A Logic for Reasoning About Security

JANICE GLASGOW and GLENN MACEWEN
Queen's University, Kingston
and
PRAKASH PANANGADEN
McGill University, Montreal

A formal framework called *Security Logic* (*SL*) is developed for specifying and reasoning about security policies and for verifying that system designs adhere to such policies. Included in this modal logic framework are definitions of *knowledge*, *permission*, and *obligation*. Permission is used to specify secrecy policies and obligation to specify integrity policies. The combination of policies is addressed and examples based on policies from the current literature are given.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic; H.2.0 [**Database Management**]: General; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Security, Theory

Additional Key Words and Phrases: Composition knowledge, integrity, logic, obligation, permission, policy, possible-worlds, secrecy, security, time

1. INTRODUCTION

An intuitive way to define nondisclosure security (secrecy) and some aspects of integrity is in terms of what information a subject[1] *knows*, what information a subject is *obligated to know*, and what information a subject has *permission to know*. In reasoning about security, it is important that one have a precise notion of the meaning of knowledge, obligation, and permission. Furthermore, it is important to know how knowledge, obligation, and permission interact and propagate from simple statements to complex ones. In this paper we present a logical approach to these questions based on a modal logic formalism. The advantage of this approach is that there already exists a well-understood theory of how modalities interact with ordinary logical connectives. By casting knowledge, obligation, and permission as modalities, we

[1] A subject is an abstraction of any entity that can possess information. The most common subjects are human users and computer processes.

This work is supported by the Natural Sciences and Engineering Research Council of Canada and the Information Technology Research Center of Ontario.

can obtain a structured framework for reasoning about security. This framework provides a language for expressing abstract policy definitions for security without considering the specific mechanisms for implementing such policies.

Reasoning about knowledge has long been an issue in philosophy. More recently it has been argued that reasoning about knowledge is important for understanding distributed system protocols [24]. As in the previous work, we define knowledge using a *possible-worlds* model. The intuition ascribed by this model is that a subject in a particular world can conceive a number of other possible worlds. The subject is said to *know* a fact only if that fact is true in all worlds that the subject considers possible.

In this paper we modify the model for reasoning about knowledge to one that also captures the notion of *inevitable knowability*. When expressing integrity properties, it may be necessary to guarantee that a subject will eventually have some knowledge, regardless of what computations may occur. For this reason, we combine our logic for knowledge with a temporal logic. We also extend the model to include operators for permission and obligation. The resulting theory, called *Security Logic* (*SL*), can be used to specify a security policy by defining how time, knowledge, obligation, and permission interact. Two fundamental properties of security are expressed using SL: (i) that a subject knows only facts that it has permission to know; and (ii) that if a subject is obligated to know something, then eventually the subject will know it.

A current research problem is to find good definitions of secrecy and integrity that adequately capture a useful notion of security, address threats such as covert channels and computer viruses, and are composable in the sense that connecting secure components in a secure manner always results in a secure system. We intend our logic to provide a framework for expressing such definitions, called *security policies*. A related research problem is to find efficient and usable methods for gaining assurance that a system conforms to a policy. To a large degree this means performing a formal verification that a system design conforms to a policy and that an implementation conforms to the design. An interpretation in our logic defines the value of truth assignments for the permission and obligation operators in the language. A policy is given as a set of *policy constraints* on interpretations. By defining the policy in the logic, we can also reason about whether or not it adheres to the fundamental security properties, which are also expressed in the theory.

Section 2 of the paper motivates our research by discussing how abstract concepts of security relate to knowledge, permission, and obligation. Section 3 reviews the standard possible-worlds models for knowledge and time and shows how these two modalities interact. In Section 4 we extend the possible-worlds model for time and knowledge to include operators for permission and obligation. The interpretations of permission and obligation depend on a particular security policy. Section 5 describes how security policies can be defined and combined using Security Logic. In Section 6 we illustrate this by defining an information flow instance of multilevel secrecy, an integrity policy, two access control instances of multilevel secrecy, and an

access-control-based policy embodying both secrecy and integrity. We conclude with a discussion of related work and propose future research.

2. LOGIC AND SECURITY

Computer security comprises a variety of general concerns including secrecy, integrity, and availability. The best understood of these concerns is *secrecy*, which means that certain information is to be kept from certain subjects of the computer system [11]. That is, only certain subjects have *permission* to *know* certain information. *Integrity*, a somewhat less well-understood concern, generally means that certain information is to be protected from improper processing by some subjects so that it retains its properties of usefulness and accessibility to other subjects [40]. One can view this as a requirement that certain subjects are *obligated* to know certain information; a consequence is that the information must be protected and transmitted to those subjects. *Availability*, well understood as an intuitive idea but difficult to enforce, means that information is to be accessible when it is needed, that is, while it is still useful [7]. For each of these general concerns, one finds in the literature and in practice that there are a variety of views, some precise and some less precise, as to what exactly the problem to be addressed is [15, 16, 41]. In other words, security can be precisely defined only relative to a particular environment and set of threats.

As a result of this variety of views, there are a growing number of security policies and associated formal models [5, 8, 12, 13, 22, 27, 30, 33, 34, 36, 39, 46]. The possibility that there might be several policies in use within a single environment raises a serious question: How might policies that were developed independently to address different security concerns interact when used in concert? Are there subtle interactions that might lead to security failures? The same concern arises if two systems, each of which enforces a different security policy, are connected. This problem leads us to be concerned with distributed system implementations.

Modal logic is a natural vehicle for reasoning about security. Logic allows us to reason about *knowledge*, a fundamental concept of computer security, since much security is based on the concept of what a subject *knows*. For example, an intuitive way to describe secrecy is in terms of what information a subject has *permission to know*. It seems clear that for any security policy, a subject should not know any information that it does not have permission to know. A particular security policy then must define the meaning of the term "permission." For example, in a multilevel secure system based on a lattice of security levels, permissions must be assigned so that a subject is permitted to know only information that is permitted to subjects at the same or a lower security level [9].

Another example of security specification involves integrity. There has been a large amount of activity recently to examine various notions of integrity and how to specify them so that designs and implementations can be verified. Integrity is a harder concept to capture than secrecy, partly because it seems to be application dependent. Application-independent exam-

ples of secrecy specification are easy to find, e.g., the multilevel secure system above. However, examples of integrity typically involve particular applications. For example, a company's double-entry ledger books require high integrity. This means that only appending updates are allowed, and only then by certain trusted subjects (the accounting staff). This also means that they must be self-consistent (debits equal credits). A consequence is that when examined by certain subjects that need accurate information (the auditors), they provide that accurate information. There is no notion of secrecy here; the books may be public. It is purely a question of protecting the accuracy of, and providing access to, the stored information.

We believe that much of what is involved in integrity can be captured by a notion of *obligation to know* certain information. To explain this in terms of the ledger example, let us identify the subjects as processes or users; we have an input user (an accountant), a ledger process (incorporating the ledger storage), and an output user (an auditor). The input user knows certain accounting transactions. To meet our integrity requirement, the ledger process is obligated to know that information. And the output user is obligated to know the information in the ledger process. We will see that our notion of integrity allows us to express the facts that the ledger will contain (will eventually know) information that it is obligated to know, and that the output user will eventually know all information in the ledger. (An additional property of secrecy could also be used to ensure that there can be no other "contaminating" information.) Notice that the latter fact can be enforced partly by allowing appending updates only, thus avoiding a problem resulting from concurrent access. Notice also that the auditor may have permission to know all information about a company, but is particularly obligated to know such things as the contents of the ledger.

Integrity seems inherently to involve time. To begin with, we have identified two kinds of integrity, weak and strong. Weak integrity means that a subject can have a way of knowing information that it is obligated to know. Strong integrity means that a subject must eventually know information that it is obligated to know. The latter is a type of liveness property, and is harder to verify in an implementation. However, temporal operators from modal logic can be useful in expressing and reasoning about strong integrity.

The third type of security, availability, also involves time. It is not our intention to address this concern here, but only to point out that the temporal operators included in the logic for integrity specification may prove useful in expressing availability properties.

Finally, one of our primary objectives is to address the problem of composite policies. To do this we need a common framework in which to express policies. Modal logic provides this framework.

Previous work on a method for reasoning about knowledge in multilevel secure systems was introduced in [13], where operator nets were used to specify a variety of security properties such as secrecy, integrity, and authority systems. This work on knowledge and permission was extended to consider the notion of *obligation* found in modal logic as a fundamental notion in formally specifying integrity [17]. Integrity, as seen by Clark–Wilson-type

models, was discussed; it was argued that the essential part of their model, for the purpose of formal specification, is an expression of required connectivity in a graph representing integrity subjects. A formal semantic definition of integrity based on operator nets was given. This paper is an extension and consolidation of these previous papers. In particular, time is included, permission and obligation are related, and examples are given in which policies from the literature are expressed in the logic.

3. KNOWLEDGE AND TIME

Fundamental to any precise formulation of security is a formalization of *knowledge*. A primary concern of a security policy is to articulate the circumstances under which a subject acquires knowledge. In our analysis of security we use a modal logical formalism that defines knowledge in terms of indistinguishability of states. The basic idea is that a subject has a partial view of a global state. A subject "knows" something in a particular global state if that thing is true in all the states that are indistinguishable (by the subject) from the global state.

In order to define the theory for knowledge precisely, we use the well-understood *possible-worlds* semantics defined by Kripke [28]. In this approach, one defines a set of "worlds" (the so-called possible worlds) and a family of indistinguishability relations, one for each subject, that define which worlds a subject cannot tell apart. A subject is said to know a formula if the formula is true in all worlds considered possible by the subject. This view matches the computational situation in distributed systems very well [23], as the possible worlds correspond to the global states and the possibility relations are determined by the local states of processes.

In developing a formalism for reasoning about security, we are concerned with the relationship between knowledge and time or, more precisely, between knowledge and the execution of a system. This is particularly important when dealing with obligation. If a subject is obligated to know something, this does not necessarily imply that the subject currently has knowledge of it but that the subject can, or will, eventually know it. Thus our logic for reasoning about security includes temporal and epistemic modal operators.

Recent research in the area of reasoning about knowledge in distributed systems has focused on studying agreement protocols. These protocols are formalized via *common knowledge* [25]. Such knowledge is important in situations in which every process in a network must know a fact before an action can be taken. Although our definition of knowledge is based on a formalism similar to those of people working on common knowledge and knowledge-based protocols, in this paper we concentrate on the use of knowledge for reasoning about secure systems.

Temporal logic [43] is a well-understood formalism for reasoning about systems [42]. In defining a temporal theory, there are two contrasting interpretations regarding the nature of time: *linear* and *branching* [10, 29]. In the linear interpretation, one talks about the truth of formulas with respect to a

given computation sequence. The branching interpretation allows one to talk about a world and *all* of its future worlds. For the purpose of this paper we wish to reason about liveness properties of distributed systems, that is, properties that must hold in all possible future computation paths. Therefore we use the branching interpretation of time.

The properties of security can be broken down into those of liveness and safety. As discussed earlier, notions of secrecy relate to safety properties. That is, certain information flows must not occur. Another aspect of security, integrity, corresponds more to liveness properties—certain information flows can or must occur. Although neither temporal logic nor a logic of knowledge alone is expressive enough to capture the intuitions necessary for both secrecy and integrity, a combined theory of time and knowledge does. In the remainder of this section we define our theory in terms of previously defined standard theories of knowledge and temporal logic. In the next section we extend this theory to include modal operators for permission and obligation. This extended theory provides a formal basis for reasoning about security.

3.1 A Logic of Knowledge

In this section we review a standard theory for reasoning about knowledge [24]. This theory is an S5 modal logic, and has previously been used for reasoning about distributed system protocols (e.g., [25]).

The language for reasoning about knowledge consists of a set of subjects U labeled $1 \ldots n$; a set of primitive propositions Σ; and for all subjects i, a modal operator K_i. A proposition of the form $K_i\phi$ states that *subject i knows that proposition ϕ is true.*

The notion of possible worlds can be formalized using *Kripke structures* [28]. In a system with n subjects, the Kripke structure is a tuple $M = (S, \pi, \kappa_1, \ldots, \kappa_n)$ where

(1) S is the set of possible worlds.

(2) The relation κ_i for a subject i is an equivalence relation (reflexive, symmetric, and transitive) on S. We say that two worlds are *indistinguishable* to subject i if they belong to the same equivalence class in κ_i.

(3) A formula is defined to be true or false in a possible world. We write $s \models \phi$ to denote that formula ϕ is true in world s.

