The SeaView Security Model

TERESA F. LUNT, MEMBER, IEEE, DOROTHY E. DENNING, ROGER R. SCHELL, MEMBER, IEEE, MARK HECKMAN, AND WILLIAM R. SHOCKLEY

Abstract—**A multilevel database is intended to provide the security needed for database systems that contain data at a variety of classifications and serve a set of users having different clearances. This paper describes a formal security model for a such a system. The model is formulated in two layers, one corresponding to a reference monitor that enforces mandatory security, and the second an extension of the standard relational model, defining multilevel relations and formalizing policies for labeling new and derived data, data consistency, and discretionary security. The model also defines application-independent properties for entity integrity, referential integrity, and polyinstantiation integrity.**

Index Terms—**Classification, database security, multilevel security, protection, relational databases, security, security model.**

I. INTRODUCTION

MANY civilian, defense, and commercial applications require a *multilevel database system* that supports data having different *access classes* (security markings) and users with different authorizations, or *clearances*. This paper presents a formal security policy model for a secure multilevel relational database system. The model was developed for the SeaView project, which was a three-year joint effort by SRI International (SRI) and Gemini Computers, Inc., sponsored by the U.S. Air Force, Rome Air Development Center (RADC). SeaView's goal was to design a multilevel secure database system that meets the criteria for Class A1 of the *DoD Trusted Computer System Evaluation Criteria* (DoD 5200.28-STD) [1]. The model formalizes the SeaView security policy [2] and was the foundation for SeaView's formal top-level specifications (FTLS) [3] and design specifications. The SeaView project has also defined a multilevel data manipulation and control language we call MSQL [4], for multilevel SQL.[1] In addition, we have produced a formal top-level specification of the MSQL commands [3] using the formal specification language of the EHDM formal verification system [5] and have partially verified that the MSQL commands conform to the properties of the SeaView model [6]. We have also built a demonstration system that illustrates key aspects of the model and design.

A. Multilevel Security

The concern for multilevel security arises when a computer system contains information with a variety of classifications and has some users who are not cleared for the highest classification of data contained in the system.

A security classification, or *access class*, consists of a hierarchical sensitivity level (e.g., TOP-SECRET, SECRET, CONFIDENTIAL, UNCLASSIFIED, etc.) and a set of nonhierarchical categories. In order for a user to be granted access to information, the user must be cleared for the sensitivity level as well as for each of the categories in the information's access class. The sensitivity levels are linearly ordered. The categories do not have such a linear ordering. However, the set of access classes (<sensitivity level, category set> pairs) is partially ordered and forms a lattice [7]. The partial ordering relation is called the *dominance* relation. Access class A dominates access B if the sensitivity level of A is greater or equal to the sensitivity level of B and the security categories of A include all those of B.

The DoD policies restricting access to classified information to cleared personnel are called *mandatory security*. Mandatory security requires that classified data be protected not only from direct access by unauthorized users, but also from disclosure through indirect means, such as covert signaling channels. Covert channels are information channels that were not designed to be used for information flow but can nevertheless be exploited by malicious software to signal high data to low users.[2] For example, a high process (i.e., a program instance having a high clearance because it is acting on behalf of a high user) may use read and write locks observable to a low process over time to encode high information (e.g., locked = 1, unlocked = 0). Mandatory security requires that no information can flow from high access classes to low.

A *trusted subject* is a subject (i.e, executing program) that is allowed to read and write within a range of access classes. This range defines a sublattice in the access class lattice. Trusted subjects are analogous to cleared users, in

Manuscript received August 1, 1989; revised January 29, 1990. Recommended by T. A. Berson and S. B. Lipner. This work was supported by the U.S. Air Force, Rome Air Development Center, under Contract F30602-85-C-0243.

T. F. Lunt is with the Computer Science Laboratory, SRI International, Menlo Park, CA 94025.

D. E. Denning is with the Systems Research Center, Digital Equipment Corp., Palo Alto, CA 94301.

R. R. Schell and M. Heckman are with Gemini Computers, Inc., Carmel, CA 93922.

W. R. Shockley is with Digital Equipment Corp., Mountain View, CA 94040.

IEEE Log Number 9034814.

[1]MSQL is an extension of SQL (Structured Query Language) that includes a built-in type *access class* to refer to data classifications, as well as operators for data of that class. MSQL allows users to retrieve and manipulate data based on their classifications.

[2]For simplicity, we are using the terms "high" and "low" to refer to any two access classes when the second does not dominate the first.

that they are "trusted" to write data at an access class that may be lower than that of some of the data they read without violating security. In order for a program to be designated as trusted, it must be analyzed to show that it does not convey information downward in access class.

Other access controls may be imposed in addition to mandatory security; these enforce *discretionary security*. The access controls commonly found in most database systems are examples of discretionary access controls.

The "trust" in trusted computer systems rests on the ability to provide convincing arguments or proofs that the security mechanisms work as advertised and cannot be disabled or subverted. The concept of a *reference monitor* was developed so as to be able to demonstrate a system's trustworthiness. Traditional security engineering practice is to segregate the security-critical functions in a reference monitor.

We assign access classes to subjects derived from the clearance of the user on whose behalf the subject is operating. The reference monitor mediates each reference to an object by any subject, allowing or denying the access based on a comparison of the access classes associated with the subject and with the object. The reference monitor must be tamperproof; it must be invoked for every reference; and it must be small enough to be verified to be correct and secure with respect to the policy it enforces. A high degree of assurance must be provided not only that the mandatory security mechanisms control access to sensitive information, but also that they enforce secure information flow. The reference monitor forms the core of the trusted computing base (TCB), which contains all security-relevant code. The *DoD Trusted Computing System Evaluation Criteria* include requirements for "minimizing the complexity of the TCB, and excluding from the TCB modules that are not protection-critical" [1], so that the reference monitor is "small enough to be verifiable" [1]. Without such a requirement, the high degree of assurance required would not be feasible.

B. The SeaView Design

In pursuit of Class A1 assurance [1], in SeaView we have adopted a design approach that is built on the notion of a reference monitor for mandatory security [8], [9].

SeaView provides the user with the basic abstraction of a multilevel relation in which the individual data elements can be individually classified. Our design approach implements multilevel relations as views over stored single-level[3] relations, transparent to the user. The single-level relations are stored in segments (of the corresponding access class) managed by an underlying mandatory reference monitor. This underlying mandatory reference monitor performs a label comparison for subjects and the segments for which they request access, to decide whether to grant access. The access class of any particular data element in a multilevel relation is derived from the access class of the single-level relation in which the data element is stored, which in turn matches the access class of the segment in which it is stored, which is known to the reference monitor, Thus, labels for each individual data element do not have to be stored, as was supposed prior to SeaView.

Implementing multilevel relations as views allows insert, delete, and update operations on the multilevel relations to be translated into corresponding operations on the single-level stored relations. Thus, our design approach is able to use the nucleus of a commercially available relational database management system to manage the single-level relations, with an added layer of software to create the abstraction of multilevel relations.

In SeaView, every database function is carried out by a single-level subject. Thus, a database system subject, when operating on behalf of a user, cannot gain access to any data whose classification is not dominated by the user's clearance. The use of only single-level subjects for routine database operations provides the greatest degree of security possible and considerably reduces the risk of disclosure of sensitive data.

This approach means that there must be at least one database server instance for each active access class (an access class is considered to be active if a subject of that class is active). Thus, the database system consists of multiple database server instances that share the same logical database.

C. Related Work

The SeaView security model allows the individual data elements within a relation to be individually classified. Several previous research efforts have proposed security models for multilevel databases. The earliest of these were the Hinke/Schaefer model [10], which supports classification at the attribute level, and the I.P. Sharp model [11], which supports classification at the relation level. More recently, the TRW model [12] was developed to support tuple-level classification. The Navy surveillance model [13] supports multilevel relations, but does so by treating entities such as relations as containers of data, rather than as identifiers, as does SeaView. The LOCK Data Views model [14] supports element-level classification but is designed for the LOCK special-purpose architecture [15].

Designs have been developed for systems based on these models. These systems differ in the amount of trusted code that is required, in whether they run on an underlying trusted operating system, and in the extent to which they make use of the trusted operating system's security mechanisms. For example, SeaView makes use of an underlying reference monitor to enforce mandatory security, but performs at least part of the enforcement of the discretionary security policy in the database system (views can be used to enforce discretionary security), whereas the Hinke/Schaefer design relied on the underlying trusted operating system for enforcement of both mandatory and discretionary security. This latter approach allows for simpler and hence more trustworthy discretionary controls, but rules out the common use of views to enforce

[3] *Single-level* means having a single access class.

discretionary security. The TRW design enforces both mandatory and discretionary security within the database system itself, hence requiring more trusted code and duplication of some of the security functionality of the operating system. The need for larger amounts of trusted code means that a significantly greater effort must be devoted to formal analysis of the trusted code, and, to the extent that the analysis is not complete or discovers channels that cannot be closed, introduces more security vulnerabilities.

