Logic in Access Control
(Tutorial Notes)

Martín Abadi

Microsoft Research
and
University of California, Santa Cruz

Abstract. Access control is central to security in computer systems. Over the years, there have been many efforts to explain and to improve access control, sometimes with logical ideas and tools. This paper is a partial survey and discussion of the role of logic in access control. It considers logical foundations for access control and their applications, in particular in languages for security policies. It focuses on some specific logics and their properties. It is intended as a written counterpart to a tutorial given at the 2009 International School on Foundations of Security Analysis and Design.

1 Introduction

Access control consists in deciding whether the agent that issues a request should be trusted on this request. For example, the agent may be a process running on behalf of a user, and the request may be a command to read a particular file. In this example, the access control machinery would be charged with deciding whether the read should be permitted. This authorization decision may, in the simplest case, rely on consulting an access control matrix that would map the user's name and the file name to a set of allowed operations [44]. The matrix may be implemented in terms of access control lists (ACLs), attached to objects, or in terms of capabilities, held by principals. Typically, however, the authorization decision is considerably more complex. It may depend, for example, on the user's membership in a group, and on a digitally signed credential that certifies this membership.

Access control is central to security and it is pervasive in computer systems. It appears (with peculiar features and flaws) in many applications, virtual machines, operating systems, and firewalls. Physical protection for facilities and for hardware components are other forms of access control.

Although access control may sometimes seem conceptually straightforward, it is both complex and error-prone in practice. The many mechanisms for access control are often broken or circumvented.

Over the years, there have been many efforts to explain and to improve access control. Some of those efforts have relied on logical ideas and tools. One may hope that logic would provide a simple, solid, and general foundation for access control, as well as methods for designing, implementing, and validating particular access control mechanisms. Indeed, although logic is not a panacea, its applications in research on access control have been substantial and beneficial.

This paper is a partial survey and discussion of the role of logic in access control. It considers logical formulations of access control and their applications, emphasizing some particular logics and languages for security policies in distributed systems. It does not however aim to be a complete overview. It deliberately neglects several relevant topics that have been the subjects of significant bodies of work. These include:

– Decidability results for problems related to access control (e.g., [36, 49, 51]).
– Logical approaches for authorizing code execution, such as those based on proof-carrying code (e.g., [53, 59]).
– Formal verification of security properties (e.g., [56]).

The next section introduces some logical constructs that have been employed in connection with access control. Section 3 explains, at a high level, some of the choices that need to be made in order to define a logic for access control. Section 4 defines some specific logics. Section 5 briefly discusses non-interference properties. Section 6 considers the relations between various formulas in these logics. Section 7 then describes languages for access control, focusing on the Binder language. Section 8 concludes.

Some of the basic material in this paper is an adaptation and update of a brief survey from 2003 [2]. Other parts summarize recent research papers [3, 4], which contain proofs and additional details.

2 From Matrices to Logics

This section aims to explain, informally, the transition from access control matrices to logics for access control. Subsequent sections define and study these logics more precisely, and also contain further examples.

2.1 From Matrices to Predicates

An access control matrix may be viewed as a description of a ternary relation, which we call `may-access`. With this interpretation, $\texttt{may-access}(p, o, r)$ would hold whenever the matrix gives principal p the right r on object o. Thus, we may obtain a first logic of access control by representing a global access control matrix with the predicate symbol `may-access`, in the setting of a predicate calculus.

This trivial logic enables us to state facts such as

$$\texttt{may-access}(\texttt{Alice}, \texttt{Foo.txt}, \texttt{Rd})$$

(which says that the principal `Alice` can perform the operation read (`Rd`) on the object `Foo.txt`). It also enables us to state rules such as

$$\texttt{may-access}(p, o, \texttt{Wr}) \rightarrow \texttt{may-access}(p, o, \texttt{Rd})$$

(which makes the `Wr` right stronger than the `Rd` right), but perhaps not much else. Therefore, this trivial logic seems of limited direct benefit. However, it suggests more elaborate systems with predicate symbols similar to `may-access`.

We may reasonably suspect that there is nothing canonical about `may-access`. We may also worry about a proliferation of variants. For instance, thinking about security policies that require separation of duty, we may imagine a predicate symbol `may-jointly-access`, with the intent that `may-jointly-access`(p, q, o, r) would hold if p and q have right r on o when they request r jointly. In addition, we may imagine many useful auxiliary predicates, such as one for expressing that a principal owns an object, and several for grouping principals and objects. We would perhaps be reluctant to develop a logic with the many built-in constructs and axioms that would result.

Nevertheless, rich logics with constructs similar to `may-access` can support a wide range of current models for access control [15, 39, 40]. In addition to primitives for authorization that greatly generalize `may-access`, the logics may include primitives pertaining to groups, roles, object containment, privilege ordering, and perhaps others. In a different direction, such a logic for access control may include a modal operator for reasoning about necessity [21]. There is much room for logical creativity, for better or for worse—richer logics are not automatically more tasteful or more useful.

2.2 Saying Predicates

Several characteristics of computer systems complicate access control. For distributed systems, in particular, these characteristics include size, heterogeneity, the autonomy of system components, and the possibility of component failures (e.g., [30, 43]). These complications have resulted in a substantial line of work. They also suggest the possibility of an important role for logical methods.

In what follows we focus on one logical construct that has often been used in this context, `says` (e.g., [2, 6, 8, 12, 13, 20, 22, 29, 33, 43, 46, 48, 54, 63]). The formula p `says` s represents that principal p makes statement s. This statement may simply be a request for an operation. In more interesting cases, the statement may express that p delegates some of its authority to another principal q, or it may express part of a security policy.

For instance, we may write:

$$p\ \texttt{says}\ \texttt{may-access}(q, o, r)$$

We may also write the rule:

$$p\ \texttt{says}\ \texttt{may-access}(q, o, r) \rightarrow (\texttt{may-access}(p, o, r) \rightarrow \texttt{may-access}(q, o, r))$$

which states that p may hand off a right r to q. As in these formulas, the use of `says` enables us to consider situations—common in distributed systems—in which there is no global, universally trusted access control policy. Each principal can make its own security-relevant assertions.

In a slightly more substantial and specific example (partly adapted from [28]), we consider a principal `Admin` (an administrator), a user `Bob`, one file `File1`, and the following informal policy:

1. If `Admin` says that `File1` should be deleted, then this must be the case.

2. `Admin` trusts `Bob` to decide whether `File1` should be deleted.

Several different formalizations of these statements are possible. For instance, much as above, we may write:

1. $\texttt{may-access}(\texttt{Admin}, \texttt{File1}, \texttt{Delete})$
2. $\texttt{Admin}\ \texttt{says}\ \texttt{may-access}(\texttt{Bob}, \texttt{File1}, \texttt{Delete})$

Alternatively, letting the proposition symbol `deleteFile1` mean that `File1` should be deleted, we may write:

1. $(\texttt{Admin}\ \texttt{says}\ \texttt{deleteFile1}) \rightarrow \texttt{deleteFile1}$
2. $\texttt{Admin}\ \texttt{says}\ ((\texttt{Bob}\ \texttt{says}\ \texttt{deleteFile1}) \rightarrow \texttt{deleteFile1})$

This second representation allows us to go further, in the following way. Suppose that `Bob` wants to delete `File1`. We may represent this statement by

$$\texttt{Bob}\ \texttt{says}\ \texttt{deleteFile1}$$

We may then expect to conclude that `deleteFile1`, so `File1` should be deleted. Some of the logics described below (in particular, CDD, of Section 4.3) do indeed yield this conclusion. A reference monitor that controls access to `File1` and that bases its decisions on such a logic could therefore derive `deleteFile1` from the other formulas. `Bob` and other principals may however help with this proof, providing the whole proof of `deleteFile1` or some of its parts; this approach is sometimes called proof-carrying authentication or proof-carrying authorization [8, 9].