For each world $s \in S$ and each primitive proposition $\phi \in \Sigma$, π assigns a truth value to ϕ in s (i.e., $\pi(s, \phi) \in \{true, false\}$). We define the notion of a formula being true in a world via the $\models$ (read "models") relation as follows. For all $s \in S$, and all well-formed formulas[2] ϕ, ψ:

For all $\phi \in \Sigma, s \models \phi$ iff $\pi(s, \phi) = true$

$s \models \phi \wedge \psi$ iff $s \models \phi$ and $s \models \psi$

$s \models \neg \phi$ iff not $s \models \phi$

$s \models K_i\phi$ iff for all s' such that $(s, s') \in \kappa_i, s' \models \phi$

[2] The set of well-formed formulas is defined in Section 3.2 and later extended in Section 4.

A formula ϕ is said to be *valid* if it is true in all worlds $s \in S$. We denote this as $\models \phi$.

A sound and complete axiom scheme that characterizes the logic described above has previously been studied extensively by philosophers. These axioms for knowledge are the following:

Axiom K1. $K_i \phi \rightarrow \phi$.
Axiom K2. $K_i(\phi \rightarrow \psi) \rightarrow (K_i \phi \rightarrow K_i \psi)$.
Axiom K3. $K_i \phi \rightarrow K_i K_i \phi$.
Axiom K4. $\neg K_i \phi \rightarrow K_i(\neg K_i \phi)$.

Axiom K1 (the *knowledge* axiom) states that a subject cannot know anything that is false. This distinguishes knowledge from belief. Axiom K2 (the *consequence closure* axiom) states that a subject knows all things that can be deduced from its knowledge. The final two axioms (the *positive* and *negative introspection* axioms) state that a subject has introspective ability.

The theory includes the proof rules of propositional logic as well as a rule that states that if a formula is valid (true in all possible worlds), then a subject knows that formula:

Rule 1. If ϕ is valid then so is $K_i \phi$.

In the following subsections we expand the Kripke structure for knowledge to include an additional relation that allows us to define modal operators for time, permission, and obligation. Thus we can define models for security logic in terms of a single Kripke structure.

3.2 Knowledge and Time

The temporal logic for knowledge presented in this paper is based on a branching time structure. In particular, we use the logic *UB*: the *unified* system of *branching* time [2] to develop our theory. In the UB system, the underlying model consists of a branching tree of all possible paths of reachable possible worlds. Temporal operators are defined that correspond to quantification over these paths.

As well as the modal operators K_i for knowledge, we include three of the six temporal operators of UB. These are the operators: $\forall\Box$ (*always*), $\forall\Diamond$ (*eventually*), and $\exists\Diamond$ (*sometimes*). The first symbol in these operators denotes the quantification over paths, while $\Box$ and $\Diamond$ denote temporal quantification over the selected path. Given a world s, a path defined as a sequence of worlds, and a formula ϕ, we interpret the temporal operators applied to ϕ, informally, as follows:

$s \models \forall\Box\phi$ iff ϕ is true in all worlds along all paths initiated at s;
$s \models \forall\Diamond\phi$ iff ϕ is true for some world along all paths initiated at s;
$s \models \exists\Diamond\phi$ iff ϕ is true for some world along some path initiated at s.

We now define a general class of models that we wish to talk about. We are given a set S of possible "worlds"; henceforth we call them *states*. These are to be thought of as possible system states. We have a set $\mathscr{R}$ of finite or

infinite *sequences* of members of S called *runs*. If r is a typical member of $\mathscr{R}$, we write $r[i]$ for the ith member of the sequence r. There is a binary relation R on the possible states S derived from the set of runs $\mathscr{R}$. We say that for any two states s and s', $(s, s') \in R$ if and only if there is a run r such that for some positive integers i and j, $r[i] = s$ and $r[i + j] = s'$. The states in S are equipped with the indistinguishability relations discussed in the last subsection. Thus, one has the indistinguishability structure that allows one to model knowledge, and the reachability relation needed to model temporal notions.

This class of models is very general and flexible, and has been used profitably to model distributed systems. In a distributed system, one has a set of autonomous computing agents that communicate in some way. The system evolves in time and the processes exchange messages. The global state of the system encodes the information of the local states of all the processes. Each process is only aware of its local state, and thus there is a natural notion of indistinguishability for each process—namely a process i cannot distinguish two global states that have the same local state for i. The runs, of course, correspond to the executions of the system.

We now give a more formal presentation of the combined logic for time and knowledge. The alphabet of the language consists of

(1) a denumerable set Σ of primitive proposition letters $\phi, \psi, \ldots$;
(2) the logical symbols T (truth) and $\perp$ (falsehood), and the logical connectives $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction) and $\rightarrow$ (implication);
(3) the modal operators K_i (knowledge) for all subjects i; and
(4) the temporal operators $\forall\Box$ (always), $\forall\Diamond$ (eventually), and $\exists\Diamond$ (sometimes).

The set of well-formed formulas for the language is the smallest set W such that

(1) every proposition letter in Σ, T, and $\perp$ are in W;
(2) if ϕ and ψ are in W, then so are $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \rightarrow \psi$, and (ϕ);
(3) if ϕ is in W, then so are $K_i\phi$ (for all subjects i), $\forall\Box\phi$, $\forall\Diamond\phi$, and $\exists\Diamond\phi$.

We now describe how the modal operators are modeled. In other words, we give the semantics for the logic. In the following we assume that the quantification of r is over runs in the model and the quantification of s, s' and s'' is over possible states.

$s \models \forall\Box\phi$ iff for all $(s, s') \in R$, $s' \models \phi$;
$s \models \forall\Diamond\phi$ iff for all r, (if $r[i] = s$, then there exists s' and nonnegative integer j such that $s' = r[i + j]$ and $s' \models \phi$);
$s \models \exists\Diamond\phi$ iff for some $(s, s') \in R$, $s' \models \phi$.

The knowledge operators are modeled exactly as described before.

We now present a deductive system of axioms and inference rules for proving validity of formulas in the theory. We make no claims of completeness

here. The axioms and rules allow one to use the formulas to reason about systems, but this is really only of secondary importance; the important point is to be able to talk about systems in a precise way. The systems (models) are primary, and the logic allows us to be precise when talking about the systems of interest. We need only consider two temporal operators $\forall\Box$ and $\forall\Diamond$, since the following holds: $\exists\Diamond\phi \leftrightarrow \neg\forall\Box\neg\phi$. To the axioms and rules for the logic of knowledge, we add the following standard axioms for branching temporal logic:

Axiom T1. $\phi \rightarrow \forall\Diamond\phi$.

Axiom T2. $\forall\Box(\phi \rightarrow \psi) \rightarrow (\forall\Box\phi \rightarrow \forall\Box\psi)$.

Axiom T3. $\forall\Box\phi \rightarrow \forall\Box\forall\Box\phi$.

Although we only consider the temporal operators involving $\Diamond$ to express our logic for security, when considering a deduction system for the logic we may wish to include a more complete temporal logic. For example, the UB operator for *next* would allow us to reason by induction over possible states in a run. However, for the most part, we expect that one will want to reason concretely with the structures of interest, and there is little to be gained by formalizing all the reasoning principles one might want to use.

The rules of inference for the system include Rule 1 for knowledge as well as *modes ponens* and the following generalization rule:

Rule 2. If ϕ is valid then so is $\forall\Box\phi$.

In the next section we will see how the combined logic of knowledge and time can be used to express abstract security policies. For example, we will be able to state a strong integrity property such as: *If subject i is obligated to know a formula ϕ, then subject i will eventually know ϕ.*

Note that this statement not only implies that communication exists that enables subject i to know ϕ, but also that a liveness property exists; that is, the communication will in fact occur. This specification is achieved using the eventuality operator, $\forall\Diamond$. A weaker form of this property (weak integrity), which implies that communication is possible but not guaranteed, can also be expressed using the sometimes temporal operator, $\exists\Diamond$. Note that weak integrity is, in fact, very weak. By itself it does not guarantee anything; it is merely a prelude to proving strong properties.

4. PERMISSIONS AND OBLIGATIONS

In the previous section we presented a possible-worlds (henceforth termed possible-states) model for reasoning about knowledge and time. The third key ingredient needed to complete the discussion is a concept of permission, and its dual concept, obligation. The notion of a logic that includes the concepts of obligation and permission is not a new one. The language of *deontic logic* extends the language of propositional logic by introducing propositions of the form $O\phi$ to denote that ϕ is *obligatory* and $P\phi$ to denote that ϕ is *permitted.* Corresponding to modal logic, the O and P operators denote deontic necessity and permission. Deontic logic formalizes these ideas; though

the intended realm of discussion there is ethics and morality. We borrow deontic ideas but construe them much more narrowly. Permission is interpreted only in the context of "permission to know" a fact and similarly for obligation. One virtue of this is that we can now define permission and obligation as modalities, i.e. they apply to formulas, rather than in classical deontic logic, where they might apply to actions. Two areas in which deontic logics have previously been applied are ethics and legal theory. In these areas the logic has been criticized because of the linguistic interpretation ambiguities. These problems are not so severe in our case, as our language will restrict the scope of the deontic operators to include only knowledge formulas.

A subject may have *permission to know* or be *obligated to know* a particular formula. We have previously introduced theories to reason about knowledge [14], permission [20, 21], and obligation [17, 19] individually. It was argued that the notion of permission corresponds to secrecy (a kind of safety), properties and obligation corresponds to integrity (a kind of liveness) properties. In this section we define a combined theory of permission and obligation, which provides a logical foundation for reasoning about security. Since it is important to consider security policies that demonstrate both safety and liveness properties, it is essential to know how permissions and obligations interact.

A fundamental axiom of deontic logic is that obligation implies permission, i.e., $O\phi \rightarrow P\phi$. This axiom is a weaker form of the modal necessity axiom $\Box\, \phi \rightarrow \phi$. The main difference is that the modal necessity operator $\Box$ implies truth, whereas obligation does not have such an implication.

In the remainder of this section we define a modal logic, which we call Security Logic (SL). This logic is based on the possible-states model presented in the previous section. We are concerned with the application of the deontic operators to the area of security, and in particular distributed system security, so that the composability problem can be addressed. We restrict our language so that the permission and obligation operators are only applied to knowledge formulas. Thus our language will contain formulas that state that a subject is *obligated to know* or has *permission to know* a particular formula.

4.1 Introduction to Security Logic

We extend the alphabet of the language introduced in Section 3.2 to include:

—The modal operators P (permission) and O (obligation).

The set of formulas for the language is also extended as follows:

—For all i, if ϕ is in W then so are $OK_i\phi$, $PK_i\phi$, $O \neg K_i\phi$, and $P \neg K_i\phi$.

The deontic operators P and O are duals in the modal logic sense. Thus we can define one in terms of the other: $P\phi \leftrightarrow \neg O \neg \phi$. For syntactic convenience, we abbreviate the forms OK_i and PK_i to O_i and P_i. Thus the

proposition $O_i\phi$ is read as *subject i is obligated to know* ϕ and the proposition $P_i\phi$ is read as *subject i has permission to know* ϕ. We do not lose any expressibility in adopting this form, since $O \neg K_i$ can be expressed as $\neg P_i$ and similarly $P \neg K_i$ can be expressed as $\neg O_i$.

The assignment of permissions and obligations to particular formulas is primarily the responsibility of a particular security policy. In our model we denote this assignment in terms of permission and obligation sets. For each subject i, we assume a set Θ_i of formulas that, if true, subject i is obligated to know, and a set Φ_i of formulas that, if true, subject i has permission to know. The formulas occurring in these sets may contain deontic, epistemic, and temporal modalities. We use the permission and obligation sets to define a possible-states semantics for SL.

More precisely, our model is extended in the following way. To every subject i we associate two sets of well-formed formulas, Φ_i and Θ_i. These sets are *not* required to be closed under negation. They are, however, required to satisfy the following closure properties.

(1) If $\phi \wedge \psi$ is in Φ_i then so are ϕ and ψ separately.
(2) If $K_j\phi \in \Phi_i$, $P_j\phi \in \Phi_i$, or $O_j\phi \in \Phi_i$, for any well-formed formula ϕ, then *exists-j* must be in Φ_i, where *exists-j* is a formula asserting that subject j exists.
(3) If $K_j\phi \in \Phi_i$, $P_j\phi \in \Phi_i$ or $O_j\phi \in \Phi_i$, then ϕ is in Φ_i.
(4) If $\forall\Diamond\phi$ is in Φ_i then ϕ must be in Φ_i.
(5) If $\forall|\Box|\phi$ is in Φ_i then ϕ must be in Φ_i.
(6) All instances of propositional tautologies are in each Φ_i.
(7) If $\phi \in \Theta_i$ then $\phi \in \Phi_i$.