II. Model Overview

The SeaView security policy consists of a mandatory access control policy, a discretionary access control policy, and supporting policies for labeling new and derived data, data consistency, sanitization, and reclassification. The supporting policies address requirements that are security-relevant but not part of access control.

The SeaView model is formulated in two layers, an inner layer called the *MAC model*[4] and an outer layer called the *TCB model*.

The MAC model defines the mandatory security policy and represents a reference monitor that meets the criteria for Class A1. It includes the concepts of subjects, objects, and current access set from the Bell and LaPadula security model [16], but contains no components that are specific to database systems. It is intended to model a general-purpose computing base that supports a wide range of applications. The SeaView design uses the commercially available GEMSOS TCB [17] as its reference monitor.

The MAC model assigns two access classes to each subject S: *read-class*(S) and *write-class*(S) (these two classes are equal for untrusted subjects), where *read-class*(S) $\geq$ *write-class*(S). The access requirements are formalized by the following two rules:

1) A subject S can read data of access class c only if *read-class*(S) $\geq c$, and

2) A subject S can write data of access class c only if *write-class*(S) $\leq c$.

The MAC model includes *tranquility* (i.e., access classes for subjects and objects are state-independent) [16]. The model also includes an information component, which represents the contents of objects, and a *program integrity property*, which is intended to prevent low integrity software from executing with the privileges of high integrity software, by preventing high integrity software from passing control to low integrity software. These concepts are discussed more fully in [18].

The TCB model defines the discretionary access control policy and the supporting policies. It specifies the components of a multilevel secure relational database system, including multilevel relations, views, integrity constraints (including classification constraints on access classes), and discretionary authorizations. Because the information implementing the TCB model is to be stored in objects mediated by the reference monitor, the TCB model layer is constrained by mandatory security.

Each model layer is defined by the following components: 1) a set of *types*, together with *functions* on these types. *Subjects* (active entities) are modeled as a type SUBJECTS. *States* are modeled as a type STATES. One state, denoted s_0, is designated an *initial state*. *State-dependent* functions have a state variable as a formal parameter. *State-independent* functions do not change in value from state to state; 2) a set of *commands* of the form $op(s_1, S, x_1, \cdots, x_n \rightarrow s_2)$, where S is a subject and $x_1, \cdots, x_n$ are other parameters. Each abstract command represents an atomic action that causes a state transition from state s_1 to state s_2; 3) a set of *axioms*, which are the assumptions of the model; and 4) a set of *properties*, which represent the formal policy statement. The four types of properties are *type properties*, which apply to state-independent functions, *state properties*, which apply to state-dependent functions, *transition properties*, which apply to commands, and *command sequence properties*, which apply to sequences of commands.

A state is *secure* if and only if it satisfies all state properties. A command $op(s_1, S, x_1, \cdots, x_n \rightarrow s_2)$ is *secure* if and only if, for all subjects S and parameters $x_1, \cdots, x_n$, it satisfies all transitions properties, s_2 is secure whenever state s_1 is secure, and for any command sequence satisfying the command sequence properties, the sequence $\alpha \circ op$ satisfies the command sequence properties.[5] A state s is *reachable* either if it is the initial state s_0 or if it results from a sequence of commands:

$$op_0(s_0, S, x_1, \cdots, x_n \rightarrow s_1), \cdots, op_k(s_k, S, y_1, \cdots, y_m \rightarrow s).$$

A system is *secure* if and only if all axioms and type properties are satisfied, the initial state s_0 is secure, and all reachable states other than s_0 result from a sequence of secure commands.

We can now state the following basic security theorem.

Theorem 1 (Basic Security Theorem): If a system satisfies all axioms and all type properties, the initial state is secure, and all commands are secure, then the system is secure.

We have written formal specifications for SeaView. These specifications formalize the SeaView security model, and define a set of objects (tuples, relations, views, databases), a set of functions on those objects, and the set of SeaView security properties enumerated in the model. The formal specifications also specify the functionality of the SeaView MSQL interface by defining operations for manipulating multilevel relations [6].

Because the functional specification of MSQL was designed to provide a foundation for a subsequent design and implementation effort, it is extremely important that the operations as specified satisfy the SeaView security properties. To verify that this is the case, we constructed proofs for several of the operations. The verification at-

[4]Mandatory Access Control.

[5]We use the symbol "$\circ$" to denote concatenation.

tempts to show that the system is secure by asserting the axioms and type properties, asserting that the initial state is secure, and demonstrating that all commands are secure. Thus, for each specified operation, we must show that it satisfies the transition properties with respect to its starting and concluding states, and that if it starts in a secure state, it results in a secure state.

The command sequence properties are used to model transaction properties, such as atomicity, serializability, and permanence. These are formalized in [18].

III. Multilevel Relations

The SeaView model supports classifications at the granularity of individual atomic facts through element-level classification. In addition, the model assigns a classification to each tuple, which represents the access class of the information in (or encoded in) the tuple.

A. Multilevel Tuples

The following types and functions define multilevel tuples:

- type MTUPLES of *multilevel relation tuples*.
- type DATUM of *element values*.
- *mlength:* MTUPLES → $\mathfrak{N}$, which gives the number of data values in a tuple.[6]
- *melement-value:* MTUPLES × $\mathfrak{N}$ → DATUM, which gives the ith data element in a multilevel tuple.
- *melement-class:* MTUPLES × $\mathfrak{N}$ → CLASSES, which gives the access class of the ith data element in a multilevel tuple.
- *mtuple-class:* MTUPLES → CLASSES, which gives the tuple class of a multilevel tuple.

Two tuples are equal if all of the above components are the same.

Because the tuple class represents the class of the information in (or encoded in) a tuple, it must dominate all element classes within the tuple, as follows.

Property 1 (Tuple Class Property): $\forall r \in$ MTUPLES and $\forall i$, $1 \leq i \leq$ mlength(r):

$$\text{melement-class}(r, i) \leq \text{mtuple-class}(r).$$

□

Multilevel relations are sets of multilevel tuples:

- type MRELATION-VALUES $\subset \mathcal{P}$(MTUPLES) of *multilevel relation values*.[7]

The relation value corresponding to the empty tuple set $\varnothing$ is called the *null relation*.

B. Named Multilevel Relations

The SeaView model has three types of named relations: multilevel real relations, snapshots, and views. *Multilevel real relations* give the abstraction of stored relations and are subject to the three application-independent integrity rules: entity integrity, referential integrity, and polyinstantiation integrity. *Snapshots* correspond to stored relations that hold the intermediate and final results of computations; they are not subject to the application-independent integrity rules. *Views* are defined by multilevel relational expressions over the multilevel relations and are evaluated each time the view is used; view evaluation yields a derived multilevel relation. All three types of relations have a degree and an associated access class. The access class represents the class of the identifier (name) of the multilevel relation.

The three types of named relations are distinguished by the following three disjoint types of identifiers:

- type MREAL-IDS of *multilevel real relation identifiers*.
- *type MVIEW-IDS of multilevel virtual relation (view) identifiers*.
- type MSNAPSHOT-IDS of *multilevel snapshot identifiers*.

These three types are disjoint subtypes of the following "union" type:

- type MRELATION-IDS = MREAL-IDS ∪ MVIEW-IDS ∪ MSNAPSHOT-IDS of *multilevel relation identifiers*.

The multilevel relation identifiers are names for the "named objects" of the system, in the sense of the *Criteria* [1]. These multilevel relation identifiers are associated, in a given state s, with values of type MRELATION-VALUES. This association is formalized in the function mrelation-instance, defined in Section III-E.

Throughout the remainder of this paper, we use the words multilevel relation, relation, view, and snapshot to mean the identifier (name), rather than the instance.

The class and degree of a multilevel relation are defined as follows:

- *mrelation-class:* MRELATION-IDS → CLASSES, which gives the access class of the relation identifier (i.e., name).
- *mdegree:* MRELATION-IDS → $\mathfrak{N}$, which gives the *degree* of all multilevel relations that can be associated with the relation identifier (the length of all mtuples in the relation).

Each multilevel relation identifier R has associated with it n *attributes*, where n = mdegree(R). These attributes are identified by an index value between 1 and n.