Attractively, `says` abstracts from the details of authentication. When p `says` s, p may transmit s in a variety of ways:

– on a local channel via a trusted operating system within a computer,
– on a physically secure channel between two machines,
– on a channel secured with shared-key cryptography, or
– in a certificate with a public-key digital signature.

We may assert that p `says` s even when p does not directly produce s. For example, when p is a user and one of its programs sends s in a message, we may find it convenient and reasonably accurate to state that p `says` s although p itself may never have even seen s. In this case, p `says` s means that p has caused s to be said, or that s has been said on p's behalf, or that p supports s. With this interpretation, we may assert that p `says` s also when p makes a statement s' stronger than s, with the idea that if p supports s' then p must also support s.

If p `says` s and p speaks for another principal q, then q `says` s. The "speaks for" relation serves to form chains of responsibility in many important situations [45]. A program may speak for a user, much like a key may speak for its owner, much like a channel may speak for its remote end-point. Therefore, some logics support "speaks for" as a primitive (e.g., [6, 12, 43]) or as a definable construct (e.g., [3, 8]).

Revisiting our example, suppose that instead of having `Bob says deleteFile1` directly, `Bob` hands off his authority to `Alice` who wants to delete `File1`. Writing $\Rightarrow$ for the "speaks for" relation, we may express these statements by:

1. $\texttt{Bob}\ \mathsf{says}\ (\texttt{Alice} \Rightarrow \texttt{Bob})$
2. $\texttt{Alice}\ \mathsf{says}\ \texttt{deleteFile1}$

From these, we may be able to derive $\texttt{Bob}\ \mathsf{says}\ \texttt{deleteFile1}$, and then to derive $\texttt{deleteFile1}$. Again, some of the logics described below (in particular, CDD) do indeed yield this conclusion. Note that, in this example, $\texttt{Bob}\ \mathsf{says}\ \texttt{deleteFile1}$ does not mean that $\texttt{Bob}$ utters the request to delete the file.

3 Defining a Logic (Preliminaries)

Existing logics differ in sometimes subtle but important ways in their syntax, in their axioms and proof rules, and in the intended interpretations of says and related constructs. For instance, in some logics says requires no special axioms and is treated quite syntactically, like the $\texttt{cert}$ construct of Halpern and van der Meyden [34]. Elsewhere, sometimes says has axioms familiar from modal logics [38], such as the axiom of closure under consequence:

$$p\ \mathsf{says}\ (s \to s') \to (p\ \mathsf{says}\ s) \to (p\ \mathsf{says}\ s')$$

and the usual necessitation rule according to which the validity of s implies the validity of $p\ \mathsf{says}\ s$. Sometimes says has additional properties. For instance, early on, Lampson suggested the axiom:

$$s \to (p\ \mathsf{says}\ s)$$

Appel and Felten essentially adopt this axiom (as rule name_i) in their work on proof-carrying authentication [8]. It is stronger than the usual necessitation rule, and should be used with caution (if at all). In a classical-logic context, it can yield unexpected consequences such as:

$$(p\ \mathsf{says}\ s) \to (s \vee (p\ \mathsf{says}\ \texttt{false}))$$

We explore this and related issues in Section 6.

One may imagine that rigorous semantics would clarify the intended interpretations of logical constructs, and that they would also shed some light on the proper choice of syntax, axioms, and proof rules. While semantics have indeed been helpful (for instance, for decidability proofs [6, 28], in relating two definitions of “speaks for” [28], and in establishing some of the results in this paper), so far they provide only limited new insight into notions such as authority and responsibility. Therefore, we tend to use semantics mostly informally or in proofs of metatheorems.

Various high-level desiderata may also influence the choice of syntax, axioms, and proof rules of a logic:

– The logic should be consistent (that is, it should not prove $\texttt{false}$). It should also be sensible in other ways. For instance, if $\texttt{Alice}$ and $\texttt{Bob}$ are unrelated principals, $\texttt{Alice}\ \mathsf{says}\ \texttt{false}$ should not imply $\texttt{Bob}\ \mathsf{says}\ \texttt{false}$. This property can be regarded as a non-interference property [5, 29]. (We discuss non-interference in Section 5.)

– Human users should find it easy to write security policies and related assertions, manipulate them, and understand them (e.g., [10]).
– Programs, such as decision procedures, should also be able to manipulate and analyze logical formulas.

These criteria are not fully orthogonal. In particular, one cannot express meaningful security policies in an inconsistent logic. On the other hand, expressiveness often conflicts with algorithmic tractability.

A common approach to addressing these desiderata consists in restricting the forms of logical formulas. In particular, the formulas may be required to be similar to Horn clauses, much as in logic programming. The Binder language of Section 7 can be seen partly as an example of this approach. More general formulas may be allowed for certain purposes (for instance, in theoretical results) but not necessarily used on a day-to-day basis, or entirely disallowed.

4 Some Logics

For concreteness, this section introduces specific logics, more formally. The logics have the same operators and intended applications, but they differ in their axioms and rules. Section 4.1 briefly introduces syntax. Section 4.2 concerns modal treatments of `says`, both intuitionistic and classical. Section 4.3 concerns another logic called CDD. It presents CDD in a self-contained manner, but it also mentions how to view CDD as an extension of the intuitionistic logic of Section 4.2.

4.1 Basic Syntax

The syntax of the logics includes that of propositional logic, second-order quantification over propositions, and the `says` operator. More precisely, formulas are given by the grammar:

$$s ::= \texttt{true} \mid (s \vee s) \mid (s \wedge s) \mid (s \rightarrow s) \mid p \ \texttt{says} \ s \mid X \mid \forall X.\, s$$

where p ranges over elements of a set $\mathcal{P}$ (intuitively the principals), and X ranges over a set of variables. The variable X is bound in $\forall X.\, s$, and subject to renaming.

We write `false` for $\forall X.\, X$. We write $s_1 \equiv s_2$ for $(s_1 \rightarrow s_2) \wedge (s_2 \rightarrow s_1)$. We write $p \Rightarrow q$ as an abbreviation for

$$\forall X.\, (p \ \texttt{says} \ X \rightarrow q \ \texttt{says} \ X)$$

This formula is our representation of “p speaks for q”. We write p `controls` s as an abbreviation for $(p \ \texttt{says} \ s) \rightarrow s$.