Although the description of these sets may be very complicated, a specific policy might be described in a simple way. For example, a multilevel security policy can be described by saying that the structure of these sets is defined using a partial order: *exists-j* is in Φ_i if and only if $level(j) \leq level(i)$, where $level(i)$ is the security level for subject i.

In order to make the link between the well-formed formulas W and the model, now augmented by the sets Φ_i and Θ_i, we give the following truth assignment. The basic intuition is that a subject is permitted to know a formula if that formula is true and is in its permission set. The permissions propagate to more complicated formulas in the following way.

For any formula $\phi \in W$, $s \vDash P_i\phi$ iff $s \vDash \phi$ and

$\phi \in \Phi_i$ or
ϕ is $(\psi \wedge \gamma)$ and $s \vDash P_i\psi$ and $s \vDash P_i\gamma$, or
ϕ is $(\psi \vee \gamma)$ and $s \vDash P_i\psi$ or $s \vDash P_i\gamma$, or
ϕ is $K_j\psi$ (or $P_j\psi$) (or $O_j\psi$) and $s \vDash P_i\psi$ and $s \vDash P_i(exists\text{-}j)$, or
ϕ is $\exists\Diamond\psi$ and $s \vDash P_i\psi$ or
ϕ is $\forall\Diamond\psi$ and $s \vDash P_i\psi$ or
ϕ is $\neg K_j\psi$ (or $\neg P_j\psi$) (or $\neg O_j\psi$) and $s \vDash P_i(exists\text{-}j)$

The above definition gives the truth value of $s \models P_i \phi$ for all possible formulas ϕ, and there is no formula ϕ such that $s \models P_i \phi$ and *not* $s \models P_i \phi$. The definition of permission for conjunction and disjunction is fairly intuitive and straightforward. The definition for the modalities may need some justification. A subject is permitted to know that another subject knows (or is permitted or obligated to know) a formula if and only if the former subject is permitted to know the formula, and also is permitted to know that the latter subject exists. This is because knowledge (and permission and obligation) of a formula implies its truth. A subject is permitted to know that a formula will eventually (or sometimes or always) be true if and only if eventually (or sometimes or always) the subject is permitted to know the formula is true.

The truth value assignments imply that permission holds through conjunction and deduction. They also imply that permission to know a formula implies that the formula is true. These properties are stated as axioms of our logic (see Section 4.2). The soundness of these axioms is proved later in the section.

When defining truth values for permission formulas we are concerned that permission holds through deduction and conjunction. Such a concern does not exist for obligation. The only formulas that a subject can be obligated to know are those that are in the obligation set. For any formula $\phi \in W$, $s \models O_i \phi$ iff $s \models \phi$ and $\phi \in \Theta_i$.

Note that a subject is only obligated or permitted to know true formulas. We make the assumption that the permission set Φ_i is fixed for each i. This implies that we cannot express dynamic user-controlled secrecy policies in the logic. To allow the expression of such policies we could make the permission set state dependent, but for simplicity we restrict ourselves to fixed permission sets and static secrecy policies.

To express the fundamental notions of secrecy and integrity that we wish all policies defined on the logic to capture, we define two properties that express how permission and obligation should interact with knowledge. The first property, the *Secrecy property*, states that if a subject knows a formula, then the subject must be permitted to know that formula. Conversely, a subject will not know any nonpermitted formulas. The second property, the *Integrity property*, states that if a subject is obligated to know a formula then the subject will eventually know it.

We say for any subject i, state s, and formula ϕ:

Secrecy property. If $s \models K_i \phi$ then $s \models P_i \phi$.
Integrity property. If $s \models O_i \phi$ then $s \models \forall \Diamond K_i \phi$.

It is important to be clear about the status of these properties. They are not axioms of the logic nor part of the definition of permission and obligation. Rather, they are properties that *may or may not* be satisfied by specific instances of policies. The point of giving these definitions is to be able to say that such and such a policy instance satisfies, for example, the secrecy property and to be clear about what this means. Furthermore, the definition of secrecy is not wrapped up in the description of a specific policy.

In a case where secrecy is not an issue, the permission sets include all possible formulas. If integrity is not relevant then the obligation sets are empty. Thus policies that consider secrecy or integrity independently or in tandem can be expressed in the logic.

The language of security logic extends beyond the expression of secrecy and integrity. By combining the deontic modalities with those of time and knowledge, we can express and reason about a number of other properties. For example, in some policies we may wish permission to also imply accessibility: if a subject is permitted to know a formula then there should exist a means of finding it out. This could be expressed as the following *Accessibility property:*

Accessibility property. For any subject i, state s, and formula ϕ, if $s \models P_i\phi$ then $s \models \exists\Diamond K_i\phi$.

Another example property, this time for obligation, expresses the requirement to explicitly record in the policy as an obligated fact anything that a subject will eventually know. More precisely, it states that if a subject will always know a formula, then the subject is *de facto* obligated to know it.

Obligation property. For any subject i, states s, and formula ϕ, if $s \models \forall\Diamond K_i\phi$ then $s \models O_i\phi$.

4.2 Axioms for Security Logic

In this section we present an axiom scheme that characterizes SL. This logic is an extension of the theory of knowledge and time defined in the previous section.

The axioms for SL consist of Axioms K1 to K4 for knowledge and the temporal axioms T1–T3. As well, we include the following axioms that describe the relationships between permission and obligation.

Axiom SL1. $P_i\phi$ for all propositional tautologies ϕ.

Axiom SL2. $P_i\phi \rightarrow \phi$.

Axiom SL3. $(P_i\phi \wedge P_i\psi) \leftrightarrow P_i(\phi \wedge \psi)$.

Axiom SL4. $P_i\phi \rightarrow P_i(\phi \vee \psi)$.

Axiom SL5. $O_i\phi \rightarrow P_i\phi$.

Axiom SL1 states that a subject is permitted to know all propositional tautologies. Axiom SL2 states that any formula that is permitted must be true. The fact that permission holds through conjunction and disjunction is expressed in Axioms SL3 and SL4. Finally, Axiom SL5 asserts that any formula that is obligated must also be permitted.

THEOREM. *The axioms SL1 through SL5 are sound with respect to our semantics.*

PROOF

SL1 follows directly from the definition of truth assignment for permission.

SL2 follows directly from the semantics of P_i. For any formula ϕ, $s \models \phi$ is a condition for $s \models P_i\phi$.

SL3 follows from the first closure property on permission sets and the semantics of permission.

SL4 follows directly from the semantics of permission.

SL5 follows directly from the 6th closure property for permission and obligation sets. □

5. DEFINING AND COMBINING SECURITY POLICIES

Various secrecy policy models have been formulated to capture certain information flows that were not captured previously [5, 12, 22, 33]. Most such policies are expressed as restrictions on the set of states for an implementation of the policy. The result is a number of different formulations of secrecy that view the notion of state in a slightly different way. We attempt to capture a large class of such policies within one model, using our logic, by expressing a secrecy policy in terms of what formulas a subject is permitted to know. For example, we can define a multilevel secrecy policy by associating a secrecy level with each formula in some set of useful formulas, and specifying that a subject is only permitted to know formulas that have a secrecy level that is the same or less than the secrecy level of the subject.

The logic also allows us to express integrity policies within the same framework that we use for secrecy policies. We attempt, in a similar way, to capture a large class of such policies within our logic by expressing an integrity policy in terms of what formulas a subject is obligated to know. For example, in the case of Clark and Wilson integrity [8] the set of formulas can be constructed to capture the occurrence of invocations of transformation procedures. Integrity is based on an obligation, at the invocation of a transformation procedure, to know about all invocations upon which it depends.

In this and the following sections we demonstrate how SL provides the basis for defining a security policy and also provides a framework within which one can clarify the meaning of a policy. A *security policy* is given by specifying properties of the sets Φ_i and Θ_i, which define the modalities of permission and obligation, and properties of an interpretation, which define the modality of knowledge. The former properties must be consistent with the closure properties on Φ_i and Θ_i. A *security policy instance* is an interpretation that is a model for the logic SL. As such, an instance defines the set of possible states, the set of formulas, the set of subjects, the truth assignment function, the equivalence relations, and the reachability relation. Since the instance satisfies the logic, the axioms and theorems of the theory hold.

A policy instance is said to be *correct* with respect to a policy if it is a model for SL and satisfies the policy properties. We illustrate below how policies are defined in our theory by defining a multilevel secrecy policy. Since multilevel secrecy is only concerned with secrecy (safety) properties, it does not put any requirements on the obligation sets. We also illustrate how we can combine (or coerce) compatible policies into a single policy that is SL secure and still satisfies all (or some) of the original policy properties. The intuition behind

combining policies is as follows:

—A subject is permitted to know a formula in the combined policy if and only if the subject is permitted to know the formulas in both (all) of the original policies.

—A subject is obligated to know a formula in the combined policy if and only if the subject is obligated to know the formula in one or more of the original policies.

5.1 Defining a Security Policy

A policy is a set of properties as described above, along with any additional information required to express and understand those properties. As an example, let us examine multilevel secrecy within the SL framework. We assume that multilevel secrecy is defined in terms of a set U of subjects, a set L_s of secrecy levels, and a set W of well-formed formulas, along with a lattice $(\preceq, L_s)$ and a function *seclevel*: $U \cup W \rightarrow L_s$.

The function *seclevel* can only be determined by the policy instance because the detailed nature of primitive propositions is not known until the instance is specified.

Given the uninterpreted sets U, L_s, and W we can specify a multilevel secrecy policy as

Multilevel Secrecy. For all $\phi \in W$, $i \in U$, $\phi \in \Phi_i$ iff *seclevel* $(\phi) \preceq$ *seclevel*(i), and the Secrecy property holds.

Multilevel Secrecy states that a subject's permission set comprises all well-formed formulas having a secrecy level dominated by that of the subject. Furthermore, it specifies that only permitted formulas can be known. It is easy to verify that these properties are consistent with the closure properties.

Multilevel Secrecy implies that the secrecy level for a formula or a subject does not change in a given implementation. Our model could be modified to include such changes; if we define the obligation and permission sets in terms of a given possible state, then permission and obligation can be dynamic. However, we do not address this issue here.

5.2 Defining a Security Policy Instance

A policy instance is a model for the logic, given as a tuple $(\Sigma, U, S, \pi, \kappa, R, \Phi, \Theta)$ where

Σ	is the set of primitive propositions
U	is the set of subjects
S	is the set of possible states
π	is the truth assignment function for the primitive propositions
κ	is the set of equivalence relations, κ_i for all subjects i
R	is the reachability relation on S
Φ	is the set of permission sets, Φ_i for all subjects i
Θ	is the set of obligation sets, Θ_i for all subjects i

There are no precise guidelines for the specifier of a policy instance since the issues depend on the nature of the formulas. The specifier first defines the sets Σ of primitive formulas and U of subjects, and in doing so attempts to capture all information relevant to this universe of discourse. The set S of possible states is then defined, along with the truth assignment for primitive formulas, the equivalence relations and the reachability relation, as a specification for a system design that satisfies the policy instance.

To adequately capture a policy, the specifier must identify the threats to that policy. For example, in the case of a secrecy policy such threats can result from public awareness of S. This will usually be the case because user subjects are aware of the set of system states, since system designs are normally public information. This follows from the fact that the security of a system should not depend on keeping secret the details of its construction. Rather, security should depend only on the secrecy of certain small amounts of information such as passwords and cryptographic keys. We say more about threats later, in the context of examples.

5.3 Combining Policies and Connecting Systems

The scope of a computer system that obeys a security policy can be increased in two ways. First, the system can be connected to another system that obeys the same policy. An important question, first posed by McCullough, is whether or not the resultant composite system also obeys the policy [33]. If so, the policy is said to be *composable.* Second, the policy can be combined with a different policy in the context of the same system. An important question is whether or not the resultant combined policy contains the constituent policies. If so, the policies are said to be *compatible.*

5.3.1 *Compatibility.* There are several ways to combine policies; the ideal way is to combine two so that the resulting policy satisfies both of the component policies and is still SL secure. A simple example is a multilevel secrecy policy combined with an access control policy that specifies which objects can be read and/or written by each subject. One would like the two policies not to conflict. That is, if a subject can read an object by the access control policy, then it should be permitted to know the contents of the object by the multilevel secrecy policy. Another application of policy combination is Rushby's proposal for secure system design based on a *separation kernel* [45]. In this proposal, system components, each enforcing a different policy, are isolated from each other by a separation kernel that permits communication between them only via the kernel. It would be important in such a design for the policies not to conflict with each other.

Any policy defined by a set of properties P is said to be *P secure.* If a policy that is P_1 secure and a policy that is P_2 secure can be combined into a single policy that is both P_1 and P_2 secure, we say that policies P_1 and P_2 are *compatible.* This requires that the policy property set $P = P_1 \cup P_2$ is consistent. By consistent, we mean that there exists a policy instance that satisfies all the properties of the resulting set P. If the policy properties are inconsistent, for example they imply that $\phi \in \Phi_i$, and $not(\phi \in \Phi_i)$ for some formula ϕ and subject i, then no such instance exists.