SeaView requires that a relation schema or view definition have a single access class, which is the access class of the relation or view identifier. Although the model assigns a class to a relation identifier, it does not assign one to those functions that define the schema, for example, the function mdegree. Thus, the model leaves the enforcement of this requirement to the design. In the SeaView design, all schema information is stored at relation-low (i.e., at the lowest access class at which data may be stored in the relation) [8].

C. Keys

Multilevel real relations have primary keys and may have foreign keys. The primary key is intended to

[6] We use the notation $\mathfrak{N}$ to denote the set of integers ≥ 0.

[7] We use the notation $\mathcal{P}(X)$ to denote the powerset of the set X; that is, $\mathcal{P}(X)$ is the set of all subsets of X.

uniquely identify a tuple t in a relation. A foreign key is a reference to a tuple r in another (designated) relation for which the foreign key value matches t's primary key value. Primary and foreign keys are defined as follows.

- *primary-key:* MREAL-IDS → $\mathcal{P}(\mathfrak{N})$, which gives the set of attributes that define a real relation's primary key.
- *key-degree:* MREAL-IDS → $\mathfrak{N}$, which gives the number of attributes forming the primary key.
- *foreign-key-ref:* MREAL-IDS × $\mathfrak{N}$ → (MREAL-IDS × $\mathfrak{N}$), which maps an attribute of one real relation to an attribute of another. For example, if *foreign-key-ref*$(R, i) = (Q, j)$, the the ith data attribute of relation R is a reference to the jth data attribute of relation Q. An attribute is mapped to itself if it does not reference an external attribute.

From the above, the following functions are defined:

- *mtuple-key* MREAL-IDS × MTUPLES → MTUPLES, which, for a given mtuple associated with a real relation R, returns the primary key of the mtuple. The primary key is also an mtuple, and has length equal to key-degree(R). Its tuple class is the least upper bound of the classes of the elements that constitute the key. (In Section III-F-1 we will see that all elements of the key must have the same access class. Thus, the tuple class of the primary key tuple equals the key class.)
- *key-class:* MREAL-IDS × MTUPLES → CLASSES, which is the access class of the primary key, defined as:

$$key\text{-}class(R, r) = \text{mtuple-class}(\text{mtuple-key}\ (R, r)).$$

D. Databases

Databases are defined by the following type and function.

- type DATABASES of *database identifiers.*
- *database-class:* DATABASES → CLASSES, which gives the access class of the database identifier.

All multilevel real relations, views, and snapshots are associated with some database identifier.

- *mrelation-database:* MRELATION-IDS → DATABASES, which gives the database to which a multilevel relation belongs.

The SeaView policy requires that the access class of a relation identifier dominate the access class of the identifier of the database to which it belongs (because to access a relation, a subject must access the database).

Property 2 (Database Class Integrity): A system satisfies *database class integrity* if and only if $\forall R \in$ MRELATION-IDS:

mrelation-class$(R) \geq$
database-class(mrelation-database (R)).

□

E. Relation Instances

Subjects with different read-classes may retrieve data from the same multilevel relation, but will see different versions of the data. Thus, in any given state, each relation has potentially different instances at different access classes. The following function associates a relation instance with each relation identifier for a given state and access class.

- *mrelation-instance:* STATES × MRELATION-IDS × CLASSES → MRELATION-VALUES, which gives the multilevel relation instance at a given access class in a given state.

All tuples in each multilevel relation instance associated with identifier R must have length equal to the degree of R. (This is formalized in [18].)

The following property states that for tuples that appear in instances of multilevel real relations, the tuple class must be the least upper bound of the element classes. This is because the tuple class represents the access class of the existence of the tuple.

Property 3 (Real Tuple Class Property): A state s satisfies the *real tuple class property* if and only if $\forall R \in$ MREAL-IDS, $\forall c \in$ CLASSES,

$r \in$ mrelation-instance$(s, R, c) \Rightarrow$
mtuple-class$(r) =$
l.u.b. $\{c' \mid c' =$ melement-class$(r, i), 1 \leq i \leq$ mlength $(r)\}$.[8]

□

The access class c used to derive an instance represents an upper bound on the classes of all tuples and elements in the instance:

Property 4 (Visible Data Property): A state s satisfies the *visible data property* if and only if $\forall R \in$ MRELATION-IDS, $\forall c \in$ CLASSES such that $c \geq$ mrelation-class (R), $\forall r \in$ mrelation-instance (s, R, c):

1) tuple-class$(r) \leq c$, and
2) $\forall i : 1 \leq i \leq$ mdegree (R), melement-class$(r, i) \leq c$.

□

For example, Fig. 1 shows mrelation-instance $(s,$ FLIGHTS, SECRET); Fig. 2 shows mrelation-instance $(s,$ FLIGHTS, UNCLASSIFIED) for the relation FLIGHTS. (The "null" element value shown in Fig. 2 would appear as a blank entry to the user. We use the value "null" in our model as a special value to indicate that no value exists for the element in that tuple.)

The SeaView policy requires that the access class of a relation identifier be dominated by the access class of any data that can be stored in the relation. For $c \geq$ mrelation-class(R), this requirement is embodied in the data correctness property (property 16) of Section IV. For $c \not\geq$ mrelation-class(R), the requirement is formalized by the following visible relation property, which states that the multilevel relation instance derived at any access class that does not dominate the class of the relation identifier must be the null (i.e., empty) relation. In other words, the following property requires that for any access class domi-

[8] *l.u.b.* = least upper bound.

FLIGHT	C1	DEPARTS	C2	DEST	C3	T
964	U	1040	U	chicago	U	U
75	U	1400	U	berlin	S	S
1125	S	1730	S	san salvador	S	S

Fig. 1. mrelation-instance(s, FLIGHTS, SECRET).

FLIGHT	C1	DEPARTS	C2	DEST	C3	T
964	U	1040	U	chicago	U	U
75	U	1400	U	null	U	U

Fig. 2. mrelation-instance(s, FLIGHTS, UNCLASSIFIED).

nated by the access class of the relation, no tuples associated with the relation are visible.

Property 5 (Visible Relation Property): A state s satisfies the *visible relation property* if and only if $\forall R \in$ MRELATION-IDS, $c \in$ CLASSES:

$c \not\geq$ mrelation-class(R) $\Rightarrow$
mrelation-instance(s, R, c) $= \varnothing$

□

The function mrelation-instance(s, R, c) represents the different relation instances for the relation R that exist at different access classes. These instances are related to each other as follows. Any tuple r in the relation instance at class c with tuple class $c' < c$ must also appear in the relation instance at class c'.

Any tuple r that appears in mrelation-instance(s, R, c) with tuple-class t and key class $k_c \neq t$ must also appear in the instances at classes c', where $k_c \leq c' \leq t$, although high element values will be replaced with nulls (see the tuple for flight 75 in Figs. 1 and 2, for example). These requirements are formalized as follows.

Property 6 (Inter-Instance Property):[9] A state s satisfies the *inter-instance property* if and only if $\forall R \in$ MREAL-IDS, $\forall c \in$ CLASSES:

$r \in$ mrelation-instance(s, R, c) $\Rightarrow$

$\forall c'$ such that mrelation-class(R) $\leq c' < c$:

1) mtuple-class(r) $\leq c' \Rightarrow$

   $r \in$ mrelation-instance(s, R, c'); and

2) mtuple-class(r) $> c'$ and key-class(R, r) $\leq c' \Rightarrow$

   $\exists r' \in$ mrelation-instance(s, R, c') such that
   mtuple-key(R, r') $=$ mtuple-key(R, r) and
   $\forall i$: key-degree(R) $< i \leq$ mdegree(R):

   a) melement-class(r, i) $\leq c' \Rightarrow$
      melement-value(r', i) $=$ melement-value(r, i) and melement-class(r', i) $=$ melement-class(r, i); and

   b) melement-class(r, i) $\not\leq c' \Rightarrow$
      melement-value(r', i) $=$ null and
      melement-class(r', i) $=$ key-class(R, r).

□

[9]This statement of the Inter-Instance Property corrects an error in the original property as state in [19]. This error was pointed out to us by Sushil Jajodia and Ravi Sandhu [20].

F. Relational Integrity Rules

In the relational data model, consistency is defined, in part, by the two basic integrity rules of the relational model: entity integrity and referential integrity. (These rules apply to real relations only, i.e., not to views or snapshots.) The SeaView model includes these rules along with an additional rule, polyinstantiation integrity (we will discuss polyinstantiation in Section III-F-3). All three rules must apply at each access class; that is, every instance of a multilevel real relation must satisfy the rules.

