# Views for Multilevel Database Security

DOROTHY E. DENNING, SELIM G. AKL, SENIOR MEMBER, IEEE, MARK HECKMAN, TERESA F. LUNT, MATTHEW MORGENSTERN, PETER G. NEUMANN, SENIOR MEMBER, IEEE, AND ROGER R. SCHELL, MEMBER, IEEE

**_Abstract_—Because views on relational database systems mathematically define arbitrary sets of stored and derived data, they have been proposed as a way of handling context- and content-dependent classification, dynamic classification, inference, aggregation, and sanitization in multilevel database systems. This paper describes basic view concepts for a multilevel-secure relational database model that addresses the above issues. All data entering the database are labeled according to views called classification constraints, which specify access classes for related data. In addition, views called aggregation constraints restrict access to aggregates of information. All data accesses are confined to a third set of views called access views.**

**_Index Terms_—Classification, multilevel security, protection, relational databases, security, views**

## I. Introduction

THE objective of this paper is to describe basic view concepts for a multilevel-secure relational database model. The model is being developed as part of a three-year project to design a system that will meet the Department of Defense Trusted Computer System Evaluation Criteria [1] for class A1. The project goals include producing a security policy, formal model, formal top-level specifications, and implementation specifications.

Although the concept of using views for security goes back at least as far as the CODASYL work [2], our approach builds more on the view concept introduced in IBM's System R database system (now called SQL/DS), which was inspired by Codd's fundamental work on relational databases [3]. In System R, a view is a derived relation expressed in the Structured Query Language SQL. The access control mechanisms of System R are tied to views in that views (as well as base relations) are the objects of authorization [4] (see also Date [5] and Denning [6]). The rationale for this decision was that views are at a higher level of abstraction than the physical data and allow the specification and enforcement of context- and content-dependent constraints. For the same reason, Stonebraker [7] adopted a high-level approach in the INGRES relational system, although the strategy there uses query modification rather than views.

Concurrent with the development work at IBM, Neumann observed that views provided an attractive method for implementing a secure relational data management system on top of SRI's Provably Secure Operating System (PSOS) [8]. In the PSOS approach, a view is restricted to a subset of a single relation and serves as a capability for selective access to the relation.

Neither the IBM nor SRI projects addressed the issues that would be raised if views were used to classify data and enforce mandatory security. Proposals to use secure views as a basis for multilevel-secure database systems were independently made by Claybrook [9] and by Denning [10], who at that time was helping to organize the 1982 Woods Hole Summer Study on Multilevel Database Management Security sponsored by the National Academy of Sciences, Air Force Studies Board. Denning observed that because views can define arbitrary sets of stored and derived data, they could provide a means of addressing the problems of context- and content-dependent classification, inference, aggregation, and sanitization on a dynamic database. A study group led by Denning and Neumann discussed the benefits and issues associated with classifying views, concluding that the benefits justified further research, but that there were many open problems and issues [11]. The approach outlined in this paper addresses these issues and provides a basis for a system design based on secure views.

Our preliminary model has several key aspects. First, we explicitly allow the specification of derived data in a database schema so that the relationships between stored and derived data (which can cause inferences) can be formally expressed. Next, we distinguish between views that retrieve or update data, called *access views*, and views that classify the data, called *classification constraints*. Access views allow retrieval of data up through the user's clearance from a relation that also may have higher-level data, without the need to retrieve from a higher-level container—our model has no containers. Data retrieved through access views can be joined and manipulated for display to the user. In addition, the data retrieved can be multilevel, with support for classification down to the element level. Data enter the database through the access views and are labeled on entry according to classification constraints. Classification constraints specify access classes and relationships among stored and derived data,

Manuscript received December 20, 1985; revised August 1, 1986. This work was supported by the U.S. Air Force, Rome Air Development Center, under Contract F30602-85-C-0243 and by the National Science Foundation under Grant MCS-8313650.

D. E. Denning, T. F. Lunt, M. Morgenstern, and P. G. Neumann are with SRI International, Menlo Park, CA 94025.

S. G. Akl is with the Department of Computer and Information Science, Queen's University, Kingston, Ont. K73 3N6, Canada.