Often two policies are compatible if all the properties of one policy, taken together, imply all the properties of the second policy. In such a case, we say that the first policy, which is stronger, *subsumes* the second and the result of combining the two policies is the first policy.

Even if two policies are not compatible, there may be ways of forcing the policies into a single policy that satisfies some of the properties of the original policies. Given two policies defined as above by property sets P_1 and P_2, find two stronger property sets P_1' and P_2' for which all the properties of P_i', taken together, imply all the properties of P_i. We say that policy P_3 is a *forced merger* of policies P_1 and P_2 if $P_3 = (P_1' \cup P_2')$ and P_3 is SL secure.

Obviously, a forced merger of policies is not as satisfactory as combining compatible policies, since it implies that one or both of the original policies are somehow compromised. However, in cases where the policies are not compatible, it may be necessary to make such compromises if it is necessary to consider more than one policy simultaneously. The advantage of our approach is that it provides a formal foundation for determining how the policies can be forced together.

5.3.2 *Composability.* Composability is different from compatibility because it deals with the result of connecting together different systems, each of which satisfies the same policy, rather than combining together different policies and applying the result to the same system. Composability is only meaningful at the policy instance level because any connection of two systems deals with the design details of those systems, their input and output channels for example. McCullough [33] and others have investigated the composability of instances, which involves some surprising subtleties deriving from the nondeterminism that exists in almost all systems. The interested reader is referred to McCullough's paper for an example, for which we do not have sufficient space here.

Connecting two systems that satisfy the same policy involves forming a composite policy instance. Consider instances $(\Sigma', U', S', \pi', \kappa', R', \Phi', \Theta')$ and $(\Sigma'', U'', S'', \pi'', \kappa'', R'', \Phi'', \Theta'')$. We form a composite instance $(\Sigma, U, S, \pi, \kappa, R, \Phi, \Theta)$ as follows:

—$U = U' \cup U''$; some subjects may exist in both systems.

—$S \subseteq S' \times S''$; some state combinations may be precluded by the composition. That is, S cannot be determined without information about the interdependencies between state transitions in the component instances.

—$\Sigma = \Sigma' \cup \Sigma''$

—π is formed as follows: for all states $s = (s', s'')$,
if $\phi \in \Sigma'$ and $\phi \in \Sigma''$ then $s \vDash \phi$ iff $\pi'(s', \phi)$ and $\pi''(s'', \phi)$
if $\phi \notin \Sigma''$ then $s \vDash \phi$ iff $\pi'(s', \phi)$
if $\phi \notin \Sigma'$ then $s \vDash \phi$ iff $\pi''(s'', \phi)$

—κ is formed as follows: for all states $s_1 = (s_1', s_1'')$, $s_2 = (s_2', s_2'')$, subjects i, $(s_1, s_2) \in \kappa_i$ iff $(s_1', s_2') \in \kappa_i'$ and $(s_1'', s_2'') \in \kappa_i''$

—R also cannot be determined without information about the interdependencies between state transitions in the component instances. A transition in

one component is *independent* if it can occur with no effect on the other component. A pair of transitions, one from each component, are *dependent* if they can occur only in synchronization. This precludes a state transition from a dependent pair from occurring by itself, for all states $s_1 = (s'_1, s''_1)$, $s_2 = (s'_2, s''_2)$,
$(s_1, s_2) \in R$ iff
$(s'_1, s'_2) \in R'$ and $s''_1 = s''_2$ and (s'_1, s'_2) is independent, or
$(s'_1, s'_2) \in R'$ and $(s''_1, s''_2) \in R''$ and $[(s'_1, s'_2), (s''_1, s''_2)]$ are dependent

—Φ is formed as follows: for all subjects i,
if $i \in U'$ and $i \in U''$ then $\Phi_i = \Phi'_i \cup \Phi''_i$
if $i \notin U''$ then $\Phi_i = \Phi'_i$
if $i \notin U'$ then $\Phi_i = \Phi''_i$;

—Θ is formed as follows: for all subjects i,
if $i \in U'$ and $i \in U''$ then $\Theta_i = \Theta'_i \cup \Theta''_i$
if $i \notin U''$ then $\Theta_i = \Theta'_i$
if $i \notin U'$ then $\Theta_i = \Theta''_i$

Since composability can be addressed only at the instance level, we can say nothing here about policy property composability.

6. EXAMPLES

In this section we discuss an information flow instance of multilevel secrecy, an integrity policy, two access control instances of multilevel secrecy, and an access-control-based policy embodying both secrecy and integrity. Our purpose here is to illustrate how one can use SL to specify and reason about policies and policies instances. These examples are necessarily simplified and briefly presented; further examples will be presented in separate papers.

First we give an interpretation for *Multilevel Secrecy*, loosely based on the system model used by Guttman and Nadel in defining their Input/Output Nondeducibility policy [22] and show that it is correct with respect to Multilevel Secrecy. Second, we specify an integrity policy that is based on Clark and Wilson's integrity policy [8]. We show that this integrity policy is compatible with Multilevel Secrecy. Third, two access control instances, based on a simplified version of McLean's access control policy framework [37], are specified. We show that the first, which is trace-based, is not correct with respect to Multilevel Secrecy, but that the second, which is state-based, is correct. Finally, we formulate a policy to correspond to the mandatory access control layer of the Seaview hierarchical model and show that this Seaview instance is correct with respect our policy.

6.1 An Information Flow Instance of Multilevel Secrecy

From the information flow models[3] that have appeared in the literature recently we arbitrarily chose a simple model based on Guttman and Nadel's

[3] The term model is used in this way here, since it is commonly used in the literature to describe what we have been calling a policy instance. The reader should not confuse such a model with a model of the logic. Although such a model may indeed provide a model for the logic.

Input/Output Nondeducibility model, a multilevel information flow secrecy model that is based on observations of a system, via input and output events, called traces.[4] This simplified information flow model, which we call simply *Nondeducibility* (*ND*), can be specified as an interpretation $(\Sigma, U, S, \pi, \kappa, R, \Phi, \Theta)$, as follows. First we specify the formulas Σ.

A *trace* is a finite sequence of interleaved input and output events that a system can exhibit. A system is specified by the set of traces it can generate. The ND definition of secrecy is a restriction on this set of traces. A reasonable restriction on any trace set is that it is prefix-closed. That is, all prefixes of a trace, including the empty trace ϵ, are also traces. In addition, input events are always possible. So, for every trace t and input event e, te is also a trace.

Subjects communicate to and from a system via a set C_i of *input channels*, and a set C_o of *output channels*, where $C = C_i \cup C_o$. Channels are associated with a secrecy level via the function *chseclevel*: $C \rightarrow L_s$. Data flows into and out from a system in the form of messages on channels. Each datum in a message is taken from a set D of message data. The set M contains messages of the form $(c, d), c \in C, d \in D$. The level of a message is the level of the channel on which it appears. The set of potential traces for a system is then M^*, the set of all finite sequences of messages. A security policy for a system allows any implementation of the policy to generate only some specified subset of M^*.

Informally, the security policy that we want to capture states that a subject must not know any information about the activity of any other subject that is not at the same or a lower level. In the context of the trace model, we can say that, for any execution of the system (i.e., for any trace), a subject should only be able to know about any interleaved subtrace containing only messages of the same or a lower level. It should not be able to deduce that the execution (trace) contains any subtrace containing higher or incomparable messages.

First we define a one-to-one correspondence σ: $M^* \rightarrow \Sigma$ to specify the primitive propositions. The converse correspondence is σ': $\Sigma \rightarrow M^*$. In other words, every possible finite sequence of messages has an associated proposition asserting that that sequence occurred. Suppose that the system generates a trace t. Then, for that execution of the system, a proposition ϕ is true if $\sigma'(\phi)$ is an interleaved subsequence of t, and false otherwise. We use $s_1 \sqsubseteq s_2$ to denote that s_1 is an interleaved subsequence of s_2; this means that s_1 is obtained by deleting zero or more messages from s_2. Also, $m \epsilon s$ denotes that the message m appears in the sequence s.

The set of subjects U is characterized by associating each subject with a channel via the function *channel*: $U \rightarrow C$ and specifying that for all $i \in U$, $seclevel(i) = chseclevel(channel(i))$.

A trace defines not only a final state but how the system got to that state (or could have gotten to that state). We interpret the possible states S of the logic to be the set of traces.

[4] For simplicity, we omit the distinction between user and system events that is an important part of their model.

The assignment π of a truth value to each primitive proposition is given by: for all $t \in S, \phi \in \Sigma, \pi(t, \phi)$ iff $\sigma'(\phi) \sqsubseteq t$. Notice that by this definition the proposition corresponding to the empty trace $\sigma(\epsilon)$ is true in all states.

The specification of the function *seclevel* is based on two functions *upper seclevel* and *lowerseclevel*, which are used to assign a secrecy level to primitive propositions and to their negation respectively. These functions are based on the secrecy lattice $(\preceq, L_s)$ introduced in the previous section. For primitive propositions ϕ, *upperseclevel*(ϕ) yields the least upper bound (lub) value of l which for all $(c, d) \in \sigma'(\phi)$, *chseclevel*$(c) \preceq l$. For primitive propositions ϕ, *lowerseclevel*(ϕ) yields the greatest lower bound (glb) value of l which for all $(c, d) \in \sigma'(\phi)$, $l \preceq$ *chseclevel*(c).

For a primitive proposition ϕ that is not negated we use *upperseclevel* to assign a level, since we want a subject, knowing that a trace occurred, to have a level permitting the occurrence of each event in that trace to be known. For a formula of the form $\neg \phi$, where ϕ is a primitive proposition, we use *lowerseclevel* to assign a level to the formula since a subject, knowing that some low-level event did not occur, can construct a trace containing that event which, of course, also did not occur. So if a subject has permission to know that some subtrace did not occur, it must also have permission to know that any trace of which it is a subtrace did not occur.

Now the function *seclevel* for a formula $\phi \in \Sigma$ and its negation can be given as

seclevel(ϕ) = if $\phi \in \Sigma$ then *upperseclevel*(ϕ)
else if $\exists \psi . \phi = \neg \psi$ and $\psi \in \Sigma$ then *lowerseclevel*(ψ)

meaning that *seclevel* is a partial function, being defined only for primitive formulas and their negations.

The definition of the equivalence relations κ_l is based on the function *seclevel*. We use the subscript notation s_l to mean the interleaved subsequence obtained from s by deleting all messages whose level is not dominated by l. The equivalence relations are given by: for all $t', t'' \in S, i \in U, (t', t'') \in \kappa_i$ iff $t'_{seclevel}(i) = t''_{seclevel(i)}$.

This captures the fact that a subject at a particular level can only view inputs and outputs at or below that level. Any two system traces with the same subsequence of such messages look identical to that subject.

The reachability relation R is given by: for all $t', t'' \in S, (t', t'') \in R$ iff there exists sequence $s \in M^*$ such that $t's = t''$.

The sets Φ and Θ of permission and obligation sets in the instance are taken to be as given by the Multilevel Secrecy policy.

We complete the definition of the Nondeducibility interpretation by constraining the states. In the following the subscript $\neg l_o$ means the trace with low outputs removed.

Definition 6.1. An interpretation is *Nondeducibility-constrained* iff for all $t, t' \in S, s \in M^*$, if $t_l = s_l$ and $t'_{\neg l_o} = s_{\neg l_o}$ then $s \in S$.

To show that the Nondeducibility interpretation is a correct instance with respect to Multilevel Secrecy, we need to show that it is a model for the logic

SL with the specified properties for Φ and Θ, and that it satisfies the Secrecy property. It is easy to see that it is a model. Therefore, we can show that the Secrecy property follows from the properties on Φ and Θ and the Nondeducibility constraint. To do this, we show that if a formula is not in Φ_i according to the policy, then according to the Nondeducibility constraint, it cannot be known by subject i. It is important here that the definition of truth assignment for compound formulas ensures that if the Secrecy property holds for all formulas in Φ_i, then it holds for any other permitted compound formula. As a result, the proof needs to be carried out only for formulas in the permission set Φ_i, i.e., only for primitives and their negations.

THEOREM 1. *A Nondeducibility-constrained interpretation is correct with respect to Multilevel Secrecy.*

PROOF. As discussed above, the Nondeducibility interpretation is a model and the Multilevel Secrecy property on Φ_i holds. Now we show that if a formula is not in Φ_i according to the policy, then, according to the Nondeducibility constraint, it cannot be known by subject i.

