *normal-user* most of the time, and enable the *manager* role only on occasion. Similarly, a machine may adopt a weak role before beginning to run a piece of untrusted software, or before delegating authority to an untrusted machine.
- Principals on behalf of principals, of the form B *for* A. The principal A may delegate authority to B, and B can then act on his behalf,

using the identity B *for* A. In the most common case, a user A delegates to a machine B. It is also possible for a machine to delegate to another machine, and for delegations to be cascaded (iterated); then we encounter expressions such as C *for* $(B$ *for* $A)$.

This list raises formal questions of some practical importance. First, there is a need to determine which of the operations on principals should be viewed as primitive, and which can be defined from the others. Then one may ask what laws the operations satisfy, as for example whether B *for* $(A \wedge A')$ and $(B$ *for* $A) \wedge (B$ *for* $A')$ are in some sense equivalent; if two principals are equivalent then they should be granted the same requests. The resulting theory of principals should provide a reasonable degree of expressiveness, and it should be amenable to a mathematical justification. It is also essential that the theory of principals be simple, because users need to specify permissions on resources and programs need to make access control decisions.

Further, a variety of protocols and algorithms must be designed and analyzed. Mechanisms are required for secure booting and loading, for joining groups and for proving membership, for adopting roles, for delegations of authority and for certifying such delegations, and for deciding whether a request should be granted.

This paper is a study of some of the concepts, protocols, and algorithms for security in distributed systems, with a focus on access control. Our treatment is fairly formal, as it is based on logics. Our main goal is to isolate some useful and mathematically tractable concepts. We account for how a principal may come to believe that another principal is making a request, either on his own or on someone else's behalf. We also provide a logical language for access control lists (ACLs), and theories for deciding whether requests should be granted. The logics enable us to explain a variety of protocols which can differ from one another in subtle ways.

On occasion, the formal analysis has suggested ideas for implementations, for example that some digital signatures could be saved. Moreover, logics make it possible to describe protocols and policies at a reasonable level of abstraction; we avoid the need for *ad hoc* arguments about particular implementations. This abstraction is important in the context of heterogeneous distributed environments, where several implementations of a design may coexist.

Our study is intended as a formal basis for parts of a security architecture, and for the Digital Distributed Systems Security Architecture (DSSA) in particular [11]; this architecture is currently under implementation. Other formal explanations of security are conceivable. We hope that this work clarifies some of the issues that these alternative accounts may address.

The next section is an overview. We describe the basic logical framework in section 3. Sections 4 and 5 extend the treatment with roles and delegation. Then, in section 6, we consider delegation schemes and algorithms for making access control decisions. Section 7 is a brief discussion of additional constructs. A short glossary appears at the end.

This paper does not cover many issues in the area; some of these issues are briefly mentioned in passing. We study only authorization mechanisms based on ACLs; we do not examine particular security policies, nor matters connected with mandatory access control. Most implementation considerations are left to a companion paper [17] and to future work.

2 Overview

Composite principals play a central role in informal reasoning; for example, one often talks about "A and B" and "B on behalf of A." Therefore, we start by introducing formal notations for composite principals.

Some classical constructors for composite principals such as conjunction and disjunction come to mind first:

- $A \wedge B$: A and B as co-signers. A request from $A \wedge B$ is a request that both A and B make. (It is not implied that A and B make this request in concert.)
- $A \vee B$ is the dual notation; $A \vee B$ represents the group of which A and B are the sole members.

Conjunction is important in our designs. Disjunction is often replaced with implication, in particular in dealing with groups. As discussed further below, "A is a member of the group G" can be written $A \Rightarrow G$. The group G can be named, as all other principals; it need not be given a definition beyond that implicit in formulas such as $A \Rightarrow G$. The membership of G can be deduced from what formulas of the form $A \Rightarrow G$ have been asserted. We do not include disjunction among our basic primitives.

There are also new connectives, in particular:

- A *as* R: the principal A in role R.
- $B|A$: the principal obtained when B speaks on behalf of A, not necessarily with a proof that A has delegated authority to B; by definition, $B|A$ says s if B says that A says s; we pronounce $B|A$ as "B quoting A."
- B *for* A: the principal obtained when B speaks on behalf of A, with appropriate delegation certificates; B *for* A says s when A has delegated to B and B says that A says s, possibly after some checking that s is reasonable.

Of these, only $|$ is primitive in our approach; *for* and *as* are important, but they are coded in terms of $\wedge$ and $|$. Since *for* is stronger than $|$, we tend to use $|$ only for encoding *for* and *as*.

In order to define the rights of these composite principals, we develop an algebraic calculus. In this calculus, one can express equations such as $(B \wedge C)$ *for* $A = (B$ *for* $A) \wedge (C$ *for* $A)$, and then examine their consequences. Since $\wedge$ is the standard meet in a semilattice, we are dealing with an ordered algebra, and we can use a partial order $\Rightarrow$ among principals: $A \Rightarrow B$ stands for $A = A \wedge B$, and it means that A is at least as powerful as B; we pronounce this "A implies B" or "A speaks for B."

A modal logic extends the algebra of principals. In this logic, A *says* s represents the informal statement that the principal A says s. Here s may function as an imperative ("the file should be deleted") or not ("C's public key is K"); imperative modalities are not explicit in the formalism.

The modal logic is a basis for various algorithms and protocols. For example, it is possible to explain logically how two principals A and B establish a public key K_d for B *for* A.

The logic also underlies a theory of ACLs. We write $\supset$ for the usual logical implication connective, and A *controls* s as an abbreviation for $(A$ *says* $s) \supset s$, which expresses trust in A on the truth of s; in the logic, an ACL for a formula s is a list of assertions of the form A *controls* s. If s represents a command to a server, then the ACL entry A *controls* s records the server's trust in A on s, and hence that A will be obeyed when he says s. When s is clear from context, the ACL for s may simply be presented as the list of principals trusted on s.

If $A \Rightarrow B$ and B *controls* s then A *controls* s as well. Therefore, ACLs may not mention all trusted principals explicitly; when B is listed, access should be granted to any A such that $A \Rightarrow B$. It is not always entirely trivial to decide whether a request should be granted, and our theory addresses this issue.

So far we have discussed ACLs for logical formulas, with the view that each formula corresponds to a separate assertion or command, and that it can have a separate ACL. In practice, these ACLs could be combined, and there may be dependencies between them. For example, the ACLs that pertain to reading and writing a file might be related, and an ACL that controls modifications to another ACL might also control modifications to itself. We do not model these combinations and dependencies.

Secure channels, including those secured by cryptography, are represented as principals; we have no formal treatment of encryption functions. The logic does not consider the problems associated with timestamps and lifetimes, either. Therefore, we do not explain how principals come to believe that messages are recent, and not replays. In these respects, the logic is more abstract than the logic of authentication proposed in [4]. The logic of authentication has as its goal describing and debugging fairly low-level authentication protocols; in contrast, the logic discussed here is intended as an aid in developing and applying a general design.

3 The Basic Logic

This section describes the basic logical framework, with syntax, axioms, and semantics. The laws for *as* and *for* appear in sections 4 and 5.

This section includes discussions of alternative approaches and of relevant mathematical concepts; in particular, the last subsection presents some interesting axioms that we do not adopt. The essential material is in the main bodies of subsections 3.1 and 3.2.

3.1 A calculus of principals

We study a calculus of principals, in a minimal syntax. As discussed later in the paper, this syntax suffices for expressing roles and the needed delegation connectives.

Principals form a semilattice under the operation of conjunction, and obey the usual semilattice axioms:

- $\wedge$ is associative, commutative, and idempotent.

As usual, $A \Rightarrow B$ can be taken as an abbreviation for $A = A \wedge B$, or $=$ can be defined in terms of $\Rightarrow$.

Implication is often used to represent group membership. This notion of membership is slightly peculiar, and in particular it is transitive. We have found that transitivity is rather convenient for reasoning about hierarchical groups (groups with subgroups). For example, if A is a member of G and G a subgroup of F, we may write $A \Rightarrow G$ and $G \Rightarrow F$, and then it follows that $A \Rightarrow F$, which expresses that A is a member of F. Moreover, $\Rightarrow$ has a fairly pleasant interaction with the other connectives of our calculus.

However, our treatment of groups is incomplete. It would be desirable to provide group subtraction and intersection operations. It would also be desirable to provide naming support for usual subgroups—so that there would be a standard operation that one applies to a group name G to obtain the group of principals with a certain position or job in G. Most probably, these operations would appear only at the lowest level of logical formulas, and would not mix with other constructs. Therefore, we do not include these additional operations here; intersection and subtraction are discussed in section 7.

The principals form a semigroup under $|$:

- $|$ is associative.

The final axiom is the multiplicativity of $|$ in both of its arguments, which means:

- $|$ distributes over $\wedge$.

Multiplicativity implies monotonicity.

In short, the axioms given for principals are those of structures known as multiplicative semilattice semigroups. (Often, these same axioms are taken for structures with some sort of union or disjunction, instead of $\wedge$, and then the term additive is preferred to multiplicative.) A common example of a multiplicative semilattice semigroup is an algebra of binary relations over a set, with the operations of union and composition; algebras of binary relations are discussed further below.

The syntax for reasoning about principals is much the obvious one. We manipulate equations between principal expressions, built from atoms A_0, A_1, A_2, ...with the symbols $\wedge$ and $|$. We combine equations into boolean formulas, and then adopt the usual axioms from propositional logic, and the axioms for the algebraic structures of choice (multiplicative semilattice semigroups, when not stated otherwise).