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

IEEE Log Number 8611563.

0098-5589/87/0200-0129$01.00 © 1987 IEEE

thereby providing a means of handling context- and content-dependencies, inferences, and sanitization. A third set of views called *aggregation constraints* serve to define and control access to aggregates of information.

Because our objective is a class A1 system, it is imperative that our formal model of the reference monitor be tractable. We plane to achieve this objective by layering our design and placing only the essential (namely, mandatory) security features in the layer that comprises the reference monitor. Most of the database system, including the query processor, will reside above the reference monitor layer. A consequence of this approach is that certain features in our policy model—e.g., for content-dependent classification and sanitization—will rely on trusted processes that reside in layers above the reference monitor. Our preliminary design, outlined in Section VI, builds on existing technology for security kernels.

Because we are in the first year of a three-year effort, our policy model and design are somewhat tentative and incomplete; there remain several open problems and many details to be resolved. The concepts described in this paper represent an initial step in our research.

## II. Basic Concepts

### A. Access Classes

Fundamental to any multilevel security policy is a set of access classes that represent the classifications associated with information and the clearance associated with users. Each *access class* is an element of a lattice structure having a partial ordering relation "≥" (e.g., see Denning [6]). For access classes $L1$ and $L2$, $L1 \geq L2$ means that access class $L1$ *dominates* access class $L2$. (If $L1 > L2$, we say $L1$ *strictly dominates* $L2$). Note that what we call "access class" is identical to what some other authors have called "security level." We prefer "access class" to 1) avoid the (incorrect) implication that "level" applies only to total orderings and 2) avoid confusion with the term "multilevel" in DoD Directive 5200.28, which refers specifically to just the totally ordered classifications CONFIDENTIAL, SECRET, etc.

All data in the database are assigned an access class, or *classification*. In addition, each user has an associated access class, or *clearance*. To access data in the database, the user's clearance must dominate the classification of the data. An access class associated with data or a user is specified by a security label.

Our model leaves unspecified the exact representation and interpretation of access class. For a given system, the access class may consist of a secrecy component, an integrity component, or both (as in the I.P. Sharp model [12]). The *secrecy* component could be a secrecy level, secrecy category, or pair <secrecy level, secrecy category>, where *secrecy level* is TOP-SECRET, SECRET, CONFIDENTIAL, UNCLASSIFIED, etc., and *secrecy category* is a set consisting of formal compartments (e.g., CRYPTO). Similarly, the *integrity* component could be an integrity level, integrity category, or pair <integrity level, integrity category> such as introduced by Biba [13]. The lattice on the access classes is defined as the Cartesian product of lattices on the individual components. Note that when integrity is integrated with secrecy, integrity levels are ordered in reverse so that $L1 > L2$ means that access class $L1$ has a higher secrecy component but lower integrity component that access class $L2$. This is because a user is permitted to read down in secrecy but up in integrity, and write up in secrecy but down in integrity.

### B. Relational Data Model

We shall develop our concept of secure data views in terms of the relational data model. The *relational data model* consists of *relations* (also called *tables*) together with a *relational algebra* for defining new relations in terms of other relations (the relational model also includes entity and referential integrity rules that govern the existence of certain records; these are not relevant to the concepts discussed here). Each relation $R$ is defined by a *schema* $R(A1, A2, \cdots, Ak)$ that specifies a set of *attributes* $A1, A2, \cdots, Ak$. The relation consists of a set of *records* (also called *tuples* or *rows*), where the elements in a record have data values in the domains of the attributes. The relational algebra consists of operators for selecting whole or partial records having certain values from relations and for joining data in different relations.