4.2 Second-Order Propositional Modal Logics

Our starting point is second-order propositional intuitionistic logic. This logic can be presented as a Hilbert system, with the following axioms:

- $\mathtt{true}$
- $s_1 \to (s_2 \to s_1)$
- $(s_1 \to (s_2 \to s_3)) \to ((s_1 \to s_2) \to (s_1 \to s_3))$
- $(s_1 \land s_2) \to s_1$
- $(s_1 \land s_2) \to s_2$
- $s_1 \to s_2 \to (s_1 \land s_2)$
- $s_1 \to (s_1 \lor s_2)$
- $s_2 \to (s_1 \lor s_2)$
- $(s_1 \to s_3) \to ((s_2 \to s_3) \to ((s_1 \lor s_2) \to s_3))$
- $(\forall X.\, s) \to s[t/X]$
- $(\forall X.\, (s_1 \to s_2)) \to (s_1 \to \forall X.\, s_2)$ where X is not free in s_1

and the rules of modus ponens and universal generalization:

$$\frac{s_1 \qquad s_1 \to s_2}{s_2} \qquad\qquad \frac{s}{\forall X.\, s}$$

A classical variant is obtained by adding the axiom of excluded middle:

$$[\textit{Excluded-middle}] \quad \forall X.\, (X \lor (X \to \mathtt{false}))$$

Going beyond these standard propositional systems, we axiomatize the operator says as a modality, with the axiom of closure under consequence:

$$\forall X, Y.\, ((p \;\mathtt{says}\; (X \to Y)) \to (p \;\mathtt{says}\; X) \to (p \;\mathtt{says}\; Y))$$

and the necessitation rule:

$$\frac{s}{p \;\mathtt{says}\; s}$$

More precisely, says is a modality indexed over $\mathcal{P}$: in other words, p says is a modality for each $p \in \mathcal{P}$. Thus, we obtain second-order, indexed versions (intuitionistic and classical) of the standard propositional modal logic K [38].

Below, we sometimes refer to the intuitionistic version as "the basic logic", because it is the weakest logic that we use for access control in this paper. Occasionally we emphasize that this logic does not include Excluded-middle, particularly when the inclusion of Excluded-middle would affect the results under consideration.

4.3 CDD

CDD is a formalism related to lax logic and the computational lambda calculus [14, 24, 52]. CDD can be seen both as a type system and as a logic, via the Curry-Howard isomorphism. Figure 1 presents CDD as a set of typing rules, in the style of those of Girard's System F [18, 31]. There, an environment Γ declares a list of distinct type variables and distinct variables with their types (for example, X, $x : X \to \mathtt{true}$, $y : X$). Figure 2 presents CDD as a logical system, with sequent notation. There, an environment Γ is a list of formulas (for example, $X \to \mathtt{true}, X$). When Γ is empty, we may write $\vdash s$, and say that s is a theorem, when $\vdash s$ is derivable by the rules of Figure 2.

$$[\textit{Var}] \quad \Gamma, x : s, \Gamma' \vdash x : s \qquad [\textit{Unit}] \quad \Gamma \vdash () : \texttt{true}$$

$$[\textit{Lam}] \quad \frac{\Gamma, x : s_1 \vdash e : s_2}{\Gamma \vdash (\lambda x : s_1.\, e) : (s_1 \to s_2)} \qquad [\textit{App}] \quad \frac{\Gamma \vdash e : (s_1 \to s_2) \quad \Gamma \vdash e' : s_1}{\Gamma \vdash (e\ e') : s_2}$$

$$[\textit{Pair}] \quad \frac{\Gamma \vdash e_1 : s_1 \quad \Gamma \vdash e_2 : s_2}{\Gamma \vdash \langle e_1, e_2 \rangle : (s_1 \wedge s_2)}$$

$$[\textit{Proj 1}] \quad \frac{\Gamma \vdash e : (s_1 \wedge s_2)}{\Gamma \vdash (\texttt{proj}_1\ e) : s_1} \qquad [\textit{Proj 2}] \quad \frac{\Gamma \vdash e : (s_1 \wedge s_2)}{\Gamma \vdash (\texttt{proj}_2\ e) : s_2}$$

$$[\textit{Inj 1}] \quad \frac{\Gamma \vdash e : s_1}{\Gamma \vdash (\texttt{inj}_1\, e) : (s_1 \vee s_2)} \qquad [\textit{Inj 2}] \quad \frac{\Gamma \vdash e : s_2}{\Gamma \vdash (\texttt{inj}_2\, e) : (s_1 \vee s_2)}$$

$$[\textit{Case}] \quad \frac{\Gamma \vdash e : (s_1 \vee s_2) \quad \Gamma, x : s_1 \vdash e_1 : s \quad \Gamma, x : s_2 \vdash e_2 : s}{\Gamma \vdash (\texttt{case}\ e\ \texttt{of}\ \texttt{inj}_1(x).\ e_1 \mid \texttt{inj}_2(x).\ e_2) : s}$$

$$[\textit{TLam}] \quad \frac{\Gamma, X \vdash e : s}{\Gamma \vdash (\Lambda X.\, e) : \forall X.\, s} \qquad [\textit{TApp}] \quad \frac{\Gamma \vdash e : \forall X.\, s}{\Gamma \vdash (et) : s[t/X]} \ \text{(t well-formed in Γ)}$$

$$[\textit{UnitM}] \quad \frac{\Gamma \vdash e : s}{\Gamma \vdash (\eta_p\ e) : p\ \texttt{says}\ s}$$

$$[\textit{BindM}] \quad \frac{\Gamma \vdash e : p\ \texttt{says}\ s \quad \Gamma, x : s \vdash e' : p\ \texttt{says}\ t}{\Gamma \vdash \texttt{bind}\ x = e\ \texttt{in}\ e' : p\ \texttt{says}\ t}$$

Fig. 1. CDD typing rules.

[*Var*] $\mathit{\Gamma}, s, \mathit{\Gamma}' \vdash s$ [*Unit*] $\mathit{\Gamma} \vdash \texttt{true}$

[*Lam*] $\dfrac{\mathit{\Gamma}, s_1 \vdash s_2}{\mathit{\Gamma} \vdash (s_1 \to s_2)}$ [*App*] $\dfrac{\mathit{\Gamma} \vdash (s_1 \to s_2) \quad \mathit{\Gamma} \vdash s_1}{\mathit{\Gamma} \vdash s_2}$

[*Pair*] $\dfrac{\mathit{\Gamma} \vdash s_1 \quad \mathit{\Gamma} \vdash s_2}{\mathit{\Gamma} \vdash (s_1 \wedge s_2)}$

[*Proj 1*] $\dfrac{\mathit{\Gamma} \vdash (s_1 \wedge s_2)}{\mathit{\Gamma} \vdash s_1}$ [*Proj 2*] $\dfrac{\mathit{\Gamma} \vdash (s_1 \wedge s_2)}{\mathit{\Gamma} \vdash s_2}$

[*Inj 1*] $\dfrac{\mathit{\Gamma} \vdash s_1}{\mathit{\Gamma} \vdash (s_1 \vee s_2)}$ [*Inj 2*] $\dfrac{\mathit{\Gamma} \vdash s_2}{\mathit{\Gamma} \vdash (s_1 \vee s_2)}$

[*Case*] $\dfrac{\mathit{\Gamma} \vdash (s_1 \vee s_2) \quad \mathit{\Gamma}, s_1 \vdash s \quad \mathit{\Gamma}, s_2 \vdash s}{\mathit{\Gamma} \vdash s}$