Section 6 discusses decidability issues for the calculus of principals.

On binary relations

As mentioned, an example of a multiplicative semilattice semigroup is an algebra of binary relations over a set, with the operations of union and composition. Now we discuss binary relations in some detail, and we use them in the rest of the paper in semantic discussions; those readers not interested in formal semantics may wish to skip all references to binary relations.

A multiplicative semilattice semigroup isomorphic to an algebra of binary relations is called representable. In fact, the binary-relation example is common in a mathematical sense: every free multiplicative semilattice semigroup is representable (e.g., [3, 21]). This implies that the equational theory of multiplicative semilattice semigroups coincides with that of binary-relation algebras.

Equational theories do not suffice for reasoning about access rights, as for example group-membership assumptions are needed. Since the binary-relation model is a clear, appealing one, it is then natural to hope that every multiplicative semilattice semigroup is representable. Andréka has recently proved that this is not the case [3]. She has also studied the distributive multiplicative semilattice semigroups, which are defined by the additional distributivity axiom:

- if $B \wedge C \Rightarrow A$ then there exist B' and C' such that $B \Rightarrow B'$, $C \Rightarrow C'$, and $A = B' \wedge C'$.

Every distributive multiplicative semilattice semigroup is representable. The class of distributive multiplicative semilattice semigroups is a quasi-variety, but neither a variety nor finitely axiomatizable.

The algorithmic applications of the distributivity axiom are hard to see, because of its existential nature. Therefore, in the remainder of the paper, we examine both the axiomatic and the model-theoretic possibilities. We tend to work axiomatically, with the theory of multiplicative semilattice semigroups, but rely on binary relations as a way of constructing models. Moreover, section 6 describes an algorithm for the binary-relation model.

Other algebraic structures

It is possible, and somewhat natural, to consider operators on principals beyond $\wedge$ and $|$. Here we consider a few operators with a mathematical origin; these operators are used very seldom elsewhere in this paper.

A unit 1 turns the semigroup into a monoid; 1 can be viewed as a perfectly honest principal. The unit introduces further distinctions between the axiomatic and the binary-relation viewpoints. For example, any relation smaller than the unit is idempotent; this is not a theorem of multiplicative semilattice monoids. Thus, Andréka's remarkable representability result seems intriguingly fragile.

A disjunction operation turns the distributive semilattice into a distributive lattice. Andréka has proved that some multiplicative semigroup distributive lattices are not representable, but the free ones are representable.

If the lattice operations are generalized to the infinite case, then we have a quantale. In a quantale, the product fully distributes over one of the lattice operations; here it is $\wedge$. In particular, this says that the empty meet 0 is absorbent for $|$. Quantales provide models for (intuitionistic) linear logics [13, 26]. In these logics, the multiplicative conjunction connective ($|$, in our case) is often understood as a form of parallel composition [7, 25, 1]. In our setting, $|$ is in fact a special form of parallel composition, though a noncommutative one.

Finally, it is possible to add the remaining operator of Kleene algebras [16], namely Kleene's iteration operator $(\ )^*$. We have not yet found a need for this.

3.2 A logic of principals and their statements

Here we develop a modal logic based on the calculus of principals.

Syntax

The formulas are defined inductively, as follows:

- a countable supply of primitive propositions $p_0, p_1, p_2, \ldots$ are formulas;
- if s and s' are formulas then so are $\neg s$ and $s \wedge s'$;
- if A and B are principal expressions then $A \Rightarrow B$ is a formula;
- if A is a principal expression and s is a formula then A *says* s is a formula.

We use the usual abbreviations for boolean connectives, such as $\supset$, and we also treat equality between principals $(=)$ as an abbreviation. In addition, A *controls* s stands for $(A \textit{ says } s) \supset s$.

Axioms

The basic axioms are those for normal modal logics [14]:

- if s is an instance of a propositional-logic tautology then $\vdash s$;
- if $\vdash s$ and $\vdash (s \supset s')$ then $\vdash s'$;
- $\vdash A \textit{ says } (s \supset s') \supset (A \textit{ says } s \supset A \textit{ says } s')$;
- if $\vdash s$ then $\vdash A \textit{ says } s$, for every A.

The calculus of principals is included:

- if s is a valid formula of the calculus of principals then $\vdash s$.

Other axioms connect the calculus of principals to the modal logic:

- $\vdash (A \wedge B) \textit{ says } s \equiv (A \textit{ says } s) \wedge (B \textit{ says } s)$;

- $\vdash (B|A)$ *says* $s \equiv B$ *says* A *says* s;
- $\vdash (A \Rightarrow B) \supset ((A$ *says* $s) \supset (B$ *says* $s))$.

The last axiom is equivalent to $(A = B) \supset ((A$ *says* $s) \equiv (B$ *says* $s))$, a substitutivity property.

Other relations between principals

Two relations similar to $\Rightarrow$ deserve mention at this point.

Let us write $A \rightarrow B$ if, for every s, if A *says* s then B *says* s. Although $\rightarrow$ is weaker than $\Rightarrow$, it suffices in many of the situations where we currently use $\Rightarrow$, and hence we have contemplated adding it to the formalism. It may also be convenient to have second-order quantification over formulas and over principals. Second-order quantification enables us to define $A \rightarrow B$ as $\forall x.((A$ *says* $x) \supset (B$ *says* $x))$. We do not adopt any of these additional constructs, for the sake of minimality. However, there is no difficulty in giving a semantics for them, and the proof rules needed in examples would be simple.

Another important relation between principals is defined by the formula $(A$ *says false*$) \supset (B$ *says false*$)$, which we abbreviate $A \mapsto B$. Intuitively, $A \mapsto B$ means that there is something that A can do (say *false*) that yields an arbitrarily strong statement by B (in fact, *false*). Thus, $A \mapsto B$ means that A is at least as powerful as B in practice. Clearly, $A \Rightarrow B$ implies $A \mapsto B$. We do not have the converse, and actually the converse does not seem attractive, as for example $B \mapsto (B|A)$ is valid but we would not want $B \Rightarrow (B|A)$ (since B may wish to say s without this being taken as a quotation of A). Nevertheless, the relation $\mapsto$ serves as a point of reference in axiomatizing the algebra of principals; a careful study of $\mapsto$ may be of some interest.

3.3 Semantics

The simplest semantics is a Kripke semantics, based on accessibility relations [14]. A structure $\mathcal{M}$ is a tuple $\langle W, w_0, I, J \rangle$, where:

- W is a set (as usual, a set of possible worlds);

- w_0 is a distinguished element of W;
- I is an interpretation function which maps each proposition symbol to a subset of W (the set of worlds where the proposition symbol is true);
- J is an interpretation function which maps each principal symbol to a binary relation over W (the accessibility relation for the principal symbol).

The meaning function $\mathcal{R}$ extends J, mapping a principal expression to a relation:

$$\begin{array}{rcl}\mathcal{R}(A_i) & = & J(A_i) \\ \mathcal{R}(A \wedge B) & = & \mathcal{R}(A) \cup \mathcal{R}(B) \\ \mathcal{R}(B|A) & = & \mathcal{R}(A) \circ \mathcal{R}(B)\end{array}$$

There is no difficulty in giving a semantics to other operations on principals such as infinite conjunctions, disjunctions, 0, and 1.

This simple relational model provides some justification for the axioms for principals. If $\mathcal{R}(A) = \mathcal{R}(B)$, then it is natural to have $A = B$, and this holds in the semantics. For example, $\mathcal{R}(C|(B|A)) = \mathcal{R}((C|B)|A)$ provides a justification for the associativity of $|$.

The meaning function $\mathcal{E}$ maps each formula to its extension, that is, to the set of worlds where it is true:

$$\begin{array}{rcl}\mathcal{E}(p_i) & = & I(p_i) \\ \mathcal{E}(\neg s) & = & W - \mathcal{E}(s) \\ \mathcal{E}(s \wedge s') & = & \mathcal{E}(s) \cap \mathcal{E}(s') \\ \mathcal{E}(A \text{ } \mathit{says} \text{ } s) & = & \{w \mid \mathcal{R}(A)(w) \subseteq \mathcal{E}(s)\} \\ \mathcal{E}(A \Rightarrow B) & = & W \text{ if } \mathcal{R}(B) \subseteq \mathcal{R}(A) \text{ and } \emptyset \text{ otherwise}\end{array}$$

where $\mathcal{R}(C)(w) = \{w' \mid w\mathcal{R}(C)w'\}$.

A formula s holds in $\mathcal{M}$ at a world w if $w \in \mathcal{E}(s)$, and it holds in $\mathcal{M}$ if it holds at w_0. In the latter case, we write $\mathcal{M} \models s$, and say that $\mathcal{M}$ satisfies s. Moreover, s is valid if it holds in all models; we write this $\models s$. The axioms are sound, in the sense that if $\vdash s$ then $\models s$. Although useful for our application, the axioms are not complete. For example, the formula

$$(C \text{ } \mathit{says} \text{ } (A \Rightarrow B)) \equiv ((A \Rightarrow B) \vee (C \text{ } \mathit{says} \text{ } \mathit{false}))$$

is valid but not provable. A more interesting source of incompleteness is that the algebras of principals that underly the semantics are obviously representable, while some multiplicative semilattice semigroups are not. (We do obtain a completeness result for a sublanguage, in section 6.)