1) Multilevel Entity Integrity: Entity integrity states that no tuple in a relation can have null values for any of the primary key attributes. If this constraint is to be satisfied with respect to the data visible at each access class, then in any given tuple, all the elements forming the primary key must all have the same access class. Otherwise, a subject whose access class is lower than that of the highest key element would see null values for some of the elements forming the key. In addition, the access class for the primary key must be dominated by the access classes of all other elements in the tuple. If the primary key class were not dominated by the class of some element in the tuple, then that element could not be uniquely selected by a subject operating at the element's access class. Thus *multilevel entity integrity* is expressed as follows.

Property 7 (Entity Integrity): A state s satisfies *entity integrity* if and only if $\forall R \in$ MREAL-IDS such that $n =$ mdegree(R) and $k =$ key-degree(R), $\forall c \geq$ mrelation-class(R), and $\forall r \in$ mrelation-instance(s, R, c), where $p =$ mtuple-key(R, r):

1) $\forall i: 1 \leq i \leq k$: melement-value($p$, i) $\neq$ null, and
2) $\forall i: 1 < i \leq k$:

   melement-class(p, i) $=$ key-class(r), and

3) $\forall i: 1 \leq i \leq n$:

   key-class(r) $\leq$ melement-class(r, i).

□

2) Multilevel Referential Integrity: Referential integrity states that every secondary key must reference a tuple that exists in some other relation where the key is primary. In a multilevel database, this means that a secondary key element cannot reference a tuple with a higher or noncomparable access class because the referenced tuple would appear to be nonexistent at the access class of the reference. *Multilevel referential integrity* requires that if a foreign key is visible at a given access class, then a tuple containing the referenced primary key must also be visi-

ble at that access class, and that the class of the foreign key element must equal the class of the referenced primary key.[10]

Property 8 (Referential Integrity): A state s satisfies *referential integrity* if and only if $\forall R \in$ MREAL-IDS, $\forall i: 1 \leq i \leq$ mdegree(R), where (Q, j) = foreign-key-ref(R, i), $\forall c \geq$ mrelation-class(R), $\forall r \in$ mrelation-instance(s, R, c):

melement-value(r, i) $\neq$ null $\Rightarrow$
$\exists q \in$ *mrelation-instance*(s, Q, c) such that

1) melement-value(r, i) = melement-value(q, j), and
2) melement-class(r, i) = melement-class(q, j).

□

3) Polyinstantiation: Unlike the standard relational model, which prohibits multiple tuples with the same primary key, in the SeaView model, a multilevel relation can have multiple tuples with the same primary key data value(s), but different access classes for either the key value(s) or for other data elements in the tuples. These tuples are referred to as *polyinstantiated data*. *Polyinstantiation* refers to the simultaneous existence of multiple data objects with the same name, where the multiple instantiations are distinguished by their access classes. Polyinstantiation is necessary, as we will see below, in order to hide the actions of high subjects from low subjects, thereby preventing signaling channels.

PolyInstantiated tuples (PITs) are tuples identified by a primary key and associated key class, so that the same multilevel relation may contain several tuple instances for a primary key value corresponding to different access classes. *PolyInstantiated elements* (PIEs) are elements identified by a primary key, key class, and element class (in addition to the attribute name), so that there may be multiple elements for an attribute that have different access classes, but are associated with the same (primary key, key class) pair.

A polyinstantiated tuple arises whenever a subject inserts a tuple that has the same primary key value as an existing but invisible (more highly classified) tuple. The effect of the operation is to add a *second* tuple to the relation, whose primary key is distinguishable from the first by its access class. Although the polyinstantiation is invisible to this subject, subjects at the higher access class can see both tuples. To illustrate, if an unclassified subject adds a tuple for flight number 1125 to the unclassified relation shown in Fig. 2, then the outcome, as seen by a SECRET subject, is as shown in Fig. 3.

A polyinstantiated element arises whenever a subject updates what appears to be a null element in a tuple, but which actually hides data with a higher access class. In this case, the update has the effect of creating a polyinstantiated element for the tuple. A polyinstantiated element can also arise when a high subject updates a low element—instead of overwriting the low element value, a PIE is created.

[10]This is a change from an earlier version of the SeaView model [19], which required only that the class of the foreign key element *dominate* the class of the referenced primary key. The change was made to eliminate the possibility of *referential ambiguity*, as pointed out by George Gajnak [21].

FLIGHT	C1	DEPARTS	C2	DEST	C3	T
964	U	1040	U	chicago	U	U
75	U	1400	U	berlin	S	S
1125	S	1730	S	san salvador	S	S
1125	U	1925	U	san francisco	U	U

Fig. 3. A polyinstantiated tuple.

We model PIE's as separate tuples. To illustrate, if our unclassified subject now replaces the perceived null value for the destination for flight 75 (see Fig. 2) with the value "paris," the outcome, as seen by a SECRET subject, is as shown in Fig. 4. Note, however, that the unclassified subject does not see two flights numbered 75—the unclassified subject's view of the relation is as shown in Fig. 5.

Polyinstantiation integrity specifies that there must never be two tuples with the same primary key unless they represent polyinstantiated tuples or elements and controls the effects of polyinstantiation.

Property 9 (Polyinstantiation Integrity): A state s satisfies *polyinstantiation integrity* if and only if $\forall R \in$ MREAL-IDS, where n = mdegree(R), and $\forall c \in$ CLASSES such that $c \geq$ mrelation-class(R):

$\forall r_1, r_2 \in$ mrelation-instance(s, R, c) such that $r_1 \neq r_2$

where p_1 = mtuple-key(R, r_1), p_2 = mtuple-key(R, r_2), $\forall i$:key-degree(R) $< i \leq n$:

1) There is a functional dependency from the primary key (including the key class) and ith element class to the ith element value:

$p_1 = p_2$ and element-class(r_1, i) = element-class(r_2, i)
$\Rightarrow$ element-value(r_1, i) = element-value(r_2, i); and

2) There is a multivalued dependency from the primary key to the ith element class and value:

$p_1 = p_2 \Rightarrow \exists r_3, r_4 \in$ mrelation-instance(s, R, c) such that mtuple-key(R, r_4) = p_1 and

element-value(r_3, i) = element-value (r_2, i)
element-class(r_3, i) = element-class(r_2, i)

element-value(r_4, i) = element-value(r_1, i)
element-class(r_4, i) = element-class(r_1, i)

and $\forall j$: key-degree(R) $< j \leq n, j \neq i$:

element-value(r_3, j) = element-value(r_1, j)
element-class(r_3, j) = element-class(r_1, j)

element-value(r_4, j) = element-value(r_2, j)
element-class(r_4, j) = element-class(r_2, j).

□

FLIGHT	C1	DEPARTS	C2	DEST	C3	T
964	U	1040	U	chicago	U	U
75	U	1400	U	berlin	S	S
75	U	1400	U	paris	U	U
1125	S	1730	S	san salvador	S	S
1125	U	1925	U	san francisco	U	U

Fig. 4. A polyinstantiated element.

FLIGHT	C1	DEPARTS	C2	DEST	C3	T
964	U	1040	U	chicago	U	U
75	U	1400	U	paris	U	U
1125	U	1925	U	san francisco	U	U

Fig. 5. View of polyinstantiated element to an unclassified subject.

G. Derived Data

New multilevel relations can be derived through relational operators. Rather than defining a fixed set of operators, the SeaView model includes a type called a multilevel relational experssion, which, when evaluated in a given state, returns a new relation.

- type MREL-EXPS of *multilevel relational expressions*. The terms of a multilevel relational expression are constants or multilevel relation identifiers (real relations, views, or snapshots).

Multilevel relational expressions are evaluated by a function *meval*, which maps a state, expression, and access class to a multilevel relation:

- *meval:* STATES $\times$ MREL-EXPS $\times$ CLASSES $\rightarrow$ MRELATION-VALUES, which returns the relation instance that results from evaluating a relational expression in a given state using the data visible at a given access class (i.e., visible to subjects whose read-class dominates the evaluation class).

The policy requires that the access class of derived information must dominate the access classes of the information used in the derivation. This reflects the fact that simply labeling data elements with their associated access classes is not enough; a tuple can encode information that is classified higher than any of the individual elements appearing in the tuple. To illustrate how this can happen, consider the multilevel EMPLOYEES relation shown in Fig. 6. Now suppose a subject performs the following query:

select emp-name from employees
where job = 'spy'

The single tuple returned from this query contains only the unclassified data elements "shockley" and "monterey"; however, this tuple clearly encodes secret information (namely, that employee "shockley" is a spy).