[*TLam*] $\dfrac{\mathit{\Gamma} \vdash s}{\mathit{\Gamma} \vdash \forall X.\, s}$ (X not free in $\mathit{\Gamma}$) [*TApp*] $\dfrac{\mathit{\Gamma} \vdash \forall X.\, s}{\mathit{\Gamma} \vdash s[t/X]}$

[*UnitM*] $\dfrac{\mathit{\Gamma} \vdash s}{\mathit{\Gamma} \vdash p \ \mathsf{says}\ s}$

[*BindM*] $\dfrac{\mathit{\Gamma} \vdash p \ \mathsf{says}\ s \quad \mathit{\Gamma}, s \vdash p \ \mathsf{says}\ t}{\mathit{\Gamma} \vdash p \ \mathsf{says}\ t}$

Fig. 2. CDD logical rules.

$$
\begin{aligned}
(\texttt{true})^q &= \texttt{true}\\
(s_1 \vee s_2)^q &= (s_1)^q \vee (s_2)^q\\
(s_1 \wedge s_2)^q &= (s_1)^q \wedge (s_2)^q\\
(s_1 \to s_2)^q &= (s_1)^q \to (s_2)^q\\
(p \; \texttt{says} \; s)^q &= \begin{cases} \texttt{true} & \text{if } q = p\\ p \; \texttt{says} \; (s)^q & \text{otherwise} \end{cases}\\
(X)^q &= X\\
(\forall X.\, s)^q &= \forall X.\, (s)^q
\end{aligned}
$$

Fig. 3. Definition of $(s)^q$.

The rules of Figure 2 are obtained from those of Figure 1 by omitting type-variable declarations and terms.

In the context of access control, CDD arose as a simplified version of the Dependency Core Calculus (DCC) [5], but it is similarly adequate as a logic for access control [3, Section 8]. (CDD is simpler than DCC and a little weaker: for instance, DCC proves $(p \; \texttt{says} \; q \; \texttt{says} \; s) \to (q \; \texttt{says} \; p \; \texttt{says} \; s)$, and CDD does not.) CDD has been used for language-based authorization [25], and its central rules also appear in other systems for access control, such as Alpaca [46].

While Figure 2 is a complete, stand-alone presentation of CDD, CDD may also be seen as an extension of the intuitionistic logic of Section 4.2. The extension consists in adopting the following two additional axioms, Unit and Bind:

[*Unit*] $\forall X.\, (X \to p \; \texttt{says} \; X)$
[*Bind*] $\forall X, Y.\, ((X \to p \; \texttt{says} \; Y) \to (p \; \texttt{says} \; X) \to (p \; \texttt{says} \; Y))$

These axioms straightforwardly correspond to the rules *UnitM* and *BindM*. It is easy to show that neither of these axioms is derivable in the logic of Section 4.2, neither intuitionistically nor classically.

5 Non-interference

The Dependency Core Calculus (DCC) [3, 5] was not initially designed as a calculus for access control. Rather, it was designed in order to capture the notion of dependency that arises in information-flow control, partial evaluation, slicing, and similar programming-language settings. In those contexts, non-interference results characterize independence properties. Interestingly, non-interference results are also relevant to access control.

A typical non-interference result would imply that if we have a proof e of $p \; \texttt{says} \; s$ and it depends on a proof x of $q \; \texttt{says} \; t$, where p and q are unrelated principals, then, from the point of view of e, it does not matter which actual proof we substitute for x. Even more strongly, we should be able to obtain that e can be constructed without x (at least under certain hypotheses on e).

As an example, we show a non-interference result for CDD. For a formula s and principal q, we define the formula $(s)^q$ in Figure 3. Intuitively, $(s)^q$ is a variant of s that corresponds to the situation in which q is completely untrustworthy, so q `says` t is always true, independently of t. For an environment Γ, we define $(\Gamma)^q$ by applying $(\cdot)^q$ to each formula in Γ. Using these definitions, Theorem 1 aims to show that q's untrustworthiness has a limited effect on other principals:

Theorem 1. *In CDD, for every typing environment Γ, formula s, and principal q, if $\Gamma \vdash e : s$ then there exists e' such that $(\Gamma)^q \vdash e' : (s)^q$.*

As a special case, we derive the following corollary from the theorem:

Corollary 1. *In CDD, for every formula s and principal q, if $\vdash s$ then $\vdash (s)^q$.*

For example, Corollary 1 says that if $q \neq p$, then

$$\vdash (q\ \texttt{says}\ t) \rightarrow (p\ \texttt{says}\ \texttt{false})$$

implies

$$\vdash \texttt{true} \rightarrow (p\ \texttt{says}\ \texttt{false})$$

and therefore

$$\vdash p\ \texttt{says}\ \texttt{false}$$

Via a simple translation to System F, we can prove that this judgment is not derivable in CDD, so we conclude that

$$\vdash (q\ \texttt{says}\ t) \rightarrow (p\ \texttt{says}\ \texttt{false})$$

is not derivable in CDD either. Thus, no matter what q says, p does not say `false`.

Such non-interference results are not unique to CDD. Analogous theorems hold for other logics for access control. In fact, one might be inclined to regard with some suspicion a logic for which an analogous theorem does not hold.

6 Relating Axioms

Perhaps because intuitive explanations of `says` are invariably loose and open-ended, the exact properties that `says` should satisfy do not seem obvious, as indicated in Section 3. The goal of this section is to explore the formal consequences and the security interpretations of several possible axiomatizations, and thus to help in identifying logics that are sufficiently strong but not inconsistent, degenerate, or otherwise unreasonable.

Some of the axioms that we consider are those of Section 4, which come from modal logic, computational lambda calculus, and other standard formal systems. Other axioms stem from ideas in security, such as delegations of authority and the Principle of Least Privilege [57]. For instance, we consider the hand-off axiom, which says that if p says that q speaks for p, then q does speak for p [43]. We evaluate these axioms in both classical and intuitionistic contexts.

More specifically, we start with the basic axioms of standard modal logic, in particular that `says` is closed under consequence, together with the necessitation rule. In addition, the axioms that we consider include the following:

1. The hand-off axiom, as described above, and a generalization: if p says that s_1 implies p says s_2, then s_1 does imply p says s_2. In the special case where s_1 is q says s_2, we obtain a hand-off from p to q for s_2.
2. A further axiom that if p can make itself speak for q, then p speaks for q in the first place. This axiom may be seen roughly as a dual to the hand-off axiom.
3. The axiom that s implies p `says` s (Unit). As indicated in Section 3, this axiom is similar to the necessitation rule but stronger. It is also suggested by the computational lambda calculus.
4. The other main axiom from the computational lambda calculus (Bind): if s_1 implies p says s_2, then p says s_1 implies p says s_2.
5. The axiom that if p `says` s then s or p `says false`. We call this axiom Escalation, because it means that whenever p `says` s, either s is true or p says anything—possibly statements intuitively "much falser" than s.
6. An axiom suggested by the Principle of Least Privilege, roughly that if a principal is trusted on a statement then it is also trusted on weaker statements.