More abstract models

A more abstract model may be desirable in order to avoid the requirement of representability. A first step would be to modify the notion of structure. A structure would become a tuple $\langle W, w_0, I, J, \mathcal{P}, \mathcal{F}\rangle$, where $\mathcal{P}$ is a multiplicative semilattice semigroup (the principals), and $\mathcal{F}$ is a homomorphism from $\mathcal{P}$ to W^2. The functions J and $\mathcal{R}$ map principal expressions not to binary relations but to principals—the corresponding binary relations being still available via $\mathcal{F}$. The corresponding semantics is:

$$\begin{array}{rcl}
\mathcal{R}(A_i) & = & J(A_i) \\
\mathcal{R}(A \wedge B) & = & \mathcal{R}(A) \wedge \mathcal{R}(B) \\
\mathcal{R}(B|A) & = & \mathcal{R}(A) \circ \mathcal{R}(B) \\
 & & \ldots \\
\mathcal{E}(A \; says \; s) & = & \{w \mid \mathcal{F}(\mathcal{R}(A))(w) \subseteq \mathcal{E}(s)\} \\
\mathcal{E}(A \Rightarrow B) & = & W \text{ if } \mathcal{R}(A) \Rightarrow \mathcal{R}(B) \text{ and } \emptyset \text{ otherwise}
\end{array}$$

Now it is consistent to have two different principals with the same associated binary relation.

This semantics is imbalanced, as it gives an abstract interpretation of principals but not of propositions. A more balanced, abstract semantics is possible, in terms of structures similar to dynamic algebras (see Pratt's [21] for a recent discussion of dynamic algebras). The operator *says* is directly analogous to the usual partial correctness operator [ ] of dynamic logic. It connects the two sorts of the algebras, that of principals and that of propositions.

3.4 On idempotence

Idempotence postulates are attractive. For example, when A is a user, there seems to be little advantage in distinguishing $A|A$ and A; in fact, this distinction seems quite artificial. Therefore, we may postulate that $A|A = A$, for all A, and hence that A *says* A *says* s and A *says* s are equivalent. The idempotence of $|$ implies the idempotence of *for* (see section 5 for details). Here we discuss the idempotence of $|$ and *for*; most of the remarks below are written in terms of $|$, but exactly the same arguments apply to *for*.

The idempotence axiom is rather convenient for handling chains of principals. For instance, suppose that G represents a collection of nodes, that B and C represent members of G, and that an ACL includes $G|A$. Thanks to idempotence, $C|B|A$ obtains access. This means that multiple "hops" within a collection of nodes does not reduce rights and should not reduce security. In particular, there is no need to postulate that $G|G \Rightarrow G$, or to make sure that $G|G|A$ appears in the ACL explicitly.

In addition, idempotence yields $(A \wedge B) \Rightarrow (B|A)$, since $(A \wedge B) = (A \wedge B)|(A \wedge B)$ and $|$ is monotonic, and similarly $(A \wedge B) \Rightarrow (B$ *for* $A)$. These implications are rather pleasing, but not necessarily intuitive. They complicate the problem of making access control decisions. When $(A \wedge B)$ makes a request, one must check whether either $(B|A)$ or $(A|B)$ is trusted, and grant access if either is trusted.

Finally, the semantics of subsection 3.3 does not validate idempotence. We have been unable to find a sensible condition on binary relations that would force idempotence and would be preserved by both union and composition. A related symptom is that idempotence seems less compelling for complex principals than for simple ones.

For all these reasons, we prefer to do without idempotence; in compensation, we often rely on assumptions of the form $G|G \Rightarrow G$, or write less elegant but more explicit ACLs.

We give a similar treatment to other possible axioms related to idempotence. Although in many cases it is reasonable to assume formulas of the forms A *controls* $(B \Rightarrow A)$ and A *controls* $(t \supset A$ *says* $s)$, this does not seem well founded in general. In particular, when A is a group, if A *controls* $(B \Rightarrow A)$ for all B then any member of A can add members to A. Therefore, we choose to adopt neither A *controls* $(B \Rightarrow A)$ nor A *controls* $(t \supset A$ *says* $s)$ as general axioms.

4 Roles

The formal system we have presented so far includes no exotic connectives for roles or delegation. We discuss roles in this section and delegation in the next one. We start with an informal argument on the nature of roles and then describe their logic.

4.1 What roles are for

There are many situations in which a principal may wish to reduce his powers. We now describe a few, as motivation for our treatment of roles. They are all examples of the principle of "least privilege," according to which a principal should have only the privileges it needs to accomplish its task.

An administrator may want to have the powers of a normal user most of the time, and exercise his extraordinary powers only when needed. For example, users of the UNIX[1] operating system with system-manager privileges typically run with their own, normal identity when that suffices, in order to avoid costly mistakes.

When a principal wishes to run a piece of untrusted software, he should be able to invoke it with reduced powers. It is important that the untrusted code should not be able to increase its powers beyond those granted initially. This is done in capability systems by restricting capabilities before passing them across address spaces, and passing as few capabilities as possible. In timesharing systems it is common for untrusted software to be tested with unprivileged user accounts that are used for no other purpose.

Analogously, when a principal wishes to delegate his powers to a machine less trustworthy than his own, he should be able to limit the rights passed.

These situations can be handled by the use of roles. A principal A may adopt a role R and act with the identity A *as* R when he wants to diminish his powers. Thus, A can decide to become A *as* R, and then later perhaps A *as* R *as* R'. Role adoption is not reversible, in the sense that A *as* R cannot, on his own, recover the full powers of A. This prevents untrusted software that has been granted only limited rights from obtaining the full rights of the user that invoked it. But this does not mean that a user who

[1]UNIX is a registered trademark of UNIX System Laboratories, Inc.

has once relied on roles can never again exercise his full powers: a process acting on behalf of the user may hold on to the credentials for the user's original identity, which it could use when instructed.

As a practical matter, it may be best for users to have only restricted powers when they first log in. In this way, users could be forced to authenticate themselves a second time when they want extraordinary powers. Our general treatment of roles does not require or preclude such implementation decisions, which are however important in building a usable system.

4.2 Roles, groups, and resources

A principal A may reduce his powers in different ways by adopting different roles, say R or R'. Some ACLs may grant permissions to A *as* R but not to A *as* R', and vice versa. There is however no intrinsic difference between R and R'; that is, if the uses of their names were swapped consistently in all ACLs, then the effects of adopting the two roles would be swapped. Moreover, R and R' may be used differently at two independent sites. Thus, the meaning of roles could be a matter of local policy at each site; this policy is reflected in the writing of ACLs.

In practice, we expect many roles to be related to groups. If G is a group, there may be a role G_{role} associated with it, so that a member of a group G can act in the role of member of G. A member A of several groups F, G, H, ...can select the privileges associated with one, say G, by adopting the corresponding G_{role}. Without this role A matches any ACL with an entry F, but A *as* G_{role} need not. However, A *as* G_{role} matches any ACL with an entry G *as* G_{role}, and if we accept the rule that $G = G$ *as* G_{role} then A *as* G_{role} also matches any ACL with an entry G.

It is tempting to establish a formal identification between the group G and a corresponding role *G_role*. In what follows, for simplicity, we do not identify roles and groups. In fact, this identification would interact somewhat strangely with our encoding of roles, given below. We do allow roles related to groups (such as *G_role*), but this relation is not formal; a formal operator for relating groups and roles may be a useful extension of our calculus.

It is reasonable to expect that not all roles make sense for all principals, and that a principal can act only in certain roles. For example, a principal may be allowed to act as *G_role* only if he is a member of G. The simplest implementation of this idea relies on the judicious writing of ACLs. Roles can be adopted freely, and any A can speak in the role G_role, with the identity A *as* G_role. However, it may be that no ACL in the system grants access to A *as* G_role, and for example the ACL G *as* G_role does not if A is not a member of G. It is this implementation that we choose.

Not all roles that arise naturally are related to groups. For example, there may be a role such that adopting it makes it possible to access only a certain directory in the file system; the role corresponds naturally to a set of resources (files) rather than to any group of principals.

4.3 The encoding

Given the view that roles can be freely adopted, it is quite satisfactory to define A *as* R to equal $A|R$.

In many cases, this encoding makes some intuitive sense. For example, let A be a machine and R a (possibly untrusted) piece of software that A is running. The requests that A makes while running this software should not emanate from A with full powers; rather, they should emanate from A in some restricted role. The role can be named after the piece of software, or after its class (e.g., *untrusted-software*). In any case, when A makes a request s in this role, A says that R says s. This is precisely $A|R$ *says* s. Similarly, if A is a user and R is one of his positions, we can think of R as a program that A is following, and apply the analysis above.

Furthermore, this presentation of roles offers formal advantages, thanks to the monotonicity, the multiplicativity, and the associativity of $|$.

- Since $|$ is monotonic in both arguments, if R is a more trusted role than R' and A a more trusted principal than A', then A *as* R is more trusted than A' *as* R', that is, if $R \Rightarrow R'$ and $A \Rightarrow A'$ then $(A \textit{ as } R) \Rightarrow (A' \textit{ as } R')$. Thus, it is possible to exploit a hierarchy of roles.

- The multiplicativity of $|$ yields that $A \wedge B$ in a role R is identical to the conjunction A and B both in role R:

  $$(A \wedge B) \textit{ as } R = (A \textit{ as } R) \wedge (B \textit{ as } R)$$

  Similarly, we can exploit the conjunction operation for roles:

  $$A \textit{ as } (R \wedge R') = (A \textit{ as } R) \wedge (A \textit{ as } R')$$

Both of these properties seem reasonable for roles as we conceive them informally, and they are quite handy.