To illustrate our database concepts, we introduce as an example a Flight database. The database is defined by the following schemas, which specify a relation ITEM giving item numbers, names, and weights; a relation FLIGHTS giving flight numbers, departure dates, destinations, and total cargo weight; and a relation PAYLOAD giving the quantity of items on board the flights:

ITEM(ITEM#, ITEMNAME, WEIGHT)
FLIGHTS(FLIGHT#, DATE, DEST, WEIGHT)
PAYLOAD(FLIGHT#, ITEM#, QTY, WEIGHT)

The set of schemas defining the relations in the database is itself represented as a relation: RELATIONS (RELNAME, ATTRNAME), which contains an entry for each attribute of each relation (sometimes two relations are used, one for the relation names and the other for the attributes). For example, the entry (viz. tuple) <FLIGHTS, WEIGHTS> specifies that the FLIGHTS relation contains the attribute WEIGHT. The RELATIONS relation may include other attributes—e.g., for specifying domain type.

### C. Multilevel Relations

To deal with multilevel security, we provide the abstraction of multilevel relations, with the assignment of access classes down to the element level. Our preliminary design actually supports multilevel relations at the view layer only; hence the multilevel relations would more appropriately be called *multilevel virtual relations*. The multilevel virtual relations will be mapped onto single-level base relations, which in turn are mapped onto single-level segments or files. Because this mapping will be transparent to the user, we will continue to regard relations as being multilevel. Our preliminary design is discussed in greater detail in Section VI.

When an access class is defined for an entire attribute (or record) of a relation, that access class applies to each data element associated with the attribute (or record) rather than to the attribute (or record) as an aggregate. Moreover, requiring that all data elements associated with a particular attribute have the same access class does not necessarily imply that the name of that attribute need be at the same access class; the access class associated with the attribute name is determined by the access class of the attribute's entry in the RELATIONS relation. Thus, the name of an attribute can have an access class dominated by the access classes of the values associated with the attribute, in order to allow a user to see the relation schema but not its data (e.g., to insert tuples into the relation).

In later sections, we shall illustrate how access classes can be associated with our Flight database. We shall assume that all access classes may be 4-tuples < secrecy level, secrecy category, integrity level, integrity category >, but make the simplifying assumption in the illustrations that all but the secrecy level are fixed; thus each access class will be specified simply as TOP-SECRET, SECRET, etc. By holding the integrity component fixed, our examples will be free of possible integrity violations. (In addition to the mandatory integrity policy embodied in the access classes, we include a policy for *database integrity*, including integrity constraints, concurrency controls, and trusted recovery [14].)

### D. Database

In the relational model a *database* is a finite set of named relations. The data in a relation can be represented either physically as stored data or logically as a derivation rule that defines how the data is computed (derivation rules, which are arbitrary formulas in the query language, are described later).

The reason for modeling derived data is that it allows interdependencies and inference rules among stored and derived data to be expressed within a single framework. For example, consider the attribute WEIGHT from the ITEM relation and the attributes QTY and WEIGHT from the PAYLOAD relation. Suppose that PAYLOAD.WEIGHT is derived data, defined in terms of the join of the two relations as follows (using a SQL-like notation where ":=" means assignment and "=" means comparison):

```
PAYLOAD.WEIGHT :=

        ITEM.WEIGHT * PAYLOAD.QTY
    where ITEM.ITEM# = PAYLOAD.ITEM#
```

Now, the relationship among ITEM.WEIGHT, PAYLOAD.QTY, and PAYLOAD.WEIGHT places constraints on how the data are classified; for example, if PAYLOAD.WEIGHT is regarded as TOP-SECRET, then it would not be secure to classify both ITEM.WEIGHT and PAYLOAD.QTY as SECRET since a user with a SECRET clearance could access these attributes and deduce PAYLOAD.WEIGHT. More generally, it would not be secure to classify any two of the attributes lower than the third, and this is true regardless of whether PAYLOAD.WEIGHT is actually stored in the database. We will show how this problem is handled later.

As another example, suppose that a sum is taken over all records in a relation, where the individual elements used to compute the sum are all SECRET, and that it is desired to release the sum at a lower level, say CONFIDENTIAL, on the grounds that it sufficiently sanitizes the individual elements. Here again, the decision whether the sum can be marked down depends only on the inferences that can be drawn from it, and not on whether it is actually stored in the database.

It is not necessary or indeed practical to represent all conceivable derivations in the database schema—that is, the database need not represent the inferential closure. As in the relational model, users can compute their own views of the data represented in the database using the query language.