Consider any subject i, any state t, and any formula ϕ of the form ψ or $\neg\psi$ where $\psi \in \Sigma$. Assume that $not(\phi \in \Phi_i)$. By Multilevel Secrecy,

$$seclevel(\phi) \not\preceq seclevel(i). \tag{1}$$

We need to show that the subject i cannot know ϕ. That is, from the Secrecy property, we must show that

$$\text{there exists } t', (t, t') \in \kappa_i \quad \text{and} \quad t' \vDash \neg\phi. \tag{2}$$

There are two cases required to show (2):

Case I: $t \vDash \neg\phi$. Therefore, there exists $t', (t, t') \in \kappa_i$ and $t' \vDash \neg\phi$.

Case II. $s \vDash \phi$. First, consider the form $\phi = \psi$ where $\psi \in \Sigma$. From (1),

$$\phi \not\sqsubseteq t_{seclevel(i)}. \tag{3}$$

From the truth assignment function π, $\phi \sqsubseteq t$.

Consider the trace t' containing only the low (relative to $seclevel(i)$) inputs from t; it must be a trace, since any input can be appended to any trace to form a trace. Take (t, t') to be the pair of traces denoted similarly in the definition of Nondeducibility. Now construct a t'' to be the required trace, according to the definition, which will contain only the low events from t. Now $(t, t'') \in \kappa_i$ as required by (2). Also, $\phi \not\sqsubseteq t''$ since t'' contains only low events from t, but from (3), ϕ contains at least one event that is not a low event. Therefore, $t'' \vDash \neg\phi$.

Now consider the form $\phi = \neg\psi$ where $\psi \in \Sigma$. From (1) and the form of ϕ, $seclevel(\neg\psi) \not\preceq seclevel(i)$. Therefore, from the definition of *lowerseclevel*, ψ contains only high-level events.

Now consider some trace u for which ψ is a subtrace; one must exist since $\neg\psi \notin \Phi_i$ and this would be contradicted if $\neg\psi$ were valid. Now form a trace t' by appending to u the low inputs from t. The resulting t' must be a trace,

since any input can be appended to any trace to form a trace. Take (t, t') to be the pair of traces denoted similarly in the definition of Nondeducibility. Now construct a t'' to be the required trace, according to the definition, which will contain all high events and low inputs from t' and all low events from t. Now $(t, t'') \in \kappa_i$, as required by (2).

Also, $\psi \sqsubseteq t''$ since ψ contains only high events, and by the construction of t'' all of these are included in t''. Therefore, $t'' \models \psi$ and $t'' \models \neg(\neg \psi)$ and $t'' \models \neg(\phi)$. □

6.2 Syntactic Reasoning

This proof of secrecy for a particular policy instance has been carried out at the semantic level in SL. It is instructive to consider, for a moment, that SL does not give us a deductive system with which to carry out such proofs and why one might want such a deductive system. First of all, simple propositional logic, upon which SL is built, does not give us a rich enough language to easily specify a policy, axiomatize a policy instance, and prove properties. We have restricted ourselves to simple propositional logic to reduce the complexity of SL, which is already quite high. However, if SL were based on a richer language and supplied with a deductive system, we might be able to specify the policy with $P_i\phi \leftrightarrow \mathit{seclevel}(\phi) \preceq \mathit{seclevel}(i)$, characterize the policy instance with appropriate axioms, and use the deductive system to prove the Secrecy theorem:

$$K_i\phi \rightarrow P_i\phi.$$

A further use of a richer language is threat analysis. For example, it is clear that a considerable amount of threat analysis must have preceded the formulation of the Nondeducibility policy instance. A richer language would enable a policy specifier to express and reason about the threats and to use that information in proving properties of a policy instance. Most important, however, the threats would be formally documented as a part of the policy statement. We illustrate this idea in the following paragraphs.

The threats in the Nondeducibility policy instance result from the structure of formulas with respect to the subsequence relation $\sqsubseteq$ on their associated traces and from the definition of knowledge. In particular, if a subject observes in some state that a subtrace s did not occur, it can deduce that any trace of which s is a subsequence could not have occurred. Conversely, observing that a trace t occurred yields the information that all subtraces of it also occurred. In a richer language, one could express this as:

Threat $T1$. For all $i \in U, t \in S$, and $s \in M^*$, if $s \sqsubseteq t$ then $K_i \neg \sigma(s) \rightarrow K_i \neg \sigma(t)$.

Threat $T2$. For all $i \in U, t \in S$, and $s \in M^*$, if $s \sqsubseteq t$ then $K_i\sigma(t) \rightarrow K_i\sigma(s)$.

Capturing the threat that results from the subjects' awareness of the set of system states is more difficult. We use the subscript notation $s_{\neg l}$ to mean the interleaved subsequence obtained from s by deleting all messages whose

level is dominated by l. The subscript l_i denotes the subsequence containing only inputs dominated by l.

Consider some subject i with $l = seclevel(i)$. We use the term low to mean all levels dominated by l and high to mean all other levels. The third threat says that if there is a trace t such that there is no other trace with the same low messages but different high messages, then the subject, seeing t_l, can conclude that $t_{\neg l}$ occurred. More precisely:

Threat $T3$. For all $i \in U, t, t' \in S$, if (if $t_l = t'_l$ then $t_{\neg l} = t'_{\neg l}$) then $K_i\sigma(t_l) \rightarrow K_i\sigma(t_{\neg l})$.

The fourth threat says that if there is a trace t such that there is no other trace with the same low messages but different high messages and there is another trace t'' with the same low inputs but different high messages, then a subject seeing t_l can conclude that $t''_{\neg l}$ did not occur. More precisely:

Threat $T4$. For all $i \in U, t, t' \in S$, if (if $t_l = t'_l$ then $t_{\neg l} = t'_{\neg l}$) and (there exists $t'' \in S, t''_{l_i} = t_{l_i}$ and $t''_{\neg l} \neq t_{\neg l}$) then $K_i\sigma(t_l) \rightarrow K_i \neg \sigma(t''_{\neg l})$.

To address these threats in Nondeducibility, the set of traces S is constrained following the rationale that low-level outputs do not provide any information about high-level events. Thus, if we have a trace t that agrees with the low-level inputs of a trace t'', then the high events of t'' can always be combined with the low events of t to form an alternative trace t'. This alternative trace t' will be indistinguishable from t to a subject with a low secrecy level. Consequently, although the subject can deduce not $\sigma(t'')$ from $\sigma(t)$ (via T1), it cannot deduce not $\sigma(t')$. Therefore, it cannot determine if the high events of t'' did (in t') or did not (in t'') occur.

6.3 A Policy for Integrity

Integrity has been the focus of a large amount of recent activity in the computer security community [40]. Since publication of the Clark–Wilson integrity model [8], there has been an effort to investigate implementations for their model and to explore the nature of integrity in order to find other useful definitions. However, since the proposal by Biba to view integrity as an exact dual of secrecy [4], there have been few attempts to formalize integrity (one is discussed later in this section). Our thesis is that obligation can be used as the fundamental notion in formally specifying integrity. In [17], we discussed integrity as seen by Clark–Wilson-type models and argued that the essential part of their model, for the purpose of formal specification, is an expression of required connectivity in a graph representing *transformation procedures* in a computer system. This is their notion of *well-formed transactions*.

In the Clark–Wilson model, a *transformation procedure* (*TP*) is a program that is certified to consume its inputs and to produce correct output. A TP can be invoked by a user subject acting in a certain *role*. That is, roles and users are distinct; a user may act in several roles and a role may be played by more than one user. Inputs and outputs are represented by *data items* (*DI*s). A *constrained data item* (*CDI*) is an object that can occur as an input parame-

ter or an output parameter to a TP. An *unconstrained data item* (*UDI*) is an object that can occur only as an input parameter to a TP. A set of *integrity relations* (our term) of the form $(ROLE_i, TP_j, DI_k, DI_l, DI_m, \ldots)$ is established to constrain the execution of procedures. Each tuple of DIs in a relation must be consistent with the number and ordering of the parameters of the specified procedure. A *well-formed transaction* is any sequence of role/procedure executions that conform to the set of relations.

An essential notion captured is that each certified procedure, when executed on behalf of a specified role, is guaranteed to obtain its inputs from certain named sources, each of which is either a specified UDI or a specified CDI, which in turn has been produced by a certified procedure. Integrity, therefore, is a requirement that certain information flows take place between procedures.

It is illuminating to contrast the requirement of this integrity model with the requirement of information flow secrecy models [12, 22, 26, 35, 46]. Secrecy is a negative assertion that certain information flows must not occur. This is what is commonly called a *safety* property; that is, a statement that some event or events must not occur. With integrity, on the other hand, we seem to have a positive assertion that certain information flows *must be able to occur* or, in the stronger form, *must eventually occur.* In terms of Clark–Wilson, a specified (*ROLE*, TP) must be able to communicate correctly with another (*ROLE*, TP). In their model, this is done through CDIs with controlled access.

The duality between integrity and secrecy that we describe is quite different from the duality in the Biba model [4]. There, duality is based on the secrecy lattice and both policies, secrecy and integrity, express a negative constraint. Integrity constrains flow to be *down only* in the lattice, and secrecy constrains flow to be *up only*. Although Biba's integrity specifies that certain flows must not occur, Clark–Wilson-type integrity specifies that certain flows must be able to occur (or, in the case of strong integrity, must occur).

The contrast with Biba's model as a dual of secrecy shows other differences as well. The well-formed transaction model requires additional constraints on the interconnectivity among TPs; it is not only important that a particular output of one TP be connected to another TP, but it must be specified to be connected to a particular input of that TP.

Is it necessary to be able to express a flow prohibition of the kind addressed by Biba, that of nontampering of the data of a high integrity user by a low integrity user? This is a negative constraint; information from a low integrity user should not flow to a high integrity user. At first glance, this seems to be a different concern than that expressible in the well-formed transaction model. However, the motivation is the same, to ensure proper flow of information between entities in a system. If one can guarantee that specified flows occur properly, then it should be unnecessary to prohibit all other flows. This seems to be what Clark and Wilson intended.

Integrity is very much a property of applications. In terms of the well-formed transaction model, the application-specific requirements appear as correct-

ness specifications for the TPs. These, however, must be treated outside a general model that expresses a mandatory policy for integrity.

An *Integrity policy* is specified by defining the sets Θ_i. We now outline how this can be accomplished for a Clark–Wilson-type integrity model using the notion of strong integrity to specify a policy called *Transactional Integrity*. We take a Clark–Wilson-type abstract integrity model to comprise:

A set *RO* of *roles*.

A set *TP* of *transformation procedures*.

A set *DI* of *data items*.

An *execution template* is a tuple (ro, tp, I) where $ro \in RO$, $tp \in TP$, and I is a relation of the form $(di_1, di_2, \ldots)$, where $di_j \in DI$, and each element in I conforms to the number and type of the parameters of tp. In other words, I is the set of actual parameter combinations allowed for the role ro executing tp. An *execution instance* is a tuple (ro, tp, P) where $P \in I$.

We now define an *integrity unit* to be a tuple (ro, tp, di) where (ro, tp, P) is an execution instance and di is an output data item in P. The set of integrity units is IU. For unit $j = (ro, tp, di)$, $D(j)$ denotes di. We assume the existence of an *integrity flow relation* Ω with elements (i, j) where i and j are integrity units. Ω is the set of all unit pairs (i, j) such that there exist two execution instances (ro_1, tp_1, P_1) and (ro_2, tp_2, P_2) in which $D(i)$ is an output in P_1 and an input in P_2, and $D(j)$ is an output in P_2. Intuitively, each member of Ω represents the use of a particular CDI as a buffer of information between two particular (ROLE, TP) instances. The specification of Ω is in the scope of the application. One can view the CDI as an implementation mechanism to enforce flows required in the integrity policy.

Integrity units are disjoint from users. One can think of an integrity unit as an abstract entity that represents a particular set of information in the application model. $\Omega(i, j)$ means that j gets information from i. Since users invoke integrity units and are the entities that carry permission and obligation in the logic, we need, of course, to relate the two. This is done with the relation INV where $(u, i) \in INV$ means that user u invokes unit i. The specification of INV is in the scope of the application.

We now define the obligation sets Θ_i. To do this we must interpret an integrity unit in the system model. This can be done naturally using the same set of system traces S used in the secrecy policy model. As before, we take the notion of an event to be an abstraction of occurrences at the system interface. An invocation of an integrity unit is described by a sequence of events. For each integrity unit i, a subset $S[i]$ of the traces in S is precisely the set of event subsequences that invokes i. Any trace from $S[i]$ represents one interaction of a user in invoking the associated unit. The obligation set Θ_i is formed as follows (For notational convenience we apply σ, the correspondence of sequences to propositions, to sets of sequences with the obvious meaning):

Transactional Integrity. For all $i \in U, j, j' \in IU, \phi \in \Theta_i$ iff $(INV(i, j)$ and $\phi \in \sigma(S[j])$ or $\phi' \in \Theta_i$ and $\phi' \in \sigma(S[j'])$ and $\Omega(j, j'))$.