- Associativity yields a satisfactory interaction between roles and delegation. In particular, the user A in role R on a machine B is identical to the user A on a machine B, with the compound in role R:

  $$B|(A\ as\ R) = (B|A)\ as\ R$$

  and then the encoding of *for*, given in the next section, yields:

  $$B\ for\ (A\ as\ R) = (B\ for\ A)\ as\ R$$

Some special properties for roles can be expected:

- We sometimes rely on the postulates that all roles are idempotent ($R|R = R$) and commute with one another ($R'|R = R|R'$). This yields:

  $$A\ as\ R\ as\ R = A\ as\ R$$

  and

  $$A\ as\ R\ as\ R' = A\ as\ R'\ as\ R$$

  (We make it explicit when these properties are assumed.)

- We assume $A \Rightarrow (A\ as\ R)$ for all A. With a unit, this could just be written $1 \Rightarrow R$. In the binary-relation model, $1 \Rightarrow R$ means that roles are subsets of the identity relation; note that the previous idempotence and commutativity properties follow from this.

5 Delegation

Delegation is a fairly ill-defined term. Here we explain our approach to delegation; some example schemes for delegation illustrate the use of the logic. Then we give an encoding of delegation.

5.1 The forms of delegation

Delegation is a basic primitive for secure distributed computing. It is the ability of a principal A to give to another principal B the authority to act on A's behalf. When B requests a service from a third principal, B might present credentials which are supposed to demonstrate that B is making the request and that A has delegated to B.

Naturally, access control decisions need to take delegations into account. A range of designs is possible.

In the simplest case, A delegates all of his rights to B. Upon B's request, a service will be granted if it would have been granted to A, had A requested it directly.

An obvious improvement is for A to delegate only some of his rights, such as the right to read certain files from a file server. Capability systems (e.g., [10, 23, 18]) embody this approach in the case where delegation certificates can be transferred; the containment of capabilities is a serious concern. If delegation certificates are not transferable, cascaded delegations are hampered, as A may have to prepare adequate certificates for any expected collaborators to B in advance. In all cases, the provider of the service never checks that B is a reasonable delegate for A.

These considerations suggest more sophisticated designs, where access control decisions may depend on the identities of both A and B. For example, the file server may not let a known malicious workstation read a user's files, even if the user was unlucky enough to log onto the workstation. The file server may also let a trusted node read files over which the user has execute-only rights, in the belief that these files will not be shown to the user, but just executed; here the node acts as a protected subsystem, and offers guarantees that its clients cannot provide by themselves. These examples illustrate a desirable flexibility. It exists in some form in a few designs and systems [24, 15], and it is a prominent component of DSSA [12]. We believe that it should and will become widespread.

Different mechanisms embody the concept of delegation in different settings. A user might rely on a smart-card for delegating to a workstation, while delegation across the address spaces of a machine might benefit from operating system support. (The use of smart-cards is discussed in [2], with a simplistic view of delegation.) Similarly, credentials can be signed with either shared-key cryptography or with public-key cryptography, or they might not be signed at all when a secure physical channel is available. Recognizing and treating all these variants as mere instances of delegation makes possible a uniform view of access control throughout an entire distributed system.

Delegation relies on authentication, which is often achieved through authentication protocols (see for example [20, 5, 19, 15, 4]). In fact, some of the messages for delegation can be combined with those for authentication. However, in our study, it is often convenient to view delegation as independent.

Delegation without certificates

The framework outlined so far enables us to describe and compare various schemes for delegation. In what follows, A delegates to B, who makes requests to C. For instance, A may be a user with a sufficiently powerful smart-card, B a workstation, and C a file server. For simplicity, the authority delegated from A to B is not limited to any particular class of requests, and A does not adopt a special role before the delegation.

We assume that synchronized clocks are available, and that appropriate timestamps, lifetimes, sequence numbers, and message type fields are included and checked for all messages. We also assume that all principals can perform digital signatures, for example with the RSA algorithm [22]. We denote by K_A and K_B the public keys for A and B, and by K_A^{-1} and K_B^{-1} the matching secret keys. The formula K *says* X represents the certificate X encrypted under K^{-1}, which is commonly written $\{X\}_{K^{-1}}$.

A certification authority S provides certificates for the principals' public keys as needed. The necessary certificates are K_S *says* $K_A \Rightarrow A$ and K_S *says* $K_B \Rightarrow B$, where K_S is S's public key.

We consider three instances of delegation. In each case, we are led to ask whether composite principals, such as $B|A$, appear on C's ACL. The simplest instance of delegation is delegation without certificates:

1. When B wishes to make a request r on A's behalf, B sends the signed request along with A's name, for example in the format K_B *says* A *says* r.

2. When C receives the request r, he has evidence that B has said that A requests r, but not that A has delegated to B; then C consults

the ACL for request r, and determines whether the request should be granted under these circumstances.

The logic serves in describing the reasoning of the service provider, C, who makes the access control decision. Some assumptions are needed. First, C must believe that K_S is S's public key:

$$K_S \Rightarrow S$$

and C obtains a certificate encrypted under the inverse of K_S:

$$K_S \textit{ says } (K_B \Rightarrow B)$$

These two formulas yield:

$$S \textit{ says } (K_B \Rightarrow B)$$

We should assume that C trusts S for such statements:

$$S \textit{ controls } (K_B \Rightarrow B)$$

and we immediately obtain that C has B's key:

$$K_B \Rightarrow B$$

In addition, C sees a message under the inverse of K_B, requesting r on behalf of A; in other words,

$$K_B \textit{ says } A \textit{ says } r$$

Immediately, C obtains:

$$B \textit{ says } A \textit{ says } r$$

that is,

$$(B|A) \textit{ says } r$$

Now C consults an ACL for r. If $B|A$ appears in this ACL, that is, if $B|A$ is trusted on r, then C believes r—access is granted.

Delegation with certificates

The protocol just described can hardly be considered satisfactory in general. It is preferable for A to issue a delegation certificate that proves the delegation to B:

1. After mutual authentication, A issues a certificate to B, under A's key. This certificate states that A has delegated some authority to B.

2. When B wishes to request r on A's behalf, no further interaction with A is needed: B can present A's certificate to C, along with K_B *says* A *says* r.

3. At this point C has evidence that B has requested r on behalf of A; now C consults the ACL for r, and determines whether the request should be granted.

For the time being, let us use the notation B *serves* A to mean that B is a delegate for A. (But *serves* need not be primitive; see subsection 5.2.) Then the delegation certificate from A for B can be expressed:

$$K_A \textit{ says } (B \textit{ serves } A)$$

and the usual checking of public-key certificates from S yields:

$$A \textit{ says } (B \textit{ serves } A)$$

If C trusts A on this statement, C gets:

$$((B|A) \textit{ says } r) \wedge (B \textit{ serves } A)$$

In our theories, this implies:

$$(B \textit{ for } A) \textit{ says } r$$

Again C can consult the ACL for r, and r will be granted if the list includes B *for* A. In particular, r will be granted if the list mentions $B|A$, a weaker principal, but may be granted even otherwise.

This scheme is illustrated in more detail by an example in section 6.

Delegation without delegate authentication

Another variant consists in omitting the authentication between B and C, leaving the responsibility of authenticating B solely to A:

1. After mutual authentication, A issues a certificate to B, under A's key. The certificate includes a key K_d and B's name. The inverse key K_d^{-1} remains a secret known at most to A and B.

2. When B wishes to request r on A's behalf, B can present A's certificate to C, along with the request under the delegation key K_d.

3. At this point C has evidence that B has requested r on behalf of A; now C consults the ACL for r, and determines whether the request should be granted.

The most obvious novelty in this scheme is the introduction of a delegation key K_d. One of the reasons why this is significant is that the delegate can simply forget the secret key K_d^{-1} in order to give up the power of acting on behalf of the delegator; section 6 discusses this use of delegation keys. For efficiency, however, requests would be made under neither K_B nor K_d, but rather under a shared session key, for example a DES key [8]. The use of a session key is a significant optimization, because of the efficiency of DES encryption, and does not compromise security.

The final items in these descriptions are deceptively identical. In the previous schemes, C obtains direct evidence that B is making the request. In this scheme, on the other hand, this evidence is provided only by A's certificate, which includes B's name. Accordingly, we may want C to make different access control decisions in the two cases. This distinction is necessary, for example in the case in which B is a protected subsystem. It is a distinction that a theory of delegation should allow, and hopefully clarify.

Let us denote by A *on* B the compound principal that is obtained by this delegation scheme. It is sensible to assume that $A \Rightarrow (A \textit{ on } B)$, for A to be believed when he claims that K_d speaks for A *on* B. This property suggests the definition $(A \textit{ on } B) = (A \vee (B \textit{ for } A))$; the definition yields suitable properties for *on*.

The use of *on* is problematic in a world where some ACLs may allow B *for* A but not A *on* B. The interaction between *on* and *for* is unfortunate. In particular, a request from A *on* B may fail where one from B *for* A would succeed; this is unpleasant, since *for* could have appeared in the request instead of *on*. It poses the problem of how to choose when to use *on* and when to use *for*. It seems best to use *for* throughout. This choice can be made affordable, and it also offers the advantage of simplicity—the design requires fewer credential formats and fewer laws. Therefore, we do not discuss *on* further.