### E. Views

A *view* is a mapping (multivalued function) from a database (set of relations) to a relation (or set thereof). For security, we are particularly concerned about the subset of data in the database that is used to compute the resulting relation. We shall call this subset of elements the *view source*. In general, the source will consist of those data elements whose attributes are named (explicitly or implicitly) in the view mapping and whose tuples are selected by conditions in the mapping. The elements in the source may be joined and manipulated (e.g., by numerical operators) to compute the result. We call this result the *view target*. We require that a view target correspond to a complete or partial relation (or set thereof) in the database—that is, that the attributes and tuples in the result correspond to (stored or derived) attributes and tuples in the database (the correspondence need not necessarily be 1-1). The target of a view can overlap with its source.

For a view $V$, we will express the derivation *source* → *target* in an SQL-like query language that includes the relational and arithmetic operators. We can express our definition of PAYLOAD.WEIGHT as a view as follows:

```
view PAYLOAD.WEIGHT :=

        ITEM.WEIGHT * PAYLOAD.QTY
    where ITEM.ITEM# = PAYLOAD.ITEM#
```

Here the source is all elements associated with the attributes ITEM.ITEM#, ITEM.WEIGHT, PAYLOAD.ITEM#, and PAYLOAD.QTY. The target is the elements associated with PAYLOAD.WEIGHT.

A view may have parameters that bind to data in the domains of the attributes. (At least initially, we have made the simplifying assumption that parameters cannot bind to a relation or attribute name in the schema or to a function.) The following definition for a view HEAVIER-THAN($x$) specifies as its source and target all records in the ITEM relation where WEIGHT $>$ $x$ for parameter $x$.

```
view HEAVIER-THAN(x) :=ITEM.all
    where ITEM.WEIGHT > x
```

(The parameter $x$ is bound to an actual value at the time the view is evaluated—e.g., when a query on the view is made. The notation "$R$.all" means all attributes in relation $R$.)

The term "view" is sometimes used to refer to the actual data returned when a view specification is applied to an instance of the database. Here, we shall reserve the term "view" for the specification or mapping function, and use "view instance" or simply "data" to refer to the data bound to a view at view application time.

Every view $V$ is defined by a view specification or definition, which has an access class. If the access class of the view specification is not dominated by the access class of a user, then that view cannot be applied by that user. This has consequences for views that are used to label data, as we shall discuss later.

We shall use views for two distinct purposes: labeling new data with an access class and accessing data. Views for labeling new data are called classification constraints, while views for accessing data are called access views. First we shall describe views for labeling data. Next we shall describe views for accessing data. Then we shall return to the aggregation problem.

## III. Labeling Data with Access Classes

Data entering the database as inserts or updates are assigned an access class according to a set of classification constraints.

### A. Classification Constraints

A *classification constraint* specifies an access class to be assigned to all data in its target. The access class can be expressed as a constant or as a formula over the access classes of the data in the source using the lattice operators "$\oplus$" (for least upper bound) and "$\otimes$" (greatest lower bound). The set of all classification constraints is denoted by *CC*.

We plan to support the following types of classification constraints (or combinations thereof):

- *Type-Dependent*—The access class of a data element is determined solely by its type (i.e., the attribute with which it is associated).
- *Value-Dependent*—The access class of a data element depends on its value or on the value or access class of another element or tuple in the database.
- *Source-Level*—The access class of a data element is the access class of the user or information source.
- *Source-Label*—The access class of a data element is obtained from the security label provided by the source or specified by the user entering the data.

Whenever a classification constraint yields an access class that is lower than (i.e., strictly dominated by) the user's login class, then assigning that class reflects a downgrade operation. Requiring user confirmation of the assigned access class via a "trusted path" [1] can provide additional assurance that this downgrade is correct.

The following classification constraints on the Flight database specify that all data associated with the attributes FLIGHT#, ITEM#, and QTY of relation PAYLOAD are to be classified SECRET, whereas all data associated with the attribute WEIGHT is to be classified TOP-SECRET (i.e., classification is at the attribute level as in the Hinke–Schaefer [15] design).

```
classification constraints on PAYLOAD
  FLIGHT#, ITEM#, QTY
    class SECRET
  WEIGHT
    class TOP-SECRET
```

This is an example of type-dependent classification constraints.