Thus, SeaView requires that the tuple class of a derived tuple $r \in$ meval(s, e, c) must dominate the classes of all

EMP-NAME	C1	ADDRESS	C2	JOB	C3	T
smith	U	sunnyvale	U	programmer	U	U
miller	U	menlo park	U	president	S	S
shockley	U	monterey	U	engineer	U	U
shockley	U	monterey	U	spy	S	S

Fig. 6. EMPLOYEES relation.

data used to derive r, which in turn means that r could be derived from data at class mtuple-class(r); that is, $r \in$ meval(s, e, mtuple-class (r)). For example, if a tuple with a tuple class of SECRET is contained in a derived relation at class TOP-SECRET, then it is also contained in a relation instance derived at class SECRET. Moreover, the model requires that the class of r must be dominated by c, because all of the data used to evaluate r must be dominated by c. These requirements are formalized as follows.

Property 10 (Labeling Derived Data Property): A state s satisfies the *labeling derived data property* if and only if for every multilevel relational expression e and class c:

1) $r \in$ meval(s, e, mtuple-class(r)), and
2) $r \in$ meval(s, e, c) $\Rightarrow$ mtuple-class(r) $\leq c$.

□

Property 10 would require the tuple returned from the query in the example above to be labeled as shown in Fig. 7.

In [22] we state how the tuple class should be derived for each of the five basic relational operators in order to satisfy the above property.

H. View Definitions

A multilevel view is defined by a relational expression (formula), which derives a multilevel view:

- *mview-def:* MVIEW-IDS $\rightarrow$ MREL-EXPS, which gives the formula for a view.

The SeaView policy requires that the class of each view definition dominate the class of any relation or view named in the view definition. Letting mview-mrels(R) denote the set of multilevel relations (real or virtual) named in the view definition of R, this requirement is formalized by the following.

Property 11 (View Class Integrity): A system satisfies *view class integrity* if and only if $\forall R \in$ MVIEW-IDS, $\forall R' \in$ mview-mrels(R):

$$\text{mrelation-class}(R) \geq \text{mrelation-class}(R').$$

□

This property also implies that the class of each view name dominates the classes of all relations that are referenced indirectly in the view formula (by transitivity of the dominance relation).

The next property states that the relation instance associated with a view must be the same as that obtained by evaluating the view definition expression.

EMP-NAME	C1	ADDRESS	C2	T
shockley	U	monterey	U	S

Fig. 7. Query result.

Property 12 (View Instance Property): A state s satisfies the *view instance property* if and only if $\forall R \in$ MVIEW-IDS, $\forall c \in$ CLASSES:

mrelation-instance(s, R, c) = meval(s, mview-def(R), c).

□

I. Snapshots

The purpose of snapshots is to give the user a mechanism for saving an instance of a multilevel relation or view, or saving the result of a query on some state of the database. In any given state s, each snapshot R has only one associated value, namely the multilevel relation instance defined at mrelation-class(R). All instances of R having classes that dominate mrelation-class(R) are equal to the instance at mrelation-class(R)—thus, there are not multiple "views" of snapshots as there are for real relations and views. In addition, the relation instance associated with a snapshot identifier in any given state must be a true "snapshot" of the database in some previous state—that is, it cannot be an arbitrary relation created by the user. These properties are formalized in the SeaView model report [18].

IV. Application-Specific Constraints

The SeaView policy requires that users be able to specify rules that define consistency of information and that authorized users be able to specify how information entering the system is to be assigned an access class. These requirements are modeled by application-dependent constraints on the values and classes that can be assigned to data entered into a multilevel real relation.

A. Value Constraints

Application-dependent value constraints correspond to user-specifiable integrity rules that restrict the values that data elements may take. Each multilevel real relation R has zero or more value constraints associated with each attribute at each access class that dominates mrelation-class(R). Each of these constraints is defined by a multilevel relational expression, which in turn defines a view. For example, the constraint $1 \leq X \leq 16$ on attribute X of R can be expressed as the multilevel relational expression "mselect(R, '$1 \leq X \leq 16$')," which defines a view on R.

When an insert or update is performed on a real relation at a given access class (typically the subject write-class), the value constraints that are visible at that class are evaluated. If the new or updated tuple is contained in all of the view instances defined by the value constraints, then the insert or update is accepted; otherwise, it is rejected. (We formalize this property later.)

Value constraints are defined by the following components:

- type VALUE-CONSTRAINTS of *value constraints*.
- *value-constraint-class:* VALUE-CONSTRAINTS → CLASSES, which gives the access class associated with a value constraint.
- *value-constraint-mrelation:* VALUE-CONSTRAINTS → MREAL-IDS, which gives the multilevel real relation identifier for which the constraint applies.
- *value-constraint-attribute:* VALUE-CONSTRAINTS → $\mathfrak{N}$, which gives the attribute index for which the constraint applies.
- *value-constraint-exp:* VALUE-CONSTRAINTS → MREL-EXPS, which gives the multilevel relational expression defining the allowable values specified by the value constraint.

The SeaView policy requires that the access class of a value constraint dominate the access class of the identifier of the relation to which it applies.

Property 13 (Value Constraint Class Property):

$\forall VC \in$ VALUE-CONSTRAINTS:
value-constraint-class(VC) $\geq$ mrelation-class(value-constraint-mrelation(VC)).

□

B. Classification Constraints

The SeaView policy requires that users be able to specify rules, called *classification constraints*, that define how information entering the system is to be assigned an access class. Classification constraints are similar to value constraints, except that they restrict the access classes of data elements rather than their values. For example, the constraint "*class*(PAYLOAD.CARGO) = *class*(FLIGHTS.FLIGHT) where PAYLOAD.FLIGHT = FLIGHTS.FLIGHT" on the CARGO attribute of relation PAYLOAD assigns the access class of the cargo's flight (as stored in the FLIGHTS relation) to CARGO values.

Each multilevel real relation R has zero or more classification constraints associated with each attribute at each access class that dominates mrelation-class(R). R also has at least one classification constraint at the access class mrelation-class(R). Each of these constraints maps to a multilevel relational expression, which defines a view, and a class expression, which represents a *lower bound* on the access class of data that falls within the view defined by the relational expression. When an insert or update is performed on a real relation at a given access class, the multilevel relational expressions for the classification constraints visible at that class are evaluated. If the tuple to be inserted or updated is associated with one or more of the views defined by the relational expressions, then the insert or update is accepted, and the element is labeled with the least upper bound of the classes assigned by the class expressions for those views; otherwise, the tuple is rejected.

We first define class expressions and a function *class-eval* for evaluating a class expression.

- type CLASS-EXPS of *class expressions*. The terms of a class expression represent constant classes (e.g., SECRET), the classes of elements in the database, the write-class of the subject, or a class provided by the subject. The operators include the least-upper-bound operator.

Each class expression evaluates to an access class through the following function.

- *class-eval:* STATES × CLASS-EXPS × CLASSES × SUBJECTS × MTUPLES → CLASSES, which returns the class that results from evaluating a class expression in a given state at a given access class relative to a given subject and to a given tuple (the tuple being updated or inserted).

The class-eval function maps from a subject in order to implement classification constraints in which data are specified to have the access class of the data source or to be source-labeled.

Classification constraints are defined by the following components:

- type CLASS-CONSTRAINTS of *classification constraints*.
- *class-constraint-class:* CLASS-CONSTRAINTS → CLASSES, which gives the access class associated with a classification constraint.
- *class-constraint-mrelation:* CLASS-CONSTRAINTS → MREAL-IDS, which gives the multilevel real relation identifier for which the constraint applies.
- *class-constraint-attribute:* CLASS-CONSTRAINTS → $\mathfrak{N}$, which gives the attribute index for which the constraint applies.
- *class-constraint-mrel-exp:* CLASS-CONSTRAINTS → MREL-EXPS, which gives the relational expression (if any) for a constraint. If no relational expression is associated with a classification constraint C, we denote this by class-constraint-mrel-exp(C) = " ". Such a constraint can be used to, in effect, prohibit new data from being entered for an element.
- *class-constraint-class-exp:* CLASS-CONSTRAINTS → CLASS-EXPS, which gives the expression defining the result class for a constraint.

To avoid disclosing the existence of a relation to subjects with access classes lower than the relation class, SeaView requires that the access class of a classification constraint dominate the access class of the identifier of the relation to which it applies.

Property 14 (Classification Constraint Class Property): $\forall CC \in$ CLASS-CONSTRAINTS:

class-constraint-class(CC) ≥ mrelation-class(class-constraint-mrelation(CC)). □

Value and classification constraints are not defined for views, because insert or update operations on views are mapped to corresponding operations on the underlying real relations, and the value and classification constraints on those real relations apply.