5.2 The encoding

The delegation of all powers can be defined convincingly: A delegates to B when A says that B speaks for A (that is, when A *says* $(B \Rightarrow A)$ holds). The concept of delegation that interests us here is subtler. When A delegates to B, the powers of B on behalf of A depend on the contents of the ACLs in the distributed system; A may be more reckless with delegations if these powers are small. In turn, the contents of the ACLs depend on how their writers understand the meaning of delegation. There is a circularity here. The meaning of delegation results from the combination of the contents of the ACLs in the system and of the behavior of delegators and delegates. Hence, we conclude that the meaning of delegation is a matter of policy or convention.

These remarks suggest an approach where the notion of delegation is primitive; we now sketch this approach, assuming for a moment that disjunction is available. The operator *serves* is taken as primitive, and axiomatized. Some straightforward axioms come to mind, and in particular that B *serves* A is antimonotonic in B, and that if B_i *serves* A for all i, then $(\bigvee_i B_i)$ *serves* A. Let $E(A)$ be the weakest principal to which A has delegated, and let $E_B(A)$ be the weakest principal stronger than B to which A has delegated. Then B *for* A can be defined to be $E_B(A)|A$, and this equals $(B \wedge E(A))|A$. Axioms for delegation yield properties of *for*; for example, the axioms given yield that *for* is multiplicative in its second argument. We have not found a satisfactory way to obtain associativity.

We prefer not having delegation as a primitive; the following thought experiment inspires an alternative approach. Imagine that there is a distinguished principal D in the distributed system who operates as a delegation server. The function of D is to certify delegations. When A wants to delegate to B, it suffices for A to tell D that if B says that A says s, then D should back

B. Thus, if $B|A$ *says* s then $D|A$ *says* s. Intuitively, D says that A says s when a delegate of A's makes this assertion. Thus, we can take $(B \wedge D)|A$ as a formal encoding for B *for* A. Notice the striking similarity between this encoding of *for* and the previous formulation.

It is not actually necessary for D to be implemented, just as typical roles do not correspond to real users and machines. When A wishes to delegate to B, A says that for all s, if B says that A says s then D says that A says s; formally, it suffices to have A *says* $(B|A \Rightarrow D|A)$. The statement A *controls* $(B|A \Rightarrow D|A)$ then represents that A can delegate to B. These two assertions together immediately yield $(B|A \Rightarrow D|A)$, and when $(B|A)$ *says* s, we obtain $(D|A)$ *says* s, and then $((B \wedge D)|A)$ *says* s. In this formal derivation, it is irrelevant whether D is a real server or not. (It is perhaps even better if D is not a real server, for then it cannot be attacked.)

Thus, we can take B *for* A to mean $(B \wedge D)|A$, where D is a distinguished fictional principal. Similarly, $(B|A \Rightarrow D|A)$ can represent B *serves* A; hence, if B *serves* A then $(B|A \Rightarrow B \textit{ for } A)$. This expresses the intuition that *for* is $|$ "plus a delegation." The logical framework also allows the possibility of weaker delegation statements, where A delegates only some of this rights; we prefer the use of roles for limiting the power of delegations.

Our encoding has sensible formal consequences:

- *for* is monotonic in both arguments.
- *for* is multiplicative in both arguments, in fact. This follows from the multiplicativity of $|$ and the definition of *for*:

$$\begin{aligned}
&(B \wedge B') \textit{ for } (A \wedge A') = (B \wedge B' \wedge D)|(A \wedge A') \\
&= ((B \wedge D) \wedge (B' \wedge D))|(A \wedge A') \\
&= ((B \wedge D)|A) \wedge ((B \wedge D)|A') \\
&\quad \wedge ((B' \wedge D)|A) \wedge ((B' \wedge D)|A') \\
&= (B \textit{ for } A) \wedge (B \textit{ for } A') \wedge (B' \textit{ for } A) \wedge (B' \textit{ for } A')
\end{aligned}$$

- B *for* A is always defined, even if A has not delegated to B. In fact, we have:

$$(B|A) \wedge (C \textit{ for } A) \Rightarrow ((B \wedge C) \textit{ for } A)$$

  and hence also

$$(B|A) \wedge (C \textit{ for } A) \Rightarrow (B \textit{ for } A)$$

This somewhat surprising theorem is the consequence of two desirable, basic properties: the monotonicity of *for*, and the antimonotonicity of delegation (which means that if A delegates to C, then it also delegates to the stronger principal $B \wedge C$).

Additional properties of D yield further consequences:

- If $D|D \Rightarrow D$ then *for* possesses a weak associativity property:

  $$C \textit{ for } (B \textit{ for } A) \Rightarrow (C \textit{ for } B) \textit{ for } A$$

  which follows from the associativity and the multiplicativity of $|$:

  $$\begin{aligned} &C \textit{ for } (B \textit{ for } A) = (C \wedge D)|((B \wedge D)|A) \\ &= (C|B|A) \wedge (C|D|A) \wedge (D|B|A) \wedge (D|D|A) \\ &\Rightarrow (C|B|A) \wedge (C|D|A) \wedge (D|B|A) \wedge (D|A) \\ &\Rightarrow (C|B|A) \wedge (D|B|A) \wedge (D|A) \\ &= ((C \textit{ for } B)|A) \wedge (D|A) \\ &= (C \textit{ for } B) \textit{ for } A \end{aligned}$$

  The other direction of the implication is not valid. The essential reason for this is that C *for* $(B$ *for* $A)$ implies $C|D|A$, while $(C$ *for* $B)$ *for* A does not. Intuitively, this is because C's part in $(C$ *for* $B)$ *for* A need not involve checking evidence that B is a delegate for A, or even that A exists.

- If $A \Rightarrow D|A$ then $(A \wedge (B|A)) \Rightarrow (B \textit{ for } A)$, because $|$ is multiplicative:

  $$\begin{aligned} A \wedge (B|A) &\Rightarrow (D|A) \wedge (B|A) \\ &= ((B \wedge D)|A) \\ &= B \textit{ for } A \end{aligned}$$

  This property means that when A makes a statement himself, there is no need to find a corresponding statement by the delegation server.

- If $A \Rightarrow D|A$ and $A = A|A$ then $A = A \textit{ for } A$, that is, the idempotence of $|$ implies the idempotence of *for*:

  $$\begin{aligned} A \textit{ for } A &= ((A \wedge D)|A) \\ &= (A|A) \wedge (D|A) \\ &= A \wedge (D|A) \\ &= A \end{aligned}$$

The additional properties for D are reasonable, and we adopt them. These properties are reminiscent of those for roles. In the binary-relation model, for example, these properties of D and those for roles all amount to saying the same thing, that the associated binary relations are subsets of the identity.

6 Protocols and Algorithms

In this section we consider some mechanisms for delegation and for access control decisions. The last subsection presents an example.

6.1 Delegation

While delegation has a single semantics, substantially different implementations are recommended for users and for nodes.

6.1.1 Delegation from users to nodes

The schemes for delegation discussed in section 5 do not seem adequate for users. Delegation from users to nodes poses special problems. One difficulty is that it is inconvenient for users to refresh delegation certificates. Refreshing a delegation certificate may require reintroducing a smart-card, for example. Therefore, it seems desirable that delegation certificates be relatively long-lived (valid for many hours).

Long-lived delegation certificates pose a security threat, because a delegation certificate may be valid long after the user has stopped using the node. An attacker may subvert the node over a relatively long time period, and abuse the user's delegation. To prevent this from happening, it is best to introduce an auxiliary delegation key that the node can forget deliberately when the user leaves. The delegation certificate remains valid, but becomes useless.

In more detail, the protocol goes as follows. First the user A delegates to the node B and to a public key K_d provided by the node:

$$A \textit{ says } ((K_d \wedge B) \textit{ serves } A)$$

The node has the inverse of the public key, and it can set up a channel Ch for $(K_d \wedge B)$ *for* A:

$$K_d \; says \; A \; says \; (Ch \Rightarrow (K_d \wedge B) \; for \; A)$$

and

$$B \; says \; A \; says \; (Ch \Rightarrow (K_d \wedge B) \; for \; A)$$

Hence

$$((K_d \wedge B) \; for \; A) \; says \; (Ch \Rightarrow (K_d \wedge B) \; for \; A)$$

follows logically. When this statement is believed, it yields:

$$Ch \Rightarrow (K_d \wedge B) \; for \; A$$

and then monotonicity leads to the desired result:

$$Ch \Rightarrow B \; for \; A$$

Hereafter B can make requests for A through the channel Ch. In practice, the channel may be obtained by multiplexing a longer-term secure channel from B; this longer-term channel may well be a DES channel.

The delegation from K_d to the channel has a relatively short lifetime, and needs to be renewed frequently. When the node forgets K_d deliberately, it loses the ability to refresh the certificate for the channel Ch.

6.1.2 Delegation from nodes to nodes

For nodes, the schemes of section 5 are convenient enough. The one corresponding to the *for* operator is simple, reasonably secure, and it can be made efficient enough for implementation.