The intuition here is that a user i knows the formula associated with any input sequence it produces. Consequently, Transactional Integrity says that i is obligated to know the set of formulas associated with the integrity units it invokes. Furthermore, i is also obligated to know, for any unit j' that i invokes, the formulas associated with all input units to j'.

THEOREM 2. *Transactional Integrity and Multilevel Secrecy are compatible.*

PROOF. Here we must show, given the two policies, that for all ϕ, if $\phi \in \Theta_i$ then $\phi \in \Phi_i$.

First consider any formula included in the obligation set by the first part of Transactional Integrity, which says that for all $i \in U, j, j' \in IU$,

$$\text{if} \quad INV(i,j) \quad \text{then} \quad \sigma(S[j]) \subseteq \Theta_i \tag{4}$$

where $S[j]$ denotes the set of input event sequences that invoke integrity unit j. From the system model any input sequence produced by a user i will have a level that is dominated by the level of i; in particular,

$$\text{for all } t, \quad \text{if } INV(i,j) \text{ and } t \in S[j] \text{ then } seclevel(\sigma(t)) \preceq seclevel(i). \tag{5}$$

The premise of (5), by (4), means that $\sigma(\mathrm{t}) \in \Theta_i$. The conclusion of (14), by Multilevel Secrecy, means that $\sigma(\mathrm{t}) \in \Phi_i$. Therefore, $O_i\sigma(t) \rightarrow P_i\sigma(t)$.

Thus, for any formula representing a trace directly invoked by a user, obligation implies permission.

Now consider any formula included in the obligation set by the second part of Transactional Integrity, which says that for all $i \in U, j, j' \in IU$,

$$\text{if} \quad \sigma(S[j']) \subseteq \Theta_i \quad \text{and} \quad \Omega(j,j') \quad \text{then} \quad \sigma(S[j]) \subseteq \Theta_i. \tag{6}$$

Suppose that there exists a t' such that $t' \in S[j']$ and $\sigma(t') \in \Phi(i)$ and $\Omega(j,j')$. Then by (6) there exists a t such that $t \in S[j]$ and $\sigma(t) \in \Theta_i$.

Assume that $\sigma(t) \notin \Phi_i$ and show a contradiction as follows. By Multilevel Secrecy, $seclevel(\sigma(t)) \not\preceq seclevel(i)$.

Consider the concatenation tt' of the traces t and t'. Since $\Omega(j,j')$, it can be shown that $\sigma(tt') \in \Phi_i$. Therefore, $seclevel(\sigma(tt')) \preceq seclevel(i)$ and $seclevel(\sigma(t)) \preceq seclevel(i)$, which is a contradiction. Therefore, $\sigma(t) \in \Phi_i$. □

6.4 An Access Control Instance of Multilevel Secrecy

We now specify two multilevel access control policy instances to demonstrate that correctness is relative to the interpretation chosen. To do this, we first choose an interpretation based on traces as the possible states, which we call *Trace-Based Access Control,* and show that it is not correct with respect to Multilevel Secrecy. We then choose an interpretation based on a state machine model and show that it is correct with respect to Multilevel Secrecy. Our definitions are intended to apply to any access control policy in which levels do not change. However, we were motivated by McLean's framework

[37], which is intended for policies in which the levels do change, because he is also interested in providing a framework for comparing policies.

First consider Trace-Based Access Control. We start with the interpretation for Nondeducibility and refine it as follows: There is a set O of *objects*. Objects have values in the domain *Val*. The function *objseclevel* is given as *objseclevel*: $O \rightarrow L_s$. The set of *object states* is given as a set V of functions such that for all $v \in V, v$: $O \times U \rightarrow Val \cup \{null\}$. The value *null* can be returned to a subject in the case that access to the object is not permitted. The behavior of a system is specified by an *execution function* E: $V \times M \rightarrow V$.

Given an initial state v_0, an execution function E, and a trace t, a final system state is computed by the function *apply* as follows: $apply(E, t, v_0) = v$ if there is a sequence $(v_0, v_1, \ldots, v_n)$ such that for all $1 \leq i \leq n$, $E(v_{i-1}, t_{i-1}) = v_i$ and $v_n = v$.

A state is *secure* if the levels of all objects that are available to a subject i have a level that is dominated by the level of i.

Definition 6.2. A *state* v in Trace-Based Access Control is *secure* iff for all $i \in U, o \in O$, if $v(o, i) \neq null$ then $objseclevel(o) \preceq seclevel(i)$.

We now specify the execution function E as the essential part of the policy instance.

Definition 6.3. An execution function E is *Access Control-constrained* iff for all $t \in S, v \in V$, if v is secure then $apply(E, t, v)$ is secure.

The function E is such that if one starts in an initial secure state v, and executes a trace t, then the final state is secure.

The relationship between the policy Multilevel Secrecy and the instance Trace-Based Access Control is that a subject knowing a formula $\phi \in \Sigma$ can compute a state v by applying E to $\sigma'(\phi)$. However, note that Trace-Based Access Control does not restrict the set of traces S, but, rather, given a set S restricts the execution function E. In doing so, it allows sets S that do permit some flows that should be disallowed under Multilevel Secrecy. Consequently, we have:

THEOREM 3. *Trace-Based Access Control is not correct with respect to Multilevel Secrecy.*

PROOF. Consider the following counter example. A system has one low input channel, one low output channel, and one high output channel. The low input event vocabulary is $\{a\}$. The low output event vocabulary is $\{c, d\}$. The high output event vocabulary is $\{x, y\}$. The trace set $S = \{\epsilon, a, ax, ay, axc, ayd\}$. The outputs represent the transmission of the values of corresponding objects C, D, X, and Y of which C and D are low and X and Y are high. This system design is Access Control constrained. However, by considering threat T3, one can see that $K_l\sigma(ac) \rightarrow K_l\sigma(x)$. But by Multilevel Secrecy, $\sigma(x) \notin \Phi_l$, so it does not satisfy Multilevel Secrecy. □

Now consider a state-based interpretation that we call *State-Based Access Control*. We start with the same trace abstraction of subjects, messages,

channels, the lattice L_s, and the function *seclevel* as we used for Nondeducibility. We augment this with the set O of objects, the function *objseclevel*, the set of object states V and the execution function E; all are defined as in Trace-Based Access Control. However, the interpretation $(\Sigma, U, S, \pi, \kappa, R, \Phi, \Theta)$ is defined quite differently.

The set Σ of primitive formulas is derived from the set of all pairs of the form (o, x) where $o \in O$ and $x \in Val$. This is done with the functions ρ and ρ':

$$\rho: \quad O \times Val \rightarrow \Sigma$$
$$\rho': \quad \Sigma \rightarrow O \times Val$$

For convenience, we also use the selector functions *first* and *second*:

$$first: O \times Val \rightarrow O$$
$$second: O \times Val \rightarrow Val$$

and define

$$obj(\phi) = first(\rho'(\phi))$$
$$val(\phi) = second(\rho'(\phi))$$

The function *seclevel*[5] on formulas is defined by $seclevel(\phi) = objseclevel(obj(\phi))$.

So the value of a primitive formula is a proposition of the form *the value of object o is x*. The set U of subjects is the same as for Trace-Based Access Control. The set S of states is taken to be the set V of object states. The set of traces is denoted T. The truth assignment π is defined as follows. For all $v \in S$, $\phi \in \Sigma$, $\pi(v, \phi)$ iff exists $i \in U, v(obj(\phi), i) = val(\phi)$. So a formula (o, x) is true in a state if and only if that state returns x as the value of o to at least one subject. Notice that a state can return different non-null values for an object o to different subjects.

The equivalence relations κ_i are defined as follows. For all $v', v'' \in S, i \in U, (v', v'') \in \kappa_i$ iff for all $o \in O, v'(o, i) = v''(o, i)$. States that return exactly the same set of object values appear equivalent to a subject.

The reachability relation R is defined as follows. For all $v', v'' \in S, (v', v'') \in R$ iff there exists $t \in T, apply(E, t, v') = v''$.

The relationship between Multilevel Secrecy and State-Based Access Control is slightly different than that for Trace-Based Access Control. Here, a subject knowing a $\phi \in \Sigma$ has some information about an object's value, but has no information about how that object acquired that value. Although, like the trace-based instance, the state-based instance does not restrict the set of traces, but that is irrelevant to a proof of correctness because the formulas capture only the value of each object and not how those values were computed. (After showing the proof we point out a serious deficiency resulting

[5] For simplicity, we use the same name here as for the analogous function on trace-based propositions. The two are not used in the same context, so no confusion should result.

from this.) Consequently, we have:

THEOREM 4. *State-Based Access Control is correct with respect to Multilevel Secrecy.*

PROOF. The structure of this proof is similar to that of Theorem 1. Consider any subject i, any state v, and any formula $\phi = (o, x)$. Assume that $not(\phi \in \Phi_i)$. By Multilevel Secrecy,

$$seclevel(\phi) \not\leq seclevel(i). \tag{7}$$

We need to show that the subject i cannot know ϕ. That is, from the Secrecy property, we must show that

$$\text{there exists } v', (v, v') \in \kappa_i \quad \text{and} \quad v' \vDash \neg \phi. \tag{8}$$

From (7) and the definition of *seclevel,*

$$objseclevel(o) \not\leq seclevel(i). \tag{9}$$

From the definition of E and (9) and assuming a secure initial state v_0,

$$\text{for all traces } t \in T,\ apply(E, t, v_0)(o, i) = null. \tag{10}$$

Therefore, $v(o, i) = null$.

Now take v' to be the state identical to v except that $v'(o, j) = null$ for all j. Then, $(v, v') \in \kappa_i$ and by the definition of π, $v' \vDash \neg \phi$. Consequently, (8) holds. □

Although State-Based Access Control is correct with the interpretation we have given, it misses a significant security threat. The flaw in our interpretation is that it does not capture how values are computed. Put another way, it does not capture the information flows that result from writes to objects, and consequently is only applicable for read-only objects. More precisely, it assumes that the function *objseclevel* is state independent, whereas in fact writing into an object can effectively raise its secrecy level. The policy instance in the following section addresses this threat.

6.5 A Database Access Control Policy

The Seaview database security policy [31, 32] comprises two parts, a mandatory access control policy and a discretionary access control policy. Seaview runs on an operating system that enforces the mandatory policy; the discretionary policy is carefully designed to use the objects protected by the operating system in such a way that the mandatory enforcement is left entirely to it. It is therefore of some interest to show that the discretionary policy is compatible with the mandatory policy, to ensure that discretionary enforcement is not required to violate the mandatory policy. Although this compatibility problem is left for a subsequent paper, we set the stage by formulating a version of the mandatory policy in SL.

The Seaview mandatory policy instance addresses both secrecy and a particular notion of integrity. In this section we use SL to define a restricted *Multilevel Combined* (secrecy and integrity) policy that abstracts from this

Seaview policy instance. We show that the Seaview instance, is Multilevel Combined secure.

Multilevel integrity is defined in terms of the same set U of subjects and a set L_t of integrity levels, along with a lattice $(\preceq, L_i)$, and functions *chintlevel:* $C \rightarrow L_t$ and *intlevel*: $U \cup W \rightarrow L_i$ where $intlevel(i) = chintlevel(channel(i))$.

Given the uninterpreted sets U, L_t, and Σ, we could specify a multilevel integrity policy as:

Multilevel Integrity. For all $\phi \in \Sigma, i \in U, \phi \in \Theta_i$ iff $intlevel(i) \preceq intlevel(\phi)$ and the Integrity property holds.

The intended meaning here is that information at a high (or equal) integrity level with respect to a subject i must be accessible to i. In other words, other subjects at a lower integrity level than i should not be able to interfere with i's ability to obtain that information. This, of course, requires strong integrity.

We could then take Multilevel Secrecy and Multilevel Integrity together, and define the following policy.

Multilevel Secrecy and Integrity. For all $\phi \in \Sigma, i \in U, \phi \in \Phi_t$ iff $seclevel(\phi) \preceq seclevel(i)$ and $\phi \in \Theta_t$ iff $intlevel(i) \preceq intlevel(\phi)$ and the Secrecy and Integrity properties hold.