In order to ensure that a complete set of classification constraints is visible at every access class at which a relation is visible, the SeaView policy requires that every attribute of a relation have a classification constraint at the class of the relation schema.

Property 15 (Classification Constraint Completeness Property): $\forall R \in$ MREAL-IDS, $\forall i$, $1 \le i \le$ mdegree(R):

$\exists CC \in$ CLASS-CONSTRAINTS such that
class-constraint-class(CC) = mrelation-class(R),
class-constraint-mrelation(CC) = R, and
class-constraint-attribute(CC) = i. □

C. Correctness Properties

When an element is inserted or updated in a real relation R, the value of the element must satisfy the value constraints associated with the corresponding attribute of R. In addition, the access class of the element must satisfy the classification constraints associated with the corresponding attribute of R, and this class must dominate the class of the relation name. In order to state this property formally, we first define what it means for an element to satisfy the constraints in a particular state. (The SeaView model does not require that all constraints be satisfied in all states, but the constraints must be satisfied by the states that result from insert and update operations.)

Each constraint has an associated relational expression e. For any given state s and class c, this expression defines a relation instance I. For the purposes of constraint application, a tuple is considered to be associated with the relation instance I if the values of its elements correspond to the values of the elements of a tuple in I (the classes of the elements and the tuple class need not match). This is expressed by a function mrelation-associated(I, r) which returns true if and only if tuple r is associated with relation instance I.

Now, an element of a tuple satisfies the value constraints of the corresponding attribute in a given state with respect to a particular subject if the tuple is associated with the view corresponding to each constraint, where the views are evaluated with respect to the read-class of the subject. The following definition states that an element is correct in a state s with respect to a subject S if for all value constraints that apply to the element, the tuple is associated with the view defined by the constraint.

Definition 1 (Value Correctness): Let S be a subject with read-class rc. Let $R \in$ MREAL-IDS, and $r \in$ mrelation-instance(s, R, rc). Then element i of tuple r, for $1 \le i \le$ mlength(r), is *correct in state s with respect to* S if and only if $\forall VC \in$ VALUE-CONSTRAINTS:

value-constraint-mrelation(VC) = R,
value-constraint-attribute(VC) = i,
value-constraint-exp(VC) = e, and

value-constraint-class (VC) $\leq rc \Rightarrow$
mrelation-associate (meval (s, e, rc), r) = true. □

Similarly, an element of a tuple satisfies the classification constraints of its corresponding attribute in a given state with respect to a particular subject if for each constraint such that the tuple is associated with the view corresponding to the constraint, the class of the element dominates the class specified by the constraint; again, the views are evaluated with respect to the read-class of the subject.

Definition 2 (Classification Correctness): Let S be a subject with read-class rc. Let $R \in$ MREAL-IDS, and $r \in$ mrelation-instance (s, R, rc). Then element i of tuple r, for $1 \leq i \leq$ mlength (r), is *classified correctly in state s with respect to subject S* if and only if:

1) There exists at least one classification constraint that assigns a class to the element and that is visible to the subject:

$\exists CC \in$ CLASS-CONSTRAINTS and $x \in$ CLASSES such that:

a) class-constraint-mrelation (CC) = R, class-constraint-attribute (CC) = i, and class-constraint-class (CC) $\leq rc$, i.e., the constraint is associated with the ith attribute of R and is visible at the subject's read-class;

b) mrelation-associate (meval (s, class-constraint-mrel-exp (CC), rc), r) = true, i.e., the tuple r is associated with the view defined by the classification constraint evaluated at class rc;

c) class-eval (s, class-constraint-class-exp (CC), rc, S, r) = x (i.e., the class assigned by the constraint is x); and

d) $x \geq$ mrelation-class (R) − i.e., the assigned class dominates the class of the relation name.

2) For all such constraints satisfying the above,

element-class (r, i) = the lub of the x

(to force *consistency*). □

The classification specifications visible at any access class must be *consistent*; that is, if two or more must be satisfied simultaneously, then they must yield the same access class. Because the class returned by a constraint is taken to be a *lower bound* on the class to be assigned to the element, the definition of classification correctness, above, ensures that the classification constraints visible at any access class are consistent by taking the least upper bound of the classes returned by all the visible constraints that apply.

The SeaView policy requires that the access class of a relation identifier be dominated by the access class of any data that can be stored in the relation. This requirement is formalized through conditions (1d) and (2) in Definition 2.

The following transition property requires that all updated and inserted elements satisfy the value and classification constraints.

Property 16 (Data Correctness Property): A command $op(s_1, S, x_1, \cdots, x_n \rightarrow s_2)$ satisfies the *data correctness property* if and only if $\forall R \in$ MREAL-IDS, $\forall c \in$ CLASSES:

1) No new tuples are associated with R:

mrelation-instance (s', R, c) $\subseteq$ mrelation-instance (s, R, c),

or

2) For each tuple r that represents an insert into R, all element values satisfy the value constraints and all element classes satisfy the classification constraints:

$\forall r \in$ mrelation-instance (s', R, c) such that $\neg \exists q \in$ mrelation-instance (s, R, c) where mtuple-key (R, r) = mtuple-key (R, q) (since the primary key is an mtuple, the classes of the elements are included for the comparison; thus, polyinstantiated tuples are not equal)

$\forall i$, $1 \leq i \leq$ mlength (r): element i of r is correct in state s' with respect to S, and element i of r is classified correctly in state s' with respect to S,

or

3) For each tuple r that represents an update, each updated element satisfies the classification constraints and value constraints:

$\forall r \in$ mrelation-instance (s', R, c) such that $\exists q \in$ mrelation-instance (s, R, c) where mtuple-key (R, r) = mtuple-key (R, q), $\forall i$, $1 \leq i \leq$ mlength (r): if $\neg \exists p \in$ mrelation-instance (s, R, c) where mtuple-key (R, r) = mtuple-key (R, p) and element-class (r, i) = element-class (p, i) and element-value (r, i) = element-value (p, i) (i.e., we really have a new element—recall that primary key and element class must determine element value),
then element i of r is correct in state s' with respect to S, and element i of r is classified correctly in state s' with respect to S.

□

Mandatory security requires that write-class (S) $\leq c \leq$ read-class (S); otherwise mrelation-instance (s', R, c) = mrelation-instance (s, R, c). This is enforced by the underlying reference monitor.

V. Discretionary Security

The SeaView policy requires that no user be given access to information unless that user has been granted discretionary authorization to the information. This policy is formalized in terms of subjects and protected database objects. The *protected database objects* are databases, relations (real relations, views, and snapshots), and objects at the MAC interface. The TCB model allows users to specify which users and groups are authorized for specific modes of access to particular database objects, as well as which users and groups are explicitly denied authorization for particular database objects.

The TCB model uses access modes to signify authorized access to database objects, and includes different sets of access modes applicable to different types of objects. A special mode, called the "null" mode, is used to signify that *no* access to the information is authorized. Granting "null" authorization is the means of denying authorization. The "null" mode supercedes all other modes.

To obtain access to a database object O, a user must be authorized for the corresponding access mode for O. The SeaView model applies a *most specific* rule: if an individual user is explicitly granted or denied authorization to a database object, this takes precedence over any authorizations that are granted or denied to groups to which the user belongs.

A user may be authorized for access mode m to a view defined on one or more multilevel relations without being authorized for mode m to the underlying multilevel relation(s). However, for a user to obtain access to a view, the user must be authorized for the reference mode on all referenced multilevel relations. This allows for the following common use of views for access control: suppose user U_1 has all authorizations for a multilevel relation $R(A, B, C)$, and attribute C contains personal information relevant only to U_1. Suppose also that U_1 wants to make attributes A and B available to user U_2 for retrieval and update. U_1 can create a view V on R consisting of attributes A and B only and can then grant U_2 retrieval and update authorization for V without granting U_2 *any* authorizations (other than the reference mode) for R. (In order for U_2 to use view V, U_1 has to grant U_2 the reference mode for R.) Note that even if U_2 had been able to create the view, U_2 still could not obtain any authorizations for the view that U_2 did not also hold on R.

When a user creates a database object other than a view, default initial authorizations are granted to the user for the object. When a user creates a view, the user acquires only those access modes (or a subset of them) that are held on each directly referenced underlying relation and view (i.e., those relations and views named in the view formula). In addition, certain groups are automatically granted authorizations for new multilevel relations.