The same ideas work for cascaded delegations. Suppose that user A has delegated to node B, and B can operate with the identity B *for* A. If a further delegation is needed, to a node C, the precise delegation statement is:

$$(B|A) \; says \; (C \; serves \; (B \; for \; A))$$

Since A has delegated to B, it follows logically that:

$$(B \; for \; A) \; says \; (C \; serves \; (B \; for \; A))$$

This statement is believed, and it yields:

$$C\ \textit{serves}\ (B\ \textit{for}\ A)$$

Now C can make a request r on behalf of B *for* A:

$$C\ \textit{says}\ (B\ \textit{for}\ A)\ \textit{says}\ r$$

and then the delegation from A yields:

$$(C\ \textit{for}\ (B\ \textit{for}\ A))\ \textit{says}\ r$$

6.2 Access control decisions

Unfortunately, the set of valid formulas of the calculus of principals is not recursive for any useful definition of validity, but it is recursively enumerable. Undecidability can be proved by a reduction from the word problem for Thue systems, for example. On the other hand, the formulas that arise in access control are not arbitrary; the next two parts discuss decidable access control problems.

6.2.1 A general access control problem

The problem of making access control decisions is computationally complex. It is important therefore to understand the precise form of its instances. The parts of an instance are:

- An expression P in the calculus of principals represents the principal that is making the request. In particular, all appropriate delegations are taken into account in constructing this expression. The various relevant certificates are presented for checking.

- A statement s represents what is being requested or asserted; the statement is presented explicitly by the requester, and the service provider does not need to derive it logically from other statements. The precise nature of s is ignored—it is treated as an uninterpreted proposition symbol.

- Assumptions state implications among principals; these typically represent assumptions about group memberships. They have the form

$P_i \Rightarrow G_i$, where P_i is an arbitrary expression in the calculus of principals and G_i an atom. Note that this syntax is liberal enough to write $G|G \Rightarrow G$ for every appropriate G of interest, obtaining some of the benefit of the idempotence axiom.

- Certain atomic symbols R_0, ..., R_i, ...are known to denote roles. This may be obvious from their names.
- An ACL is a list of expressions E_0, ..., E_i, ... in the calculus of principals; these represent the principals that are trusted on s.

The basic problem of access control is deciding whether $\bigwedge_i(P_i \Rightarrow G_i)$ and $\bigwedge_i(E_i$ *controls* $s)$ imply P *controls* s, given the special properties of roles and of the delegation server D. We simplify the problem, and ask instead whether $\bigwedge_i(P_i \Rightarrow G_i)$ implies $P \Rightarrow E_j$ for some j.

In this simplification, ACL entries are considered one by one, and their possible interactions are ignored. ACL entries written by two different users cannot lead to results that neither of them could foresee. The simplification of the problem is both convenient and sound, but in some cases it is incomplete. For example, when the model is a monoid of binary relations, if R *controls* s and R' *controls* s and both R and R' are roles, then $(R'|R)$ *controls* s; on the other hand, the requester $(R'|R)$ is denied access when the entries R and R' are considered separately. We do not know under what general conditions one has completeness; however, we prove a completeness result in the next subsection.

Fixing a j and introducing an auxiliary atom A, the access control question becomes whether $\bigwedge_i(P_i \Rightarrow G_i)$ and $E_j \Rightarrow A$ imply $P \Rightarrow A$. After some renamings, this is: given expressions P, Q_0, ..., Q_i, ..., and atoms A, B_0, ..., B_i, ..., does $\bigwedge_i(Q_i \Rightarrow B_i)$ imply $P \Rightarrow A$?

This problem has a few interesting, tractable cases. If we ignore roles and D for the time being and allow only the operator $\wedge$, we have the problem of deriving a Horn clause from other Horn clauses. If instead the only operator is $|$, we have the problem of deciding membership of a word in a context-free language. The word corresponds to P, and the grammar for the context-free language comes from the implications $(Q_i \Rightarrow B_i)$; the atom A serves as initial symbol.

It is the combination of $\wedge$ and $|$ that makes the problem hard. More precisely, let us ignore roles and D, allow both $|$ and $\wedge$, and assume the theory of multiplicative semilattice semigroups. Context-free grammars can still be encoded, but in addition the "universal branching" due to $\wedge$ alternates with the "existential branching" that comes from choosing between rules of the encoded grammars. Using these ideas, we have sketched a proof that the problem of making access control decisions is equivalent to the acceptance problem for alternating pushdown automata [6].

It follows that the fairly general access control problem that we posed requires exponential time. We believe that the inclusion of various reasonable properties for roles and for D does not worsen this complexity. (We have considered some of the cases.) Although all of the expressions involved seem likely to be small, we fear that our algorithms would not run fast enough in practice. Further restrictions are wanted.

6.2.2 A more tractable problem

A second version of the access control problem is based on some further simplifications:

- The *for* operator takes a more prominent role. Since it is expected to be common, one would want an algorithm that does not treat it by expanding it in terms of $\wedge$ and $|$, with an exponential blow-up.
- In effect, *for* is treated as an associative operator. More precisely, we assume that all ACLs have the weakest parenthesization possible. This enables us to ignore the parenthesization of the requester.
- The $|$ operator occurs only in the encoding of *for* and *as*, and D occurs only in the encoding of *for*.
- Roles are differentiated from other principals (called proper principals), and occur in special positions. The set of atomic symbols for proper principals is $\mathcal{P}$; the set of atomic symbols for roles is $\mathcal{Q}$.
- Assumptions (group memberships) are restricted to be of the form $atom \Rightarrow atom$, where both atoms are either in $\mathcal{P}$ or in $\mathcal{Q}$. In particular, this forbids the assumptions of the form $G|G \Rightarrow G$, which would give us some benefits of idempotence where appropriate.

- It is therefore necessary to include a construct for writing, in an ACL, that any positive number of iterations of a principal are allowed. We introduce a metalogical construct $(\ )^+$ for this purpose; for example F *for* G^+ is a shorthand for the list F *for* G, $(F$ *for* $G)$ *for* G, $\ldots$.

An ACL is a list of ACL entries. In turn, an ACL entry is a conjunction of ACL entries free of $\wedge$, called *for*-lists. A *for*-list is a list connected by *for*'s, that is, it has the form P_0 *for* $\ldots$ *for* P_m, where each P_i is a principal in roles. A principal in roles is of the form Q *as* R_1 *as* $\ldots R_n$, where Q is a proper principal and each R_j is a role.

This definition is summarized by a grammar:

$$\begin{array}{rcl}
\textit{ACL} & = & \textit{list of Entry} \\
\textit{Requester} & = & \textit{Entry} \\
\textit{Entry} & = & \textit{conjunction of for-list} \\
\textit{for-list} & = & \textit{Principal-in-Roles} \mid \textit{for-list for Principal-in-Roles} \\
\textit{Principal-in-Roles} & = & \textit{Proper-Principal} \mid \textit{Principal-in-Roles as Role} \\
\textit{Membership} & = & \textit{Proper-Principal} \Rightarrow \textit{Proper-Principal} \mid \textit{Role} \Rightarrow \textit{Role}
\end{array}$$

The syntactic conditions can be loosened, as this normal form for ACL entries can be obtained by two syntactic transformations justified by the laws of the logic. Conjunctions can be pushed outwards, since both *as* and *for* distribute over $\wedge$; this normalization is the only source of exponential behavior, and it can be expected to be unimportant in the common examples where conjunctions are rare. Roles can be pushed inwards, into the delegator argument of *for*'s; this step is efficient.

With these syntactic restrictions, a more efficient access control algorithm is possible:

- The request is granted if the requester implies one of the ACL entries.
- Each ACL entry is a conjunction of *for*-lists, and so is the requester. For the requester to imply an ACL entry, it must be that for each conjunct of the ACL entry there exists some conjunct of the requester that implies it.
- A *for*-list implies another if they are of the same length, and each principal in roles of the requester implies the corresponding one of the entry.

- A principal in roles Q *as* R_1 *as* $\ldots$ *as* R_n implies another Q' *as* R'_1 *as* $\ldots$ *as* $R'_{n'}$ if Q implies Q' and for each R_j there exists R'_k such that R_j implies R'_k.
- An atomic symbol P implies another atomic symbol P' if there is a chain of assumptions $P = P_0 \Rightarrow \ldots \Rightarrow P_n = P'$.

It is simple to describe this algorithm to users, and it is also simple to implement it. The adequate speed of a prototype implementation suggests that the algorithm might well be practical. The metalogical construct $(\ )^+$ does not introduce any major algorithmic difficulty. Its treatment is well understood, as it is standard in the context of regular languages.

The algorithm is sound for the binary-relation model, where we take all roles to be subsets of the identity. The algorithm is also complete for the binary-relation model. Note in particular that the algorithm treats all roles as idempotent, and they all commute with one another.

Theorem 1 (Soundness and Completeness) *Let $E_0, \ldots, E_i, \ldots$ be an ACL, P a requester, and $\bigwedge_i (B_i \Rightarrow B'_i)$ a conjunction of assumptions. The algorithm grants the request if and only if*

$$\bigwedge_i (B_i \Rightarrow B'_i) \wedge \bigwedge_i (E_i \textit{ controls } p) \supset (P \textit{ controls } p)$$

is valid for all p over binary-relation models.

Proof The soundness proof is simple. The algorithm works by attempting to prove that $P \Rightarrow E_i$ for some i from the assumptions, and does so soundly. Moreover, the truth of $P \Rightarrow E_i$ implies that if $\bigwedge_i (E_i \textit{ controls } p)$ holds then so does P *controls* p.

The completeness proof is clearest when there are no assumptions, and hence we first treat that case. We suppose that the algorithm does not grant a request and obtain that there is a binary-relation model where $\bigwedge_i (E_i \textit{ controls } p)$ holds but P *controls* p does not, for some p. This requires that p be false, that P *says* p be true, and that E_i *says* p be false for all i.