In short, we obtain the following relations between these axioms:

– In classical logics, the addition of axioms beyond the basic ones from modal logic quickly leads to strong and surprising properties that may not be desired. Bind is equivalent to Escalation, while Unit implies Escalation.
  Pictorially, we have:

$$\begin{tikzcd}
\text{Unit} \arrow[d] \\
\text{Escalation} \arrow[d, leftrightarrow] \\
\text{Bind}
\end{tikzcd}$$

  Some milder, reasonable additions do not lead to Escalation. For instance, we may require the standard axiom C4 from modal logic (if p says p says s then p says s) without obtaining Escalation. Unfortunately, those additions do not always suffice in applications.
– In intuitionistic logics, we have a little more freedom. In particular, CDD (which includes Unit and Bind) does not lead to Escalation.
  Pictorially, we have:

$$\begin{tikzcd}
\text{CDD} \arrow[d] \arrow[dr] & & \text{Escalation} \arrow[dl] \\
\text{Unit} & \text{Bind} &
\end{tikzcd}$$

  Many further refinements become possible, in particular because Escalation and Unit are independent intuitionistically.
– The general form of the hand-off axiom (1) is equivalent to Bind.
– Unit implies axiom (2). This axiom is equivalent to Unit if there is a truth-telling principal.
– Finally, Escalation implies axiom (6). Conversely, this axiom and C4 imply Escalation.

Next we elaborate on these relations between the axioms. Sections 6.1 and 6.2 focus on CDD, considering axioms (1), (2), (3), and (4). Section 6.3 focuses on Escalation (axiom (5)). Section 6.4 considers axiom (6).

6.1 C4

As noted in Section 4.3, CDD amounts to adding the axioms Unit and Bind to the basic intuitionistic logic of Section 4.2. We can replace Bind with the simpler C4 when we have Unit. Formally, C4 is:

$$[\textit{C4}] \quad \forall X.\ (p\ \texttt{says}\ p\ \texttt{says}\ X \rightarrow p\ \texttt{says}\ X)$$

We obtain:

Proposition 1. *Starting from the basic logic (without Excluded-middle), we have:*

1. *Bind implies C4;*
2. *Unit and C4 (together) imply Bind;*
3. *C4 does not imply Bind;*
4. *Unit does not imply C4 (and a fortiori not Bind).*

Assuming Excluded-middle, we have:

1. *C4 implies neither Bind nor Unit.;*
2. *Unit implies C4 (and therefore Bind);*
3. *Bind does not imply Unit.*

6.2 Hand-off

The hand-off axiom is:

$$[\textit{Hand-off}] \quad p\ \texttt{controls}\ (q \Rightarrow p)$$

In the basic logic, this axiom is not a theorem. Therefore, some examples and applications that require hand-off do not immediately work. For instance, in the example of Section 2, `deleteFile1` does not follow from `Bob` `says` (`Alice` ⇒ `Bob`) and `Alice` `says` `deleteFile1`. The axiom or ad hoc assumptions need to be added for those examples and applications (as in [43]). This need may be regarded as a shortcoming of the basic logic.

On the other hand, in CDD we obtain the hand-off axiom as a theorem. A slight generalization of the hand-off axiom is also interesting and also a theorem:

$$[\textit{Generalized-hand-off}] \quad \forall X, Y.\ p\ \texttt{controls}\ (X \rightarrow p\ \texttt{says}\ Y)$$

Theorem 2. *Starting from the basic logic, Bind is equivalent to Generalized-hand-off.*

Suppose that a principal p is trusted on whether it speaks for another principal q on every statement. In CDD, it follows that p must speak for q in the first place, whether it says so or not. If p does not wish to speak for q, it should reduce its authority, for instance by adopting an appropriate role [43, Section 6.1]. This result might be seen as a reassuring characterization of who can attribute the right to speak for q; it may also be seen as a dual or a limitation of hand-off in the context of CDD.

More precisely, we define:

$$[\textit{Authority-shortcut}] \quad (\forall X.\ p\ \texttt{controls}\ (p\ \texttt{says}\ X \rightarrow q\ \texttt{says}\ X)) \rightarrow (p \Rightarrow q)$$

We obtain:

Theorem 3. *Starting from the basic logic, Unit implies Authority-shortcut.*

Next we show the proof of this result. Suppose that, for all X,

$$p\ \texttt{controls}\ (p\ \texttt{says}\ X \to q\ \texttt{says}\ X)$$

and suppose that, for some particular X, we have $p\ \texttt{says}\ X$. (The following argument is peculiar in that the assumption $p\ \texttt{says}\ X$ is exploited twice in different ways.) In order to establish that $p \Rightarrow q$, we wish to derive $q\ \texttt{says}\ X$. Because $p\ \texttt{says}\ X$, Unit implies $p\ \texttt{says}\ q\ \texttt{says}\ X$. (Here we apply Unit under $\texttt{says}$.) Then by closure under consequence we have $p\ \texttt{says}\ (p\ \texttt{says}\ X \to q\ \texttt{says}\ X)$. By our assumption that, for all X, $p\ \texttt{controls}\ (p\ \texttt{says}\ X \to q\ \texttt{says}\ X)$, we obtain $p\ \texttt{says}\ X \to q\ \texttt{says}\ X$. Combining $p\ \texttt{says}\ X \to q\ \texttt{says}\ X$ with $p\ \texttt{says}\ X$, we obtain $q\ \texttt{says}\ X$, as desired.

A small variant of the argument shows that Unit also implies:

$$\forall X.\ ((p\ \texttt{controls}\ (p\ \texttt{says}\ X \to q\ \texttt{says}\ X)) \to (p\ \texttt{says}\ X \to q\ \texttt{says}\ X))$$

In other words, writing $p \Rightarrow_X q$ for $p\ \texttt{says}\ X \to q\ \texttt{says}\ X$ [45], we have that Unit implies:

$$\forall X.\ ((p\ \texttt{controls}\ (p \Rightarrow_X q)) \to (p \Rightarrow_X q))$$

The converse of Theorem 3 is almost true. Suppose that there is a truth-telling principal p, that is, a principal for which

$$\forall X.\ (X \equiv (p\ \texttt{says}\ X))$$

Applying Authority-shortcut to this principal, we can derive $s \to (q\ \texttt{says}\ s)$ by propositional reasoning, for every q and s. In other words, given such a truth-teller, we obtain Unit.

Nevertheless, the converse of Theorem 3 is not quite true. All basic axioms plus rules, plus Authority-shortcut, hold when we interpret $p\ \texttt{says}\ s$ as true, for every p and s. Unit does not hold under this interpretation.

In addition, Authority-shortcut does not follow from other axioms (such as Bind), even in classical logic. In other words, Authority-shortcut appears to be very close to Unit, and, if one wishes, it can be avoided by dropping Unit.

6.3 Escalation

As indicated above, Escalation is the following axiom:

$$[\textit{Escalation}] \quad \forall X, Y.\ ((p\ \texttt{says}\ X) \to (X \vee (p\ \texttt{says}\ Y)))$$

Equivalently, Escalation can be formulated as:

$$\forall X.\ ((p\ \texttt{says}\ X) \to (X \vee (p\ \texttt{says}\ \texttt{false})))$$

Escalation embodies a rather degenerate interpretation of $\texttt{says}$. At the very least, great care is required when Escalation is assumed. For instance, suppose that two principals p

and q are trusted on s, and that we express this as $(p\ \texttt{controls}\ s) \wedge (q\ \texttt{controls}\ s)$; with Escalation, if $p\ \texttt{says}\ q\ \texttt{says}\ s$ then s follows. Formally, we can derive:

$$(p\ \texttt{controls}\ s) \wedge (q\ \texttt{controls}\ s) \rightarrow ((p\ \texttt{says}\ q\ \texttt{says}\ s) \rightarrow s)$$

This theorem may be surprising. Its effects may however be avoided: p should not say that q says s unless p wishes to say s. As a result, though, the logic loses flexibility and expressiveness.