A user can grant or revoke an access mode for a database object to or from other users and groups, provided the granting user is so authorized. The propagation of access modes is controlled through the access modes "grant" and "give-grant." If a user is authorized the "grant" access mode for a database object O, then that user can grant and revoke any access mode other than "grant" and "give-grant" for O, and also can deny all authorization to O from a user or group. A user authorized the "give-grant" access mode can additionally grant and revoke the "grant" and "give-grant" access modes. Revocation of authorizations does not extend dynamically to users who may have been granted authorization by the user from whom authorization is being revoked as in System R [23].

Unlike the mandatory security policy, the discretionary security policy does not require secure information flow. In our model, the granting and revoking of access modes affects the current *authorizations* but not the current *accesses* (it affects future accesses). Thus, the discretionary security properties are stated as transition properties and relate the discretionary authorizations and accesses in the current state to the accesses in the next state.

To enforce the discretionary policy, discretionary authorizations and accesses for a multilevel relation R must be stored in objects classified at mrelation-class(R). Otherwise, relevant authorization information about R might not be visible. The underlying reference monitor prevents the discretionary authorizations from being used as covert signaling channels because any authorizations stored at access class c cannot be applied to subjects at classes lower than c.

VI. Model Implementation Issues

The SeaView policy requires that the security mechanisms of a system that enforce discretionary security and all supporting policies be constrained by a reference monitor that enforces mandatory security.

For TCB model components defined as types (domains of values), the only elements visible to a subject S are those whose access class is dominated by read-class(S). All information implementing the TCB model, including both state-dependent and state-independent functions, is contained in objects of appropriate classification, which in turn are managed by the reference monitor. Implicit in the state properties and the state-independent properties and axioms of the TCB model is the requirement that the properties be true for all $c \in$ CLASSES with respect to the information visible at that class. Implicit in the transition properties and the command sequence properties is the requirement that the *visible* values of the state-dependent functions in the new state depend only on the *visible* values of any functions in the previous state, where visible means at the read-class of the subject causing a state transition.

In the SeaView design, all information about a multilevel relation R (e.g., mdegree, primary-key) is stored in objects of class mrelation-class(R). This means that a subject S with read-class $rc \leq$ mrelation-class(R) is unable to obtain information about R. Value and class constraints classified higher than mrelation-class(R) are stored in objects at their respective (higher) classifications.

Relation instances, which are a function of a state s, relation identifier R, and access class $c \geq$ mrelation-class(R), are derived from the contents of objects at class c or lower as follows. For a mutlilevel real relation R, mrelation-instance(s, R, c) is derived (using the standard relational operators) from standard, single-level relations at classes dominated by c as described in Section VI-A, below. Thus, each multilevel real relation is effectively a view over single-level relations, where each single-level

relation R_1 has a class $c_1 \leq c$ and is stored in one or more objects at class c_1. For a multilevel view R, mrelation-instance(s, R, c) is derived from the multilevel relations referenced in the view definition formula, where the instances used to derive the view are at class c. For both real relations and views, a subject with read-class rc can obtain mrelation-instance(s, R, c) only if $rc \geq c$. For a snapshot R, mrelation-instance(s, R, c) for $c \geq$ mrelation-class(R) is stored in a standard, single-level relation R_1 of class mrelation-class(R), which is stored in one or more objects of that class.

A. The SeaView Decomposition

As we discussed in Section I-B, SeaView implements multilevel real relations as views over underlying single-level stored relations. Each single-level standard relation, in turn, is mapped onto one or more single-level segments, which are protected by an underlying reference monitor. Therefore, a subject S will be unable to access any data in an underlying relation (in order to derive a multilevel relation) unless *read-class*(S) dominates the class of the object(s) that contains the stored data.

The decomposition automatically satisfies polyinstantiation integrity. The redundancy defined by the multivalued dependencies is removed in the decomposition, but restored when the full relation is instantiated.

The following decomposition corrects the decomposition and recovery formula in our previously-published multilevel relational data model [22]. The formula in that paper used the full outer join operator to perform the recovery, which had the effect of reclassifying the low key values associated with high "orphans" as high, an approach which, as we saw above, could result in the failure to preserve polyinstantiation integrity. Our formula uses the left outer join, which avoids this problem.[11] Otherwise, our decomposition here is similar to the approach there, with the notable exception that here we require far fewer base relations.

All attributes of a multilevel real relation R that are uniformly classified (that is, attribute groups whose values have like classifications within any given tuple) are handled as a unit. Thus, in the following discussion, each A_i in the relation $R(A_1, C_1, \cdots, A_n, C_n)$ represents a set of attributes that are uniformly classified, and C_i represents a classification attribute for the set. (A user may designate a set of attributes to be uniformly classified in order to control polyinstantiation.) Note that because multilevel entity integrity requires the apparent primary key to be uniformly classified, the attributes forming the apparent primary key belong to the same attribute group, which we can assume with no loss of generality to be A_1. (The apparent primary key is composed of those attributes designated by the user as forming the primary key.) A_1 may include other attributes besides the key attributes if those additional attributes are also uniformly classified at the key class. We let K denote the set of attributes forming the apparent primary key; thus $K \subseteq A_1$.

The domain of the classification attribute C_i is the range of classifications for data that can be associated with attribute group A_i. This domain is a sublattice of the lattice of access classes, having a lowest class L_i and a highest class H_i. We designate this domain $range(A_i) = [L_i, H_i]$.

Our decomposition of multilevel real relations into single-level base relations is as follows. We create one single-level relation $R_{1,c}$ for each $c \in [L_1, H_1]$. Each of these relations has classification c and consists of the attribute group A_1 (which includes the key attributes). Then for each attribute group A_i for $i > 1$, a standard base relation $R_{i,c}$ is created for each class $c \in [L_i, H_i]$. Each of these relations has classification c and consists of the key attribute K, the key class C_1, and the attribute A_i. In short:

Primary Key Group Relations: $\forall c \in [L_1, H_1]$, create $R_{1,c}(A_1)$ with class c.

Attribute Group Relations: For $i = 2, \cdots, n$: $\forall c \in [L_i, H_i]$, create $R_{i,c}(K, C_1, A_i)$ with class c.

Note that multilevel entity integrity guarantees that $C_1 \leq C_i$, so that the values for K can be stored in a relation of class c. By encoding the primary key class as a field in the base relation itself, instead of in its name, we need fewer base relations for the decomposition that did the previously published formula [22], which created a base relation for each possible combination of key class and attribute class. In addition, by including in A_1 all non-key attributes that are uniformly classified at the key class, we also need fewer base relations than did our earlier formula [22]. We do not actually create a base relation from the multilevel relation until data exists at the appropriate class.

Our recovery of a multilevel real relation from its single-level base relations is as follows. Let $P_{1,c}$ represent the derived relation (derived from $R_{1,c}$)

$$P_{1,c} = (A_1, C_1 = c)$$

and let $P_{i,c}$ represent the derived relation (derived from $R_{i,c}$)

$$P_{i,c} = (K, C_1, A_i, C_i = c).$$

In each of these, $C_i = c$ represents the constant column of value c. Now we can derive R as follows:

$$R = (P_1 \bowtie P_2 \bowtie \cdots \bowtie P_n)$$

where for $i = 1 \cdots n$,

$$P_i = \bigcup_{c \in [L_i, H_i]} P_{i,c}$$

and where $\bowtie$ denotes the left outer join operator, and all joins are on the attributes A_1, C_1; that is, on the key value and key class.

B. Data Design Considerations

When a multilevel relation is defined, an allowable access class domain is specified for each data attribute or

[11] In [8], we inadvertently wrote "right outer join" where the left outer join had been intended. This error was also pointed out by Jajodia and Sandhu [20].

attribute group. If all attribute groups of a multilevel relation R are single-level (i.e., their domains all contain a single class), then polyinstantiation does not arise in R and no unions are used to recover R. If furthermore all the single-level attribute groups are the same class, then the decomposition of R yields a single base relation, so that there is no overhead in instantiating R.

Uniformly classified attributes form attribute groups in a multilevel relation schema; for example, latitude and longitude would probably be uniformly classified. If all attributes form a single group, the relation is in effect classified at the tuple level, and no joins are needed to instantiate the multilevel relation.

Specifying narrow classification ranges for attributes can potentially improve performance of the recovery algorithm by reducing the number of unions needed. Grouping uniformly classified attributes whenever possible reduces the number of base relations needed in the decomposition and the number of joins needed in the recovery. Both narrowing the classification ranges and grouping the attributes also reduce the effects of polyinstantiation. MSQL allows users to specify these data design decisions.