It suffices to find some binary-relation model where $\neg(w_0 \mathcal{R}(P) w_0)$ and $\mathcal{R}(E_i)(w_0) \not\subseteq \mathcal{R}(P)(w_0)$ for all i. Then we can define an interpretation that makes the proposition symbol p true at all w such that $w_0\mathcal{R}(P)w$, and false at all other worlds.

The binary relation model is constructed in a fairly usual way, from strings. Let $T = \mathcal{P} \times 2^{\mathcal{Q}}$, and let S be the set of strings over T. The set S is our set of possible worlds; the empty string is w_0.

If $A_i \in \mathcal{Q}$, then its interpretation $J(A_i)$ is defined by:

$$I(A_i) = \{(x \cdot (A_j, B),\ x \cdot (A_j, B)) \mid A_j \in \mathcal{P},\ A_i \in B \subseteq \mathcal{Q}\}$$

Hence roles are interpreted by subsets of the identity, as required. If $A_i \in \mathcal{P}$ then, instead:

$$J(A_i) = \{(x,\ x \cdot (A_i, B)) \mid B \subseteq \mathcal{Q}\}$$

with the exception of the special constant D:

$$J(D) = \emptyset$$

Thus, in this model, *for* and $|$ are equivalent; this is merely a convenience.

The interpretation of a principal expression Q relates the empty string to all strings

$$(P_n, \{R_n^1, \ldots, R_n^{m_n}\}) \cdot \ldots \cdot (P_1, \{R_1^1, \ldots, R_1^{m_1}\})$$

such that

$$(P_1\ \mathit{as}\ R_1^1\ \mathit{as}\ \ldots\ \mathit{as}\ R_1^{m_1})\ \mathit{for}\ \ldots \mathit{for}\ (P_n\ \mathit{as}\ R_n^1\ \mathit{as}\ \ldots\ \mathit{as}\ R_n^{m_n})$$

is one of the *for*-lists in Q, and also to all strings obtained from the strings

$$(P_n, \{R_n^1, \ldots, R_n^{m_n}\}) \cdot \ldots \cdot (P_1, \{R_1^1, \ldots, R_1^{m_1}\})$$

by enlarging the role sets. Note that in any case the empty string is never related to itself, as the *for*-lists are not empty, so in particular $\neg(w_0\mathcal{R}(P)w_0)$ holds.

Suppose that, for some i, the algorithm does not manage to prove that $P \Rightarrow E_i$. Then there must be some conjunct in E_i that is not implied (according to the algorithm) by any of the conjuncts in P. Suppose this conjunct is:

$$(P_1\ \mathit{as}\ R_1^1\ \mathit{as}\ \ldots\ \mathit{as}\ R_1^{m_1})\ \mathit{for}\ \ldots \mathit{for}\ (P_n\ \mathit{as}\ R_n^1\ \mathit{as}\ \ldots\ \mathit{as}\ R_n^{m_n})$$

This implies that the empty string is related to:

$$(P_n, \{R_n^1, \ldots, R_n^{m_n}\}) \cdot \ldots \cdot (P_1, \{R_1^1, \ldots, R_1^{m_1}\})$$

by the interpretation of E_i. On the other hand, there is no conjunct in P that establishes this relation. Thus, the model satisfies $\mathcal{R}(E_i)(w_0) \not\subseteq \mathcal{R}(P)(w_0)$.

This proves our initial completeness claim. Next we deal with assumptions. Without loss of generality, we can take the assumptions to be transitive, that is, if both $A_i \Rightarrow A_j$ and $A_j \Rightarrow A_k$ are assumed then so is $A_i \Rightarrow A_k$.

Let us replace each symbol A_i in P with the conjunction of all symbols A_j such that $A_i \Rightarrow A_j$ is among the assumptions. Let P' be the expression obtained after this replacement and after normalization (after $\wedge$'s are moved outwards). The algorithm grants the request to P if and only if it grants the request to P'—the additional conjuncts in P' do not add strength in presence of the assumptions. Moreover, the algorithm grants the request to P' using the assumptions if and only if it grants the request without the assumptions—as the additional conjuncts in P' can be exploited to dispense with the assumptions. In short, P obtains access with the assumptions if and only if P' obtains access without them.

Now it suffices to show that a similar equivalence holds at the semantic level. Given a model $\mathcal{M}'$ where $\bigwedge_i(E_i \textit{ controls } p)$ does not imply $P' \textit{ controls } p$, we construct a model $\mathcal{M}$ where all of the assumptions hold and $\bigwedge_i(E_i \textit{ controls } p)$ does not imply $P \textit{ controls } p$. The models $\mathcal{M}$ and $\mathcal{M}'$ differ only in their interpretation functions, $J_{\mathcal{M}}$ and $J_{\mathcal{M}'}$:

$$J_{\mathcal{M}}(A_i) = \bigcup\{J_{\mathcal{M}'}(A_j) \mid A_i \Rightarrow A_j \text{ is among the assumptions}\}$$

Note that all roles are still interpreted as roles.

The meaning of P' in $\mathcal{M}'$ is equal to the meaning of P' in $\mathcal{M}$, and it is equal to the meaning of P in $\mathcal{M}$. Moreover, all of the assumptions hold in $\mathcal{M}$. The meaning of the E_i's in $\mathcal{M}$ is larger than in $\mathcal{M}'$, thus making weaker the assumption $\bigwedge_i(E_i \textit{ controls } p)$. This shows that $\mathcal{M}$ has the desired properties. ▮

6.3 An example

The most typical, complete example is that of a user logging into a workstation, and the workstation making requests on a server.

- The user A authenticates and delegates in some role R_A to a workstation B in role R_B. The user may rely on a smart-card, and use the scheme for user delegation outlined in subsection 6.1. The delegation certificate is:

$$K_A \textit{ says } R_A \textit{ says } ((K_d \wedge (B \textit{ as } R_B)) \textit{ serves } (A \textit{ as } R_A))$$

- The workstation sets up a secure channel to a server. Logically, this requires two statements:

  $$K_d \textit{ says } (A \textit{ as } R_A) \textit{ says } (Ch \Rightarrow ((B \textit{ as } R_B) \textit{ for } (A \textit{ as } R_A)))$$

  under the delegation key, and

  $$K_B \textit{ says } R_B \textit{ says } (A \textit{ as } R_A) \textit{ says } (Ch \Rightarrow ((B \textit{ as } R_B) \textit{ for } (A \textit{ as } R_A)))$$

  (For simplicity, the workstation acts in the role R_B, but any stronger role would do just as well.)

- The server needs to check that K_A is the user's key and that K_B is the workstation's key, that is, $K_A \Rightarrow A$ and $K_B \Rightarrow B$. Typically this requires looking at certificates from a certification authority; it is possible that a chain of certification authorities needs to be involved, as there may be no single universally trusted certification authority.

- With these properties of the keys, it follows from the delegation certificate that:

  $$(A \textit{ as } R_A) \textit{ says } ((K_d \wedge (B \textit{ as } R_B)) \textit{ serves } (A \textit{ as } R_A))$$

  and this statement is believed. The channel set-up certificates yield:

  $$\begin{aligned}&((K_d \wedge (B \textit{ as } R_B))|(A \textit{ as } R_A)) \textit{ says}\\ &\qquad (Ch \Rightarrow ((B \textit{ as } R_B) \textit{ for } (A \textit{ as } R_A)))\end{aligned}$$

  and this leads to

  $$((B \textit{ as } R_B) \textit{ for } (A \textit{ as } R_A)) \textit{ says } (Ch \Rightarrow ((B \textit{ as } R_B) \textit{ for } (A \textit{ as } R_A)))$$

  and then

  $$Ch \Rightarrow ((B \textit{ as } R_B) \textit{ for } (A \textit{ as } R_A))$$

  as $(B \textit{ as } R_B) \textit{ for } (A \textit{ as } R_A)$ is trusted on this matter.

- The user may indicate to the workstation that he wishes to reduce his privileges, and adopt a further role R'_A, in order to make a request r; or the workstation may do this on behalf of the user, on its own initiative:

$$(Ch \textit{ as } R'_A) \textit{ says } r$$

  The requester here is $((B \textit{ as } R_B) \textit{ for } (A \textit{ as } R_A)) \textit{ as } R'_A$, which is equivalent to $(B \textit{ as } R_B) \textit{ for } (A \textit{ as } R_A \textit{ as } R'_A)$.

- The ACL at the server may contain $(G' \textit{ as } R_B) \textit{ for } (G \textit{ as } R''_A)$. The server may have, or the workstation may present, certificates that prove that $A \Rightarrow G$, $R_A \Rightarrow R''_A$, $R'_A \Rightarrow R''_A$, and $B \Rightarrow G'$.

- Actually, the group membership certificates come signed with someone's public key. In each case it must be possible to resolve this public key into a name, and then to discover that the name is that of someone who is trusted to certify membership in the group.

- At this point, the algorithm for access control can easily determine that access should be granted.

7 Extensions

Our main goal in this paper has been to isolate some useful and mathematically tractable concepts. We have only touched on many practical and theoretical matters that deserve more detailed treatments. In addition, the basic logic could be extended in many interesting ways. To conclude, we briefly describe three extensions.

7.1 Intersection

An intersection operation $\cap$ permits the construction of groups from groups. It is not a logical connective in the sense that conjunction is, as it can only be applied to atomic symbols (so for example $(A_j|A_i) \cap A_k$ is not a legal expression).