However, we will see that such a policy is not relevant to Seaview because of the Integrity property; the notion of integrity in Seaview is quite different from that presented above. Integrity in Seaview is more like secrecy than integrity as we have presented in this paper. Consequently, without discussing the details of the Seaview policy instance (which we shall do presently), we give below a property to be used in place of the Secrecy and Integrity properties. Using the state-based model of access control, for all states v, subjects i, and formulas ϕ, we have:

Seaview property. If $v \vDash K_t\phi$ then $v \vDash P_t\phi$ and $v \vDash O_t\phi$

and the associated policy:

Multilevel Combined. For all $\phi \in \Sigma, i \in U, \phi \in \Phi_t$ iff $seclevel(\phi) \preceq seclevel(i)$ and $\phi \in \Theta_t$ iff $intlevel(i) \preceq intlevel(\phi)$ and the Seaview property holds.

Notice that we could have omitted the text "$v \vDash P_t\phi$ and" from the Seaview property, since in *SL* obligation implies permission. It was not omitted to emphasize that the Seaview policy instance does not include such an implication.

We now use SL to express a formulation of the Seaview mandatory policy instance of the Multilevel Combined policy. We start with the state-based policy instance developed previously, and modify it to express the Seaview instance. First we combine the integrity lattice $(\preceq, L_i)$ with the secrecy lattice, to form the lattice $L, L = L_s \times L_i^{-1}$, and introduce the access modes set $modes = \{read, write, execute\}$ and refine messages $m \in M$ to be of the

form $(c, op, x_1, \ldots, x_n)$, so that the set of *commands*, each of which is expressed in the Seaview instance as $op(v_1, i, x_1, \ldots, x_n \rightarrow v_2)$ is expressed here by defining the execution function E as: for all $i \in U$, $E(v_1, m) = v_2$ where $m = (channel(i), op, x_1, \ldots, x_n)$.

The set of *command sequences* in the Seaview instance is expressed as the set T_c of traces that can appear on a channel. A trace t is in T_c if and only if for all messages $m, m' \epsilon t, m.c = m'.c$. The notation $m.c$ selects the component c of the message m.

Each object o has an associated secrecy level $objseclevel(o)$ and integrity level $objintlevel(o)$.

$$objseclevel: O \rightarrow L_s$$
$$objintlevel: O \rightarrow L_i$$

We specify *seclevel* and *intlevel* as follows:

$$seclevel(\phi) = objseclevel(obj(\phi))$$
$$intlevel(\phi) = objintlevel(obj(\phi))$$

The key to capturing flows due to writing lies in the equivalence relations κ_i. The definition for Seaview specifies that states returning exactly the same set of object values appear equivalent to a subject. However, consider two states v_1 and v_2 that are identical, except possibly in the value of object o. Call these two values x_1 and x_2. Suppose that subject i writes x_3 into o, moving from v_1 to v_3, and some other subject j writes x_4 into v, moving from v_2 to v_4, where $x_3 = x_4$. By the state-based definition of κ_i, v_3 and v_4 are equivalent. However, i certainly perceives that they are different; in one case i computed the value of o and in the other case i sees that some other subject computed the value of o.

From this example one can see that two states can be equivalent to subject i if they both result only from actions of i. (We leave open the question as to whether or not two states can be equivalent to subject i if they both result only from actions of some other subjects.) Any pair of states not satisfying this condition cannot be equivalent, regardless of the values they return. So we revise the equivalence relations as follows:

for all $v', v'' \in S, i \in U$,
  $(v', v'') \in \kappa_i$ iff $samevalues(i, v', v'')$ and for all $t', t'' \in T$,
  if $apply(E, t', v_0) = v'$ and $apply(E, t'', v_0) = v''$
  then $sameview(i, t', t'')$

where

$samevalues(i, v', v'')$ iff for all $o \in O, v'(o, i) = v''(o, i)$

$sameview\ (i, t', t'')$ iff for all $m' \epsilon t', m'' \epsilon t'', m'.c = m''.c = i$.

The following functions are specified:

$$\begin{aligned} \textit{secrecycomponent}&: L \rightarrow L_s \\ \textit{integritycomponent}&: L \rightarrow L_i \\ \textit{readlevel}&: U \rightarrow L \\ \textit{writelevel}&: U \rightarrow L \\ \textit{objlevel}&: O \rightarrow L \\ \textit{currentaccessset}&: S \times U \times O \rightarrow \textit{modes}^*, \end{aligned}$$

and we define

$$\begin{aligned} \textit{objseclevel}(o) &= \textit{secrecycomponent}(\textit{objlevel}(o)) \\ \textit{objintlevel}(o) &= \textit{integritycomponent}(\textit{objlevel}(o)) \end{aligned}$$

Seaview includes trusted subjects, defined by

$$\textit{trusted}(i) \text{ iff } \textit{writelevel}(i) \prec \textit{readlevel}(i)^6$$

However, as we shall see, to satisfy the multilevel combined policy, it will be necessary to disallow trusted subjects with the specification: for all $i \in U$, $\textit{readlevel}(i) \preceq \textit{writelevel}(i)$.

We specify *seclevel* and *intlevel* for subjects in Seaview as follows:

$$\begin{aligned} \textit{seclevel}(i) &= \textit{secrecycomponent}(\textit{readlevel}(i)) \\ \textit{intlevel}(i) &= \textit{integritycomponent}(\textit{readlevel}(i)) \end{aligned}$$

A word of justification is needed here to explain why both subject levels, secrecy and integrity, are derived from a subject's Seaview *readlevel.* As noted previously, integrity in Seaview is similar to secrecy; a subject can read an object if its secrecy level dominates that of the object, and a subject can read an object if its integrity level dominates that of the object.

The following *Read property* characterizes the *read* and *execute* access rights and restricts the state functions to return to all subjects the same value for an object.[7]

Read property

for all $i, j \in U, o \in O, v \in S$,
  if $v(o, i) \neq \textit{null}$ then ($\textit{read} \in \textit{currentaccessset}(v, i, o)$ or
    $\textit{execute} \in \textit{currentaccessset}(v, i, o)$)
  and
  if ($v(o, i) \neq \textit{null}$ and $v(o, j) \neq \textit{null}$) then $v(o, i) = v(o, j)$

[6] $\prec$ means $\preceq$ but not equal.

[7] The Seaview model does not characterize *read* and *execute*, but this seems to be required to show correctness. The Seaview property of returning the same value to different subjects does not appear to be needed for correctness.

and *objcontents* is defined by

for all $i \in U, o \in O, v \in S$,
  if $v(o, i) \neq null$ then $objcontents(v, o) = v(o, i)$

The following *Write property* characterizes the *write* access right.[8]

Write property
  for all $i \in U, o \in O, v_1, v_2 \in S$,
    if $E(v_1, m) = v_2$ and $m.c = i$ and $v_1(o, i) \neq v_2(o, i)$
      then $write \in currentaccessset(v_1, i, o)$

A state in Seaview is *secure* if and only if (a) the levels of all objects that are readable by a subject i have a level that is dominated by the read level of i; (b) the levels of all objects that are writable by a subject i have a level that dominates the write level of i; (c) the levels of all objects that are executable by a subject i have a secrecy level that is dominated by the secrecy part of the read level of i and have an integrity level that dominates the integrity part of the write level of i; and (d) the Read property holds.[9]

Definition 6.4

A state v in Seaview is *secure* iff for all $i \in U, o \in O$,
  if $read \in currectaccessset(v, i, o)$ then $objlevel(o) \preceq readlevel(i)$ and
  if $write \in currectaccessset(v, i, o)$ then $writelevel(i) \preceq objlevel(o)$ and
  if $execute \in currectaccessset(v, i, o)$
      then $objseclevel(o) \preceq secrecycomponent(readlevel(i))$
      and $integritycomponent(writelevel(i)) \preceq objintlevel(o)$ and
  the Read property holds for v

A command in Seaview is *secure* if and only if it always takes a secure state into a secure state and the Write property holds.

Definition 6.5.

A command m in Seaview is *secure* iff for all $i \in U, o \in O, v_1, v_2 \in S$,
  if v_1 is secure then $E(v_1, m)$ is secure and
  the Write property holds for v_1 and v_2.

Finally, the major constraint of the Seaview mandatory policy instance is:

Definition 6.6. An execution function E is *Seaview constrained* iff the initial state is secure and all commands are secure.

The Seaview mandatory policy instance is richer than the previous state-based instance in four major ways. First, it addresses integrity by introducing the lattice L_i. Second, it specifically addresses *program integrity* by introducing the *execute* access right and placing a condition on it in the definition of a secure state. Third, it rules out a spontaneous change to the value of an object (one not caused by an authorized subject) by adding the Write prop-

[8] The Seaview model calls this the Information Transition property.
[9] The Read property is implicit in Seaview.

erty. Fourth, it incorporates trusted subjects. This last component of the model means that correctness cannot be proved if any trusted subjects are present (at least not without proving certain properties of all the trusted subjects).

There is one further aspect of Seaview that precludes it from being a model for SL. The closure conditions on the permission and obligation sets require that if $\phi \in \Theta_i$ then $\phi \in \Phi_i$, which is equivalent to: if $intlevel(i) \preceq intlevel(\phi)$ then $seclevel(\phi) \preceq seclevel(i)$. Seaview includes no such restriction; it is quite allowable to specify an object that has a higher integrity than some subject, and is therefore obligated to be known, but which is not permitted to that subject. This of course is due to the restricted notion of integrity in Seaview. Consequently, in order to show correctness, we need to assume this closure condition for our formulation of Seaview.

THEOREM 5. *The Seaview mandatory policy instance is correct with respect to the Multilevel Combined policy if no subjects are trusted and if the closure conditions are satisfied.*

PROOF. Consider any subject i, any state v, and any formula $\phi = (o, x)$. There are two cases: one for secrecy and one for integrity. We take secrecy first. Assume that $not(\phi \in \Phi_i)$. We need to show that the subject i cannot know ϕ. That is, from the required Seaview property, we must show that

$$\text{there exists } v', (v, v') \in \kappa_i \quad \text{and} \quad v' \models \neg \phi. \tag{11}$$

By Multilevel Combined,

$$seclevel(\phi) \not\preceq seclevel(i). \tag{12}$$

From the definitions of *seclevel* and *objseclevel*,

$$secrecycomponent(objlevel(o)) \not\preceq secrecycomponent(readlevel(i)) \tag{13}$$

so that

$$objlevel(o)) \not\preceq readlevel(i). \tag{14}$$

By Definition 6.6, v_0 is secure and all commands are secure. By Definition 6.5, all states are secure, so v is secure. By Definition 6.4 and (14), $read \notin currentaccessset(v, i, o)$ and $execute \notin currentaccessset(v, i, o)$. By Definition 6.4 and the Read property, $v(o, i) = null$.

Now we derive the bounds imposed by Seaview on $objlevel(o'')$ for any object o'' not accessible to i. Consider all subjects j that read o'' in some state v'', that is $read \in currentaccessset(v'', j, o'')$. By Definition 6.4, $objlevel(o'') \preceq readlevel(j)$.

Since there are no trusted processes, $readlevel(j) \preceq writelevel(j)$, so that $objlevel(o'') \preceq writelevel(j)$.

But $objlevel(o'')$ is also bounded by subjects k that write to o''. Consider all subjects k that write to o in v'', that is, $write \in currentaccessset(v'', k, o'')$. (Execute does not appear to play a role nor to conflict here.) By Definition 6.4, $writelevel(k) \preceq objlevel(o'')$.

Again, since there are no trusted processes, $readlevel(k) \preceq writelevel(k)$, so that $readlevel(k) \preceq objlevel(o'') \preceq writelevel(j)$.
More precisely, we define

$$maxlevel(v, o'') = \operatorname{lub}_{k \in U} \{readlevel(k) \text{ such that } write \in currentaccessset(v, k, o'')\}$$

and

$$minlevel(v, o'') = \operatorname{glb}_{j \in U} \{writelevel(j) \text{ such that } read \in currentaccessset(v, j, o'')\}.$$

Therefore, for each state v'', $maxlevel(v'', o'') \preceq objlevel(o'') \preceq minlevel(v'', o'')$, and for all states $v'' \in S$,

$$\operatorname{lub}_{v'' \in S} \{maxlevel(v'', o)\} \preceq objlevel(o'') \preceq \operatorname{glb}_{v'' \in S} \{minlevel(v'', o'')\}. \quad (15)$$

Now consider any trace $t \in T$ such that $apply(E, t, v_0) = v$. It is always possible that i writes to any o'' in t, say in some state v_k. In other words, $write \in currentaccessset(v_k, i, o'')$. This is justified as follows. By Definition 6.4, such writing requires that $writelevel(i) \preceq objlevel(o'')$.
At the same time, (15) requires that

$$\operatorname{lub}_{k \in U} \{readlevel(k) \text{ such that } write \in currentaccessset(v_k, k, o'')\} \preceq objlevel(o'')$$

Since i is one of the k's, $readlevel(i) \preceq objlevel(o'')$. Since i is not trusted, $readlevel(i) \preceq writelevel(i)$, so $readlevel(i) \preceq objlevel(o'')$, and no violation of the constraints occurs.