On the whole, we consider that Escalation is not a desirable property. Unfortunately, it can follow from the combination of properties that may appear desirable in isolation, as we show.

Theorem 4. *Starting from the basic logic (without Excluded-middle),*

1. *Unit and Bind (together) do not imply Escalation (in other words, Escalation is not a theorem of CDD);*
2. *Escalation implies Bind (and therefore C4).*

Assuming Excluded-middle, we have:

1. *Unit implies Escalation (and therefore Bind);*
2. *Escalation (and a fortiori Bind) does not imply Unit;*
3. *Bind implies Escalation;*
4. *C4 does not imply Escalation.*

Going further, in classical logic Unit implies that each principal p is either a perfect truth-teller or says $\texttt{false}$. In the former case, p speaks for any other principal; in the latter case, any other principal speaks for p. Formally, we can derive:

$$(p \Rightarrow q) \vee (q \Rightarrow p)$$

While this conclusion does not represent a logical contradiction, it severely limits the flexibility and expressiveness of the logic: policies can describe only black-and-white situations. This point is a further illustration of the fact that usefulness degrades even before a logic becomes inconsistent.

6.4 Monotonicity of Controls

The monotonicity of controls means that, if a principal controls a formula X, then it controls every weaker formula Y. Formally, we write:

$$[\textit{Control-monotonicity}] \quad \forall X, Y.\ \begin{pmatrix} (X \rightarrow Y) \\ \rightarrow \\ ((p\ \texttt{controls}\ X) \rightarrow (p\ \texttt{controls}\ Y)) \end{pmatrix}$$

This monotonicity property may seem attractive. In particular, it may make it easier to comply with the Principle of Least Privilege. This principle says [57]:

> Every program and every user of the system should operate using the least set of privileges necessary to complete the job.

The monotonicity of controls implies that, if p wants to convince a reference monitor of Y, and it can convince it of a stronger property X, then p should be able to state Y directly, rather than the stronger property X. For instance, suppose that Y is the statement that q may access a file f_1, and that X is the statement that q may access both f_1 and another file f_2. When p wishes to allow q to access f_1, it should not have to state also that q may access f_2. The monotonicity of controls allows p to say only that q may access f_1.

Nevertheless, the monotonicity of controls has questionable consequences. Starting from the basic logic (without Excluded-middle), Control-monotonicity implies:

$$(p\ \texttt{controls}\ s_1) \rightarrow (p\ \mathsf{says}\ s_2) \rightarrow (s_1 \vee s_2)$$

Here, the formulas s_1 and s_2 may be completely unrelated. For instance, suppose that p controls whether q may access a file f_1, and p says that q may access another file f_2; curiously, we obtain that q may access f_1 or q may access f_2.

In fact, the monotonicity of controls is equivalent to Escalation in the presence of C4. (Intuitionistically, C4 is strictly required for this equivalence.)

Theorem 5. *Starting from the basic logic (without Excluded-middle), the following are equivalent:*

- *Escalation,*
- *C4 and Control-monotonicity.*

However, neither Control-monotonicity nor C4 implies the other, not even in combination with Unit.

Assuming Excluded-middle, Control-monotonicity is equivalent to Escalation.

The theorems of this section should not be construed as a criticism of the Principle of Least Privilege. Formulations weaker than Control-monotonicity might be viable and less problematic.

7 Languages

The logics described above have had several applications. In particular, a number of research projects have relied on these logics for designing or explaining various languages and systems, such as virtual machines and operating systems (e.g., [7–9, 11, 12, 19, 20, 25, 41, 43, 46, 58, 60]). They have also influenced the languages that are the subject of this section.

7.1 From Logics to Languages

Languages for access control aim to support the practical expression and the enforcement of policies (e.g., [13, 16, 17, 22, 23, 33, 42, 48, 50, 54, 55, 61, 62]). The languages are general and flexible enough for programming a wide range of policies—for example, in file systems and for digital rights management.

Many of these languages are targeted at distributed systems in which cryptography figures prominently. They serve for expressing the assertions contained in cryptographic credentials, such as the association of a principal with a public key, the membership of a principal in a group, or the right of a principal to perform a certain operation at a specified time. They also serve for combining credentials from many sources with policies, and thus for making authorization decisions. More broadly, the languages sometimes aim to support trust management.

Several of these languages rely on concepts and techniques from logic, specifically from logic programming: D1LP [48], SD3 [42], RT [50], Binder [22], Soutei [54],SecPAL [13], and DKAL [33]. Other languages such as SDSI [55], SPKI [23], and XrML 2.0 [62] include related ideas though typically with less generality. Some of these have been influenced by logical work, but they have not been designed or presented as logical systems. We may however view them as logics, at least in a rudimentary sense. They all define systems of notations for describing principals, their statements, authorizations, and sometimes more. The notations come with rules for combining facts and deriving their consequences—for instance, rules for chaining certificates in public-key infrastructures.

Despite substantial progress, one might still question whether the deployment of these sophisticated languages would reduce the number of ways in which access control can be broken or circumvented. Policies in these languages might be difficult to write and to understand—but probably no worse than policies embodied in Perl scripts and the like, which are often the alternative.

7.2 A Look at Binder

Binder is a good representative for this line of work. It shares many of the goals of other languages and several of their features. It has a clean design, based directly on that of logic-programming languages.

Basically, a Binder program is a set of Prolog-style logical rules. Unlike Prolog, Binder does not allow function symbols; in this respect, Binder is close to the Prolog fragment Datalog. Also unlike Prolog, Binder has a notion of context and a distinguished operator `says`. For instance, in Binder we can write:

```
may-access(p,o,Rd) :- good(p)
may-access(p,o,Rd) :- Bob says may-access(p,o,Rd)
```

These clauses can be read as expressing that any principal `p` may access any object `o` in read mode (`Rd`) if `p` is good or if `Bob` says that `p` may do so.

Here only `:-` and `says` have built-in meanings. The other constructs (even the predicate symbol `may-access`) have to be defined or axiomatized. As in Prolog, `:-` stands for reverse implication ("if"). For instance,

```
may-access(Alice,Foo.txt,Rd)
```

would follow from these clauses and from

```
Bob says may-access(Alice,Foo.txt,Rd)
```

As in previous logical treatments of access control, `says` serves for representing the statements of principals and their consequences. Thus,

```
Bob says may-access(Alice,Foo.txt,Rd)
```

holds if there is a statement from `Bob` that contains a representation of the formula

```
may-access(Alice,Foo.txt,Rd)
```

or it can be derived if there is a statement from `Bob` that contains a representation of the formula

```
may-access(Alice,Foo.txt,Wr)
```

and another one that contains a representation of the clause

```
may-access(p,o,Rd) :- may-access(p,o,Wr)
```

Each formula is relative to a context (a source of statements). In our example, `Bob` is a context. Another context is implicit: the local context in which the formulas apply. For example,

```
may-access(p,o,Rd) :- Bob says may-access(p,o,Rd)
```

is to be interpreted in the implicit local context, and `Bob` is the name for another context from which the local context may import statements.