Conjunction is strictly weaker than intersection: $(A \cap B) \Rightarrow (A \wedge B)$ is valid but $(A \wedge B) \Rightarrow (A \cap B)$ is not. A weaker property than $(A \wedge B) \Rightarrow (A \cap B)$ holds: if $C \Rightarrow A$ and $C \Rightarrow B$ then $C \Rightarrow (A \cap B)$ provided C is in some sense atomic. Atomicity is not a logically defined notion; intuitively, simple principals such as users and machines are atomic, while the conjunction of two such principals is not.

An application of intersection concerns restricting access to only a particular member of a group. The ACL entry $A \wedge G$ grants access to A if A is a member of G, but it also grants access to $A \wedge B$ if B is a member of G. In contrast, $A \cap G$ grants access to A if A is a member of G, but not to $A \wedge B$ even if B is a member of G.

7.2 Subtraction

Another important operation on groups is subtraction ($-$). It provides the ability to specify that certain named individuals or subgroups within a supergroup should be denied access, even when such access is granted to the supergroup. Subtraction is the only negative construct for ACLs in this paper.

Subtraction has a simple semantics in centralized systems, where it is easy to decide group membership. There, subtraction is simply set subtraction. The situation is different in distributed systems, because in general it is possible to prove only membership in groups, but not nonmembership.

Consider, for example, a situation in which $G'' - G$ appears in an ACL, with $A \Rightarrow G$, $A \Rightarrow G'$, $G \Rightarrow G''$, and $G' \Rightarrow G''$. Should access be granted to A? The facts $A \Rightarrow G$ and $G \Rightarrow G''$ may not be available to the procedure making the access control decision. Moreover, we cannot expect A to provide proofs for them—as this is probably of no benefit to A. Therefore, the access control decision cannot depend on $A \Rightarrow G$ and $G \Rightarrow G''$, and hence access should be granted in this case. In short, $G'' - G$ means "all members of G'', except for those that are members of G'' only via G."

This definition of subtraction is somewhat operational. Nevertheless, it is the best possible, and it has even been suggested that it provides an intuitive semantics for subtraction.

Even in distributed systems, there are groups for which it is possible to prove both membership and nonmembership. Typically, these groups are those managed by fairly centralized subsystems of the distributed system. The traditional set subtraction operation can be made available for these groups.

7.3 Variables

So far, we have provided for only the most basic form of joint signatures, and a more general form would be useful. One would like to be able to express that access should be granted to any two different members of a given group when they sign jointly, without listing all allowed combinations. Moreover, one may require that the two signers bear a certain relation to one another.

The introduction of variables provides the desired expressiveness, and more. Variables can be included in ACLs. When a requester matches an ACL entry, we identify the parts of the requester that match against each of the variables in the ACL entry, and bind the variables to these parts. It is then possible to evaluate arbitrary predicates on these variables.

For example, the entry $y|x$ is equivalent to $G|F$ when we require that $x \Rightarrow F$ and $y \Rightarrow G$. Other requirements on x and y can be added, for instance that x and y be different, or that y be a machine owned by x. (Intuitively, it becomes possible to generalize from unary relations to arbitrary ones.) The entry has an implicit universal quantifier: it means that, for all x and y, if x and y satisfy all requirements then $(y|x)$ *controls* s, where s is the statement under consideration. Trivially, $B|A$ matches $y|x$; access is granted if the requirements hold when x is replaced with A and y with B.

The manipulation of variables can be added to the algorithm of subsection 6.2.2. There are some complications, however. In particular, some metalogical construct to handle lists of principal expressions is needed if variables are to be legal in the scope of the iteration construct $(\ )^+$.

The use of variables may not be the only or the best way to provide the desired expressiveness. It seems to deserve further study.

Glossary

A *says* s: A makes the statement s.

A *controls* s: A is trusted on s: if A *says* s then s. This is the meaning of A appearing in the ACL for s.

$A \Rightarrow B$: A is a stronger principal than B. (For example, when B represents a group, A may represent a member or a subgroup of B.) If A *says* s then B *says* s, and if B *controls* s then A *controls* s.

$A = B$: $A \Rightarrow B$ and $B \Rightarrow A$.

B *serves* A: B is a delegate for A. This can be defined as $(B|A \Rightarrow B \textit{ for } A)$.

$A \wedge B$: A and B in conjunction; $(A \wedge B)$ *says* s if and only if A *says* s and B *says* s.

$B|A$: B quoting A; $(B|A)$ *says* s if and only if B *says* A *says* s.

B *for* A: B on behalf of A. This can be defined in terms of a fictional delegation server D, as $(B \wedge D)|A$. If B *says* A *says* s and B *serves* A then $(B \textit{ for } A)$ *says* s.

A *as* R: A in role R. This can be encoded as $A|R$.

$A \cap B$: the intersection of A and B, if A and B are atomic symbols.

$A - B$: A minus B, if A and B are atomic symbols.

Acknowledgements

Morrie Gasser, Andy Goldstein, and Charlie Kaufman were at the origin of many of the ideas discussed here. Tim Mann, Garret Swart, and Ted Wobber participated in fruitful discussions about implementations and examples. Bill Shockley convinced us of the practical importance of the extension described in subsection 7.3. Greg Nelson and Vaughan Pratt helped in understanding the algebra of principals. Roger Maddux told us about Andréka's work. Luca Cardelli and Cynthia Hibbard provided editorial suggestions.

References

[1] M. Abadi and G. Plotkin. A Logical View of Composition and Refinement. Proceedings of the Eighteenth ACM Symposium on Principles of Programming Languages, January 1991, pp. 323–332.

[2] M. Abadi, M. Burrows, C. Kaufman, and B. Lampson. Authentication and Delegation with Smart-Cards. In *Theoretical Aspects of Computer Software*, Springer-Verlag LNCS 526, September 1991, pp. 326–345.

[3] H. Andréka. Representations of Distributive Lattice-ordered Semigroups with Binary Relations. Manuscript, August 1989.

[4] M. Burrows, M. Abadi, and R.M. Needham. A Logic of Authentication. *Proceedings of the Royal Society of London A* Vol. 426, 1989, pp. 233–271.

[5] CCITT. CCITT Blue Book, Recommendation X.509 and ISO 9594-8: The Directory-Authentication Framework. Geneva, March 1988.

[6] A. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *JACM* Vol. 28, No. 1, January 1981, pp. 114-133.

[7] M. Dam. Relevance Logic and Concurrent Computation. Proceedings of the Third IEEE Symposium on Logic in Computer Science, July 1988, pp. 178–185.

[8] National Bureau of Standards. Data Encryption Standard. Fed. Inform. Processing Standards Pub. 46. Washington DC, January 1977.

[9] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory* IT-22, No. 6, November, 1976, pp. 644–654.

[10] R. Fabry. Capability-based Addressing. *CACM* Vol. 17, No. 7, July 1974, pp. 403-412.

[11] M. Gasser, A. Goldstein, C. Kaufman, and B. Lampson. The Digital Distributed System Security Architecture. Proceedings of the 1989 National Computer Security Conference, October 1989, pp. 305-319.

[12] M. Gasser and E. McDermott. An Architecture for Practical Delegation in a Distributed System. Proceedings of the 1990 IEEE Symposium on Security and Privacy, May 1990, pp. 20–30.

[13] J.-Y. Girard. Linear Logic. *Theoretical Computer Science* Vol. 50, 1987, pp. 1-102.

[14] G.E. Hughes and M.J. Cresswell. An Introduction to Modal Logic. Methuen Inc., New York, 1968.

[15] J. Kohl, C. Neuman, and J. Steiner. The Kerberos Network Authentication Service (version 5, draft 3). Available by anonymous ftp from athena-dist.mit.edu as /pub/doc/kerberos/V5DRAFT3-RFC.{PS,TXT}, October 1990.

[16] D. Kozen. A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events. Cornell TR90-1123, May 1990.

[17] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in Distributed Systems: Theory and Practice. Proceedings of the Thirteenth Symposium on Operating System Principles, October 1991, pp. 165–182.

[18] H. Levy. Capability-based Computer Systems. Digital Press, 1983.

[19] S.P. Miller, C. Neuman, J.I. Schiller, and J.H. Saltzer. Kerberos Authentication and Authorization System. *Project Athena Technical Plan* Section E.2.1, MIT, July 1987.

[20] R.M. Needham and M.D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *CACM* Vol. 21, No. 12, December 1978, pp. 993–999.

[21] V. Pratt. Dynamic Algebras as a Well-behaved Fragment of Relation Algebras. In *Algebraic Logic and Universal Algebra in Computer Science*, Springer-Verlag LNCS 425, 1990.

[22] R.L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *CACM* Vol. 21, No. 2, February 1978, pp. 120-126.

[23] J. Saltzer and M. Schroeder. The Protection of Information in Computer Systems. *Proceedings of the IEEE* Vol. 63, No. 9, September 1975, pp. 1278-1308.

[24] K. Sollins. Cascaded Authentication. Proceedings of the 1988 IEEE Symposium on Security and Privacy, April 1988, pp. 156-163.

[25] S. Vickers. Samson Abramsky on Linear Process Logics. Foundation Workshop Notes, October-November 1988.

[26] D.N. Yetter. Quantales and (Noncommutative) Linear Logic. *Journal of Symbolic Logic* Vol. 55, No. 1, March 1990, pp. 41-64.