VII. Conclusions

Multilevel security has all-pervasive effects on the data model, data consistency, database system architecture and design, and data manipulation language. We have presented SeaView's security model, which defines multilevel relations to contain classification attributes as well as data attributes. We discussed SeaView's multilevel relational integrity rules, which extend the integrity constraints of the relational model in order to provide consistency for data at different access classes, including data that becomes polyinstantiated. We also discussed SeaView's decomposition method for mapping all multilevel real relations into standard (single-level) base relations. By implementating multilevel relations as views over single-level base relations, we obtain element-level labeling without significant storage overhead. The SeaView design builds on an existing database management system ported to an existing reference monitor to obtain A1 assurance for mandatory security for the system as a whole.

References

[1] Nat. Comput. Security Center, *Dep. Defense Trusted Computer System Evaluation Criteria*, Tech. Rep. DOD 5200.28-STD, Dec. 1985.

[2] T. F. Lunt, D. E. Denning, P. G. Neumann, R. R. Schell, M. Heckman, and W. R. Shockley, *Final Report Vol. 1: Security Policy and Policy Interpretation for a Class A1 Multilevel Secure Relational Database System*. Comput. Sci. Lab., SRI International, Menlo Park, CA, Tech. Rep., 1988.

[3] T. F. Lunt and R. A. Whitehurst, *Final Report Vol. 3A: The SeaView Formal Top Level Specifications*. Comput. Sci. Lab., SRI International, Menlo Park, CA, Tech. Rep., 1989.

[4] T. F. Lunt, R. R. Schell, W. R. Shockley, M. Heckman, and D. Warren, "Toward a multilevel relational data language," in *Proc. Fourth Aerospace Computer Security Applications Conf.*, Orlando, FL, IEEE Computer Society Press, Dec. 1988.

[5] J. S. Crow, R. Lee, J. M. Rushby, F. W. von Henke, and R. A. Whitehurst, "EHDM verification environment: An overview," in *Proc. 11th Nat. Computer Security Conf.*, Nat. Bureau Standards/National Computer Security Center, Baltimore, MD, Oct. 1988.

[6] R. A. Whitehurst and T. F. Lunt, "The SeaView verification," in *Proc. Second Workshop Foundations of Computer Security.*, Franconia, NH, IEEE Computer Society Press, June 1989.

[7] D. E. Denning, *Cryptography and Data Security*. Reading, MA: Addison-Wesley, 1982.

[8] T. F. Lunt, R. R. Schell, W. R. Shockley, M. Heckman, and D. Warren, "A near-term design for the SeaView multilevel database system," in *Proc. 1988 IEEE Symp. Security and Privacy*, Oakland, CA, IEEE Computer Society Press, Apr. 1988.

[9] T. F. Lunt, "Multilevel database systems: Meeting class A1," in *Database Security II, Status and Prospects*. New York: Elsevier Science, Oct. 1988.

[10] T. H. Hinke and M. Schaefer, "Secure data management system," System Development Corp., Tech. Rep. RADC-TR-75-266, Nov. 1975.

[11] M. J. Grohn, "A model of a protected data management system," I. P. Sharp Associates Ltd., Tech. Rep. ESD-TR-76-289, June 1976.

[12] T. H. Hinke, C. Garvey, N. Jensen, J. Wilson, and A. Wu, "A1 secure DBMS design," in *Postscript to Proc. 11th Nat. Computer Security Conf.*, Nat. Bureau Standards/Nat. Comput. Security Center, Baltimore, MD, Oct. 1988.

[13] R. D. Graubart and J. P. L. Woodward, "A preliminary naval surveillance DBMS security model," in *Proc. 1982 IEEE Symp. Security and Privacy.*, Oakland, CA, IEEE Computer Society Press, Apr. 1982.

[14] P. Dwyer, E. Onuegbe, P. Stachour, and B. Thuraisingham, "Query processing in LDV: A secure database system," in *Proc. Fourth Aerospace Computer Security Applications Conf.*, Orlando, FL, IEEE Computer Society Press, Dec. 1988.

[15] O. S. Saydjari, J. M. Beckman, and J. R. Leaman, "LOCK trek: Navigating uncharted space," in *Proc. 1989 Symp. Research in Security and Privacy*, Oakland, CA, IEEE Computer Society Press, May 1989.

[16] D. E. Bell and L. J. LaPadula, "Secure computer systems: Unified exposition and multics interpretation," MITRE Corp., Bedford, MA, Tech. Rep. ESD-TR-75-306, Mar. 1976.

[17] R. R. Schell, T. F. Tao, and M. Heckman, "Designing the GEMSOS security kernel for security and performance," in *Proc. 8th Nat. Computer Security Conf.*, Nat. Bureau Standards/Nat. Comput. Security Center, 1985.

[18] T. F. Lunt, D. E. Denning, R. R. Schell, M. Heckman, and W. R. Shockley, "Final report Vol. 2: The SeaView formal security policy model," Comput. Sci. Lab., SRI International, Menlo Park, CA, Tech. Rep., 1989.

[19] D. E. Denning, T. F. Lunt, R. R. Schell, W. R. Shockley, and M. Heckman, "The SeaView security model," in *Proc. 1988 IEEE Symp. Security and Privacy.*, Oakland, CA, IEEE Computer Society Press, Apr. 1988.

[20] S. Jajodia and R. Sandhu, "Polyinstantiation integrity in multilevel relations," Center of Excellence for Command, Contr., Commun., Intell., George Mason Univ., Fairfax, VA, unpublished paper, 1989.

[21] G. E. Gajnak, "Some results from the entity/relationship multilevel secure DBMS project," in *Proc. Fourth Aerospace Computer Security Applications Conf.*, Orlando, FL, IEEE Computer Society Press, Dec. 1988.

[22] D. E. Denning, T. F. Lunt, R. R. Schell, M. Heckman, and W. R. Shockley, "A multilevel relational data model," in *Proc. 1987 IEEE Symp. Security and Privacy.*, Oakland, CA, IEEE Computer Society Press, Apr. 1987.

[23] P. P. Griffiths and B. W. Wade, "An authorization mechanism for a relational database system," *ACM Trans. Database Syst.*, vol. 1, no. 3, Sept. 1976.

Teresa F. Lunt (M'89) received the A.B. degree from Princeton University, Princeton, NJ, in 1976, and the M.A. degree in applied mathematics from Indiana University, Bloomington, in 1979.

She is with SRI's Computer Science Laboratory, where she manages a research program in computer security. She is leading two landmark programs: the SeaView multilevel secure relational database system and the IDES Intrusion-Detection system. She is also leading a new research area in security for knowledge-based systems and using AI techniques for computer security. Prior to joining SRI in early 1986, she worked

at The MITRE Corporation and later at SYTEK's Data Security Division. She has also worked on audit trail analysis, automated security guards, security models, and formal verification of secure systems. She has published over 30 conference and journal papers and over 15 technical reports in the area of computer security. She is founding editor and principal contributor to the *Data Security Letter*.

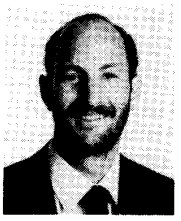
Dorothy E. Denning received the Ph.D. degree in computer science from Purdue University, West Lafayette, IN.

She is a member of the Research Staff at Digital Equipment Corporation Systems Research Center. Before joining Digital, she was a senior staff scientist at SRI and an Associate Professor of Computer Science at Purdue University. She is author of *Cryptography and Data Security* and numerous articles on computer security.

Dr. Denning is past President of the International Association for Cryptologic Research.

Roger R. Schell (S'69-M'74) received the B.S. degree in electrical engineering from Montana State College, the M.S. degree in electrical engineering from Washington State University, Pullman, and the Ph.D. degree in computer science from the Massachusetts Institute of Technology, Cambridge.

He is Vice President for Engineering and one of the founders of Gemini Computers, Inc., Monterey, CA. His interests include databases, operating systems, and computer security. For three years he was the Deputy Director of the Department of Defense Computer Security Center. He was an Associate Professor of Computer Science at the Naval Postgraduate School from 1978 to 1981. He has served as program manager for large military software developments, has been a systems programmer, and introduced the security kernel technology.



Mark Heckman received the B.S. degree in computer engineering from the University of California at San Diego in 1982.

He is the Director of Engineering Services at Gemini Computers, Inc., where he is the chief contact for customer technical support. He is currently responsible for the design and implementation of systems security and system administrator capabilities for the Gemini Multi-processing Secure Operating System, in addition to participating actively in the Multilevel Secure Data Views project.

William R. Shockley, photograph and biography not available at the time of publication.