In addition to logic-programming rules, Binder includes a special proof rule for importing clauses `a :- a1, ..., an` from one context into another. The rule applies only to clauses where the atom `a` in the head is not of the form `q says s`. When importing a clause from context `p`, the rule replaces `a` with `p says a`, and replaces `ai` with `p says ai` if `ai` is not of the form `q says s`, for `i` = `1`..`n`. For example, when `Charlie` exports the clauses:

```
may-access(p,o,Rd) :- good(p)
may-access(p,o,Rd) :- Bob says may-access(p,o,Rd)
```

the local context obtains:

```
Charlie says may-access(p,o,Rd) :- Charlie says good(p)
Charlie says may-access(p,o,Rd) :-
   Bob says may-access(p,o,Rd)
```

This proof rule is complicated enough to call for some logical analysis. It can be partly justified by standard modal logic, in particular via the theorem

$$((p \;\mathtt{says}\; s) \wedge (p \;\mathtt{says}\; s')) \rightarrow p \;\mathtt{says}\; (s \wedge s')$$

and the axiom of closure under consequence. However, more is needed, even for our example. The proof rule can be derived once we add the strong axiom Unit. In fact, a restricted form of Unit suffices:

$$(q \;\mathtt{says}\; s) \rightarrow (p \;\mathtt{says}\; q \;\mathtt{says}\; s)$$

Garg has recently studied the logical foundations of Binder and Soutei, considering this restricted form of Unit [26, 27].

Binder does not assume or require that predicate symbols mean the same in every context. For example, `Bob` might not even know about `may-access`, and might assert `lecteur(Alice,Foo.txt)` instead of `may-access(Alice,Foo.txt,Rd)`. In that situation, one may translate explicitly with the clause:

```
may-access(p,o,Rd) :- Bob says lecteur(p,o)
```

On the other hand, Binder does not provide much built-in support for local name spaces. A closer look reveals that the names of contexts have global meanings. In particular, when `Charlie` exports:

```
may-access(p,o,Rd) :- Bob says lecteur(p,o)
```

the local context obtains:

```
Charlie says may-access(p,o,Rd) :-
   Bob says lecteur(p,o)
```

without any provision for the possibility that `Bob` might not be the same locally and for `Charlie`.

Other systems, such as SDSI and SPKI [23, 55], support more elaborate naming mechanisms, with corresponding logical explanations and problems (e.g., [1, 32, 34, 35, 37, 47]). They enable the linking of name spaces, allowing for the possibility that the intended meaning of a name might not be the same in all contexts.

8 Outlook

Logics for access control have been used in a variety of ways, as indicated in this paper. They have played a helpful role in the design and understanding of languages and systems. Nevertheless, they have not replaced traditional mechanisms for access control, nor is there any prospect that they will in the near future. Although logics can be powerful and general, for each specific application there typically exist special-purpose, expedient alternatives. It is questionable whether the proliferation of those alternatives contributes to security. Still, whether logics should be routinely employed for actual access control (at “compile-time” or “run-time”, rather than “design-time” or “understanding-time”), and how they should be employed, remains open to debate.

A great deal of caution should be applied in selecting logics for access control, considering both their formal properties and their security implications. In particular, while in a classical setting we may want to stay close to basic modal logic, in an intuitionistic setting we may adopt CDD. This move may be attractive, in particular, because CDD supports the hand-off of authority. Nevertheless, other logics (for instance, with weaker axioms, or with additional operators) may be reasonable choices as well. We do not argue that the use of a particular set of axioms is required for writing good security policies. It is possible that reasonable security policies and other assertions can be formulated in many different systems, with different underlying logics. However, understanding the properties and consequences of these logics is essential for writing appropriate formulas reliably.

Acknowledgments I am grateful to Andrew Appel, Mike Burrows, William Cook, John DeTreville, Cédric Fournet, Deepak Garg, Butler Lampson, Dave Langworthy, Greg Morrisett, Roger Needham, and Frank Pfenning for discussions and collaborations that contributed to this paper or to its origins.

References

1. Martín Abadi. On SDSI's linked local name spaces. *Journal of Computer Security*, 6(1-2):3–21, 1998.
2. Martín Abadi. Logic in access control. In *Proceedings of the Eighteenth Annual IEEE Symposium on Logic in Computer Science*, pages 228–233, 2003.
3. Martín Abadi. Access control in a core calculus of dependency. *Electronic Notes in Theoretical Computer Science*, 172:5–31, April 2007. *Computation, Meaning, and Logic: Articles dedicated to Gordon Plotkin*.
4. Martín Abadi. Variations in access control logic. In Ron van der Meyden and Leendert van der Torre, editors, *Deontic Logic in Computer Science, 9th International Conference, DEON 2008*, volume 5076 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 2008.
5. Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. A core calculus of dependency. In *Proceedings of the 26th ACM Symposium on Principles of Programming Languages*, pages 147–160, 1999.
6. Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, October 1993.
7. Martín Abadi and Ted Wobber. A logical account of NGSCB. In *Formal Techniques for Networked and Distributed Systems – FORTE 2004*, pages 1–12. Springer, 2004.
8. Andrew W. Appel and Edward W. Felten. Proof-carrying authentication. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, pages 52–62, 1999.
9. Lujo Bauer. *Access Control for the Web via Proof-Carrying Authorization*. PhD thesis, Princeton University, November 2003.
10. Lujo Bauer, Lorrie Cranor, Robert W. Reeder, Michael K. Reiter, and Kami Vaniea. A user study of policy creation in a flexible access-control system. In *CHI 2008: Conference on Human Factors in Computing Systems*, pages 543–552, 2008.
11. Lujo Bauer, Scott Garriss, Jonathan M. McCune, Michael K. Reiter, Jason Rouse, and Peter Rutenbar. Device-enabled authorization in the Grey system. In *Proceedings of the 8th Information Security Conference (ISC '05)*, pages 431–445, 2005.
12. Lujo Bauer, Scott Garriss, and Michael K. Reiter. Distributed proving in access-control systems. In *Proceedings of the 2005 Symposium on Security and Privacy*, pages 81–95, 2005.
13. Moritz Y. Becker, Cédric Fournet, and Andrew D. Gordon. Design and semantics of a decentralized authorization language. In *Proceedings of the 20th IEEE Computer Security Foundations Symposium*, pages 3–15, 2007.
14. P. N. Benton, G. M. Bierman, and V. C. V. de Paiva. Computational types from a logical perspective. *Journal of Functional Programming*, 8(2):177–193, 1998.
15. Elisa Bertino, Barbara Catania, Elena Ferrari, and Paolo Perlasca. A logical framework for reasoning about access control models. *ACM Transactions on Information and System Security*, 6(1):71–127, February 2003.
16. Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The KeyNote trust-management system, version 2. IETF RFC 2704, September 1999.

17. Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings 1996 IEEE Symposium on Security and Privacy*, pages 164–173, 1996.
18. Luca Cardelli. Type systems. In Allen B. Tucker, editor, *The Computer Science and Engineering Handbook*, chapter 103, pages 2208–2236. CRC Press, Boca Raton, FL, 1997.
19. J. G. Cederquist, R. Corin, M. A. C. Dekker, S. Etalle, J. I. den Hartog, and G. Lenzini. Audit-based compliance control. *Int. J. Inf. Secur.*, 6(2):133–151, 2007.
20. Andrew Cirillo, Radha Jagadeesan, Corin Pitcher, and James Riely. Do as I SaY! programmatic access control with explicit identities. In *Proceedings of the 20th IEEE Computer Security Foundations Symposium*, pages 16–30, 2007.
21. Jason Crampton, George Loizou, and Greg O'Shea. A logic of access control. *The Computer Journal*, 44(2):137–149, 2001.
22. John DeTreville. Binder, a logic-based security language. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 105–113, 2002.
23. Carl Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylönen. SPKI certificate theory. IETF RFC 2693, September 1999.
24. M. Fairtlough and M. V. Mendler. Propositional lax logic. *Information and Computation*, 137(1):1–33, August 1997.
25. Cédric Fournet, Andrew D. Gordon, and Sergio Maffeis. A type discipline for authorization in distributed systems. In *Proceedings of the 20th IEEE Computer Security Foundations Symposium*, pages 31–45, 2007.
26. Deepak Garg. Principal-centric reasoning in constructive authorization logic. Technical Report CMU-CS-09-120, Computer Science Department, Carnegie Mellon University, April 2009.
27. Deepak Garg. Proof search in an authorization logic. Technical Report CMU-CS-09-121, Computer Science Department, Carnegie Mellon University, April 2009.
28. Deepak Garg and Martín Abadi. A modal deconstruction of access control logics. In Roberto M. Amadio, editor, *Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS 2008*, volume 4962 of *Lecture Notes in Computer Science*, pages 216–230. Springer, 2008.
29. Deepak Garg and Frank Pfenning. Non-interference in constructive authorization logic. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop*, pages 283–296, 2006.
30. Morrie Gasser, Andy Goldstein, Charlie Kaufman, and Butler Lampson. The Digital Distributed System Security Architecture. In *Proceedings of the 1989 National Computer Security Conference*, pages 305–319, 1989.
31. Jean-Yves Girard. *Interprétation Fonctionnelle et Elimination des Coupures de l'Arithmétique d'Ordre Supérieur*. Thèse de doctorat d'état, Université Paris VII, June 1972.
32. Adam J. Grove and Joseph Y. Halpern. Naming and identity in epistemic logics, I: The propositional case. *Journal of Logic and Computation*, 3(4):345–378, 1993.
33. Yuri Gurevich and Itay Neeman. DKAL: Distributed-knowledge authorization language. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium*, pages 149–162, 2008.
34. Joseph Y. Halpern and Ron van der Meyden. A logic for SDSI's linked local name spaces. *Journal of Computer Security*, 9(1-2):47–74, 2001.
35. Joseph Y. Halpern and Ron van der Meyden. A logical reconstruction of SPKI. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, pages 59–72, 2001.
36. Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, August 1976.
37. Jon Howell and David Kotz. A formal semantics for SPKI. In *Proceedings of the Sixth European Symposium on Research in Computer Security*, volume 1895 of *Lecture Notes in Computer Science*, pages 140–158. Springer, 2000.

38. G. E. Hughes and M. J. Cresswell. *An Introduction to Modal Logic*. Methuen Inc., New York, 1968.
39. Sushil Jajodia, Pierangela Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 31–42, 1997.
40. Sushil Jajodia, Pierangela Samarati, V. S. Subrahmanian, and Elisa Bertino. A unified framework for enforcing multiple access control policies. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, volume 26(2) of *SIGMOD Record*, pages 474–485, 1997.
41. Limin Jia, Jeffrey A. Vaughan, Karl Mazurak, Jianzhou Zhao, Luke Zarko, Joseph Schorr, , and Steve Zdancewic. Aura: A programming language for authorization and audit. In *Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming*, pages 27–38, 2008.
42. Trevor Jim. SD3: A trust management system with certified evaluation. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 106–115, 2001.
43. Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, November 1992.
44. Butler W. Lampson. Protection. In *Proceedings of the 5th Princeton Conference on Information Sciences and Systems*, pages 437–443, 1971.
45. Butler W. Lampson. Computer security in the real world. *IEEE Computer*, 37(6):37–46, June 2004.
46. Christopher Lesniewski-Laas, Bryan Ford, Jacob Strauss, M. Frans Kaashoek, and Robert Morris. Alpaca: extensible authorization for distributed services. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 432–444, 2007.
47. Ninghui Li. Local names in SPKI/SDSI. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 2–15, July 2000.
48. Ninghui Li, Benjamin N. Grosof, and Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information and System Security*, 6(1):128–171, February 2003.
49. Ninghui Li and John C. Mitchell. Datalog with constraints: A foundation for trust-management languages. In *Proceedings of the Fifth International Symposium on Practical Aspects of Declarative Languages (PADL 2003)*, volume 2562 of *Lecture Notes in Computer Science*, pages 58–73. Springer, 2003.
50. Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust-management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130, 2002.
51. Ninghui Li, John C. Mitchell, and William H. Winsborough. Beyond proof-of-compliance: security analysis in trust management. *J. ACM*, 52(3):474–514, 2005.
52. Eugenio Moggi. Notions of computation and monads. *Information and Control*, 93(1):55–92, 1991.
53. George C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM Symposium on Principles of Programming Languages*, pages 106–119, 1997.
54. Andrew Pimlott and Oleg Kiselyov. Soutei, a logic based trust-management system. In Masami Hagiya and Philip Wadler, editors, *Proceedings of the Eighth International Symposium on Functional and Logic Programming*, volume 3945 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2006.
55. Ronald L. Rivest and Butler Lampson. SDSI — A Simple Distributed Security Infrastructure. On the Web at `http://theory.lcs.mit.edu/~cis/sdsi.html`, 1996.

56. John Rushby. Design and verification of secure systems. *Proceedings of the 8th ACM Symposium on Operating System Principles, ACM Operating Systems Review*, 15(5):12–21, December 1981.
57. Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer system. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.
58. Jeffrey A. Vaughan, Limin Jia, Karl Mazurak, and Steve Zdancewic. Evidence-based audit. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium*, pages 177–191, 2008.
59. Nathan Whitehead. A certified distributed security logic for authorizing code. In Thorsten Altenkirch and Conor McBride, editors, *Types for Proofs and Programs, International Workshop, TYPES 2006, Revised Selected Papers*, volume 4502 of *Lecture Notes in Computer Science*, pages 253–268, 2007.
60. Edward Wobber, Martín Abadi, Michael Burrows, and Butler Lampson. Authentication in the Taos operating system. *ACM Transactions on Computer Systems*, 12(1):3–32, February 1994.
61. eXtensible Access Control Markup Language (XACML) version 1.0. OASIS Standard, at `http://www.oasis-open.org/committees/xacml/repository/`, 2003.
62. eXtensible Rights Markup Language (XrML) version 2.0. At `http://www.xrml.org/`.
63. Wenchao Zhou, Yun Mao, Boon Thau Loo, and Martín Abadi. Unified declarative platform for secure networked information systems. In *Proceedings of the 25th International Conference on Data Engineering*, pages 150–161, 2009.