Now construct an initial state v_0 and a trace $t \in T$ such that $apply(E, t, v_0) = v$ and for all commands $m \epsilon t$, $m.c = i$ and for all accessible objects o''', $v_0(o''', i) = v(o''', i)$ and for all inaccessible objects o'''', i writes $v(o, i)$ to o''''.

Now take v' to be a state identical to v except that $v'(o, j) = null$ for all j and take t' to be the trace identical to t. Then, $(v, v') \in \kappa_i$, and by the definition of π, $v' \vDash \neg \phi$. Consequently, (11) holds.

Now take the integrity case, which is similar. Assume that $not(\phi \in \Theta_i)$. We need to show that the subject i cannot know ϕ. That is, from the required Seaview property, we must also show that (11) holds. By Multilevel Combined,

$$intlevel(i) \npreceq intlevel(\phi). \quad (16)$$

From the definitions of *intlevel* and *objintlevel*,

$$integritycomponent(readlevel(i)) \npreceq integritycomponent(objlevel(o)) \quad (17)$$

so that

$$objlevel(o) \npreceq readlevel(i). \quad (18)$$

The remainder of the proof for integrity is identical to that for secrecy. □

In a subsequent paper we present a discretionary access control policy to capture the essential notions of the Seaview discretionary policy instance. A *Compatibility property* for this policy with respect to the Multilevel Combined policy is derived. We also define a simple version of the Seaview discretionary policy instance, which we show is correct with respect to our discretionary policy. This proof includes a proof of the Compatibility property as well as the Seaview property.

7. DISCUSSION

We have described SL, a logic for specifying and reasoning about secure distributed systems. The expressiveness of this logic is a result of combining modalities for knowledge and time with modalities for permission and obligation. All of these modalities can be expressed using a Kripke possible-worlds semantics.

Notions of security can be expressed as both safety and liveness properties. We have illustrated how secrecy can be expressed as safety properties and integrity expressed as liveness properties. SL provides a language that can express both of these concepts. Thus we provide a formalism in which secrecy, integrity, and the interaction of these properties can be expressed.

It could be argued that anything that we specify using modalities can also be expressed in a first-order predicate logic. The advantage of using the possible-worlds approach is that we can specify abstract properties of security, without considering the details of an implementation of a policy. For example, we are able to express a fundamental notion of secrecy, *that if a subject is not permitted to know something then it will in fact not know it*, without explicitly stating how permission is defined.

In this paper we have concentrated on security in asynchronous distributed systems. We have previously illustrated that a possible-worlds model can also be interpreted for security in databases [26]. When considering databases, the possible worlds correspond to possible states of the database.

Applications of logics for knowledge in security have also been considered by other researchers. This related research has concentrated on the use of knowledge and belief to analyze authentication [3, 6] and decryption-based key distribution protocols [3, 38] and defining trust [44]. More investigation is needed into the relationship between this work and ours.

Another aspect of the logic that needs investigation is a formal proof system for verifying implementations of a particular policy. Previously, we defined a secure system, SNet [13], that was specified using operator nets [1]. An operational semantics for operator nets has been defined that can be used to express communication in a distributed system [18]. This model provides a formalism for defining the set of valid states (possible worlds) of a system. A formal deduction system will allow us to verify that an operator net is a correct implementation of a given security policy.

The relationship between information flow and knowledge is a complex problem that we only begin to address in this paper. As illustrated in the previous section, the set of potential input and output traces of a system can

correspond to the set of primitive propositions. What we need is a formal method of determining what a subject knows in a particular state. Our possible-worlds model defines this to be that which is true in all states that are indistinguishable to the subject. This implies that the knowledge of a subject depends only on the local state for the process.

When considering encrypted messages, the relationship between messages and knowledge is even more complex. For a process to know the information in the message, it may also be necessary to know an encryption key. As well, a process may deduce new information through some pattern in timing and repetition of messages. Thus, it is obvious that the relationship among communication, information flow, and knowledge is a problem that needs further study.

REFERENCES

1. ASHCROFT, E. A., AND JAGANNATHAN, R. Operator nets. In *IFIP TC-10 Conference on Fifth Generation Computer Architectures.* North-Holland, Amsterdam, 1985.
2. BEN-ARI, M., MANNA, Z., AND PNUELI, A. The temporal logic of branching time. In *8th Annual ACM Symposium on Principles of Programming Languages*, ACM, New York, 1981, 164–176.
3. BURROWS, M., ABADI, M., AND NEEDHAM, R. A logic of authentication. *ACM Trans. Comput. Syst. 8,* 1 (1990), 18–36.
4. BIBA, K. J. Integrity considerations for secure computer systems. Tech. Rep. TR-3153, MITRE Corp., April 1977.
5. BELL, D. E., AND LAPADULA, L. J. Secure computer systems: Mathematical foundations and model. Tech. Rep., MITRE Corp., Bedford, Mass., 1974.
6. BURROWS, M., NEEDHAM, R., AND ABADI, M. Authentication: A practical study in belief and action. In *Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning About Knowledge* (Pacific Grove, Calif., 1988). ACM, New York, 1988, 325–342.
7. COMMUNICATIONS SECURITY ESTABLISHMENT, DND. Canadian Trusted Computer Product Evaluation Criteria Workshop (Ottawa, Ont., Aug. 1988).
8. CLARK, D. D., AND WILSON, D. R. A comparison of commercial and military computer security policies. In *Proceedings of IEEE Symposium on Security and Privacy* (Oakland, Calif., April 1987). IEEE, New York, 1987, 184–194.
9. DENNING, D. E. A lattice model of secure information flow. *Commun. ACM 19*, 5 (May 1976), 236–243.
10. EMERSON, E. A., AND HALPERN, J. Y. "Sometimes" and "not never" revisited. In *9th Annual ACM Symposium on Principles of Programming Languages.* ACM, New York, 1982, 127–139.
11. GASSER, M. *Building a Secure Computer System.* Van Nostrand, New York, 1988.
12. GOGUEN, J. A., AND MESEGUER, J. Security policies and security models. In *IEEE Symposium on Security and Privacy* (Oakland, Calif., April 1982). IEEE, New York, 1982, 11–20.
13. GLASGOW, J. I., AND MACEWEN, G. H. The development and proof of a formal specification for a multilevel secure system. *ACM Trans. Comput. Syst. 5*, 2 (May 1987), 151–184.
14. GLASGOW, J. I., AND MACEWEN, G. H. Reasoning about knowledge in multilevel secure distributed systems. In *Proceedings of the IEEE Symposium on Security and Privacy* (Oakland, Calif., 1988). IEEE, New York, 1988, 122–128.
15. GLASGOW, J. I., AND MACEWEN, G. H. Obligation as the basis of integrity specification. In *Proceedings of the IEEE Symposium on Security and Privacy* (Oakland, Calif.,), IEEE, New York, 1982–91.
16. GLASGOW, J. I., AND MACEWEN, G. H. Obligation as the basis of integrity specification. In *Proceedings of the Computer Security Foundations Workshop* (Franconia, N.H.). MITRE Corp., 1988–91.

17. GLASGOW, J. I., AND MACEWEN, G. H. Obligation as the basis of integrity specification. In *Proceedings of the Computer Security Foundations Workshop* (Franconia, N. H., June 1989). MITRE Corp., 1989, 64–70.
18. GLASGOW, J. I., AND MACEWEN, G. H. An operator net model for distributed systems. *J. Distributed Syst. 3,* 4 (Sept. 1989), 196–209.
19. GLASGOW, J. I., AND MACEWEN, G. H. A logic for reasoning about security. In *Proceedings of the Computer Security Foundations Workshop* (Franconia, N.H., June 1990). IEEE, New York, 1990.
20. GLASGOW, J. I., MACEWEN, G. H., AND PANANGADEN, P. Reasoning about knowledge and permission in secure distributed systems. In *Proceedings of the Computer Security Foundations Workshop* (Franconia, N.H., 1988). IEEE, New York, 1988, 139–146. Also appears in *IEEE Cipher.*
21. GLASGOW, J. I., MACEWEN, G. H., AND PANANGADEN, P. Security by permission in databases. In *Database Security II: Status and Prospects.* Elsevier North-Holland, Amsterdam, 1988, 197–205.
22. GUTTMAN, J. D., AND NADEL, M. E. What needs security? In *Proceedings of the Computer Security Foundations Workshop* (Franconia, N.H., June 1988), 34–57.
23. HALPERN, J. Reasoning about knowledge. In *Annual Reviews of Computer Science,* Annual Reviews Inc., 1987, 21–35.
24. HINTAKKA, J. *Knowledge and Belief.* Cornell University Press, 1962.
25. HALPERN, J. Y., AND MOSES, Y. Knowledge and common knowledge in a distributed environment. In *Proceedings of the 3rd ACM Conference on Distributed Computing.* 1984, 50–61.
26. JOHNSON, D. M., AND THAYER, F. J. Stating security requirements with tolerable sets. *ACM Trans. Comput. Syst. 6,* 3 (Aug. 1988), 284–295.
27. JOHNSON, D. M., AND THAYER, F. J. Security and the composition of machines. In *Proceedings of the Computer Security Foundations Workshop* (Franconia, N.H., June 1988), 72–89.
28. KRIPKE, S. Semantical considerations of modal logic. *Acta Philosophica Fennica 16* (1963), 83–94.
29. LAMPORT, L. Sometimes is sometimes not never. In *Proceedings of 7th Annual ACM Symposium on Principles of Programming Languages,* ACM, New York, 1980, 174–185.
30. LANDWEHR, C. E. Formal models for computer security. *ACM Comput. Surv. 13,* 3 (Sept. 1981), 247–278.
31. LUNT, T. F. ET AL. The Seaview formal security policy model. Tech. Rep., SRI International, Computer Science Lab., Menlo Park, Calif, Feb. 1989.
32. LUNT, T. F. ET AL. The Seaview security model. *IEEE Trans. Softw. Eng. SE-16,* 6 (June 1990).
33. MCCULLOUGH, A. D. Specifications for multi-level security and a hook-up property. In *Proceedings of IEEE Symposium on Security and Privacy* (Oakland, Calif., April 1987), IEEE, New York, 1987, 161–166.
34. MCCULLOUGH, A. D. Covert channels and degrees of insecurity. In *Proceedings of the Computer Security Foundations Workshop* (Franconia, N.H., June 1988), 1–33.
35. MCCULLOUGH, A. D. Non-interference and the composability of security properties. In *Proceedings of IEEE Symposium on Security and Privacy* (Oakland, Calif., April 1988), IEEE, New York, 1988, 177–186.
36. MCLEAN, J. Reasoning about security models. In *Proceedings of IEEE Symposium on Security and Privacy* (Oakland, Calif., April 1987), IEEE, New York, 1987, 123–131.
37. MCLEAN, J. The algebra of security. In *Proceedings of IEEE Symposium on Security and Privacy* (Oakland, Calif., April 1988), IEEE, New York, 1988, 2–7.
38. MOSER, L. A logic of knowledge and belief for reasoning about computer security. In *Proceedings of the Computer Security Foundations Workshop II* (Franconia, N.H., 1989), 57–63.
39. MACEWEN, G. H., POON, V. W. W., AND GLASGOW, J. I. A model for multilevel security based on operator nets. In *Proceedings of IEEE Symposium on Security and Privacy* (Oakland, Calif., April 1987), IEEE, New York, 1987, 150–160.
40. NATIONAL COMPUTER SECURITY CENTER. *Workshop on Integrity Policy in Computer Information Systems* (Waltham, Mass., Oct. 1987).

41. *Proceedings of the National Computer Security Conference,* Dept. of Defense, 1984–88.
42. PNUELI, A. The temporal logic of concurrent programs. *Theor. Comput. Sci. 13* (1981), 45–60.
43. PRIOR, A. *Time and Modality.* Oxford University Press, 1957.
44. RANGAN, P. V. An axiomatic basis of trust in distributed systems. In *Proceedings of the 1988 IEEE Computer Society Symposium on Security and Privacy* (Oakland, Calif., 1988), 204–211.
45. RUSHBY, J. M. Design and verification of secure systems. *ACM SIGOPS 15*, 5 (Dec. 1981), 12–21.
46. SUTHERLAND, A. D. A model of information. In *Proceedings of 9th National Computer Security Conference* (Gaithersburg, Md., Sept. 1986), Dept. of Defense, 175–183.

Received September 1990; revised October 1991; accepted March 1992