Value-dependent classification constraints provide a means of classifying data at the tuple or element level, where the classification is context- or content-dependent (or, more generally, dependent on any information in the database). For example, the following content-dependent constraints classify the records in the FLIGHTS relation at either SECRET or TOP-SECRET depending on their destination:

```
classification constraints on FLIGHTS
  ALL
    class TOP-SECRET where FLIGHTS.DEST = 'Iran'
    class SECRET where else
```

Assuming we do not want to make this classification rule available to SECRET users, the rule must be classified TOP-SECRET. Doing so, however, means that an additional rule, at the SECRET level, is needed in order that a SECRET user can insert new records into the FLIGHTS relation. Such a constraint might specify that all records are to be classified SECRET:

```
classification constraint on FLIGHTS
  ALL
    class SECRET
```

Consequently, if a SECRET user inserts a record with DEST = 'Iran', then that record will be classified SECRET. If the SECRET user attempts to insert a record with a primary key (FLIGHT#) corresponding to an existing TOP-SECRET record, the "duplicate" record will be inserted into the relation. This is because the existing record must be completely invisible to the user and all database subjects operating on the user's behalf in order to avoid introducing covert channels. This does not, however, violate the requirement for "no duplicate keys" because the concept of key can be extended to include its access class. We use the term *polyinstantiation* to refer to the extension of object name to include its access class [16].

With value-dependent classification constraints, the access class for new data can depend on the access class of related data in the database. Given the preceding constraints, the following shows how data in the PAYLOAD relation can be classified according to the access class of the flight they are associated with:

```
classification constraint on PAYLOAD
  ALL
    class FLIGHTS.FLIGHT#.class
      where PAYLOAD.FLIGHT# = FLIGHTS.FLIGHT#
```

This classification constraint assigns the access class associated with a flight (denoted by FLIGHTS.FLIGHT#.class) to all fields of all payload records for that flight (assuming that the FLIGHTS relation has a record for that flight that is visible to the user inserting the payload record).

It is often desirable to insert data into a database where the access class associated with the incoming data is supplied with the data. For example, items may be inserted in the ITEM relation through messages that contain an item number, name, weight, and access class for the item (i.e., for the entire record). This would be done with a source-labeled classification constraint.

We shall now discuss two special types of classification constraints, derivation rules and sanitization rules.

### *B. Derivation Rules*

A *derivation rule* specifies how derived data are computed. Because derived data generally reveal information about the source data, each derivation rule is also interpreted as a classification constraint, where the access class associated with the derived data is computed as the least upper bound of the access classes of all data in the source (derivations that sanitize their source are described in the next subsection).

Let *DR* be the set of all derivation rules; since derivation rules are classification constraints, we have $DR \subseteq CC$. For a derivation rule $D \in DR$, the access class of the derived data is expressed by the following axiom:

*Derivation Axiom:*

$$\forall y \in D.target,\ y.class = \oplus \{x.class | x \in D.source\}.$$

The following is a derivation rule that classifies PAYLOAD.WEIGHT:

```
derivation rule on PAYLOAD
  PAYLOAD.WEIGHT := PAYLOAD.QTY * ITEM.WEIGHT
    where PAYLOAD.ITEM# = ITEM.ITEM#
```

The access class for each element of PAYLOAD.WEIGHT is computed to be:

⊕{ITEM.WEIGHT.class, PAYLOAD.QTY.class, ITEM,ITEM#.class, PAYLOAD.ITEM#.class}.

The access class computed by a derivation rule is used to attach a convenience label to derived data presented to a user and to locate certain inference problems (see Section III-E). If derived data is stored back into the database, then the computed access class can be assigned to the stored data only if: 1) it dominates the access class associated with the user on whose behalf it was computed; *or* 2) the downgrade is authorized and confirmed by the user.

### *C. Sanitization Rules*

A *sanitization rule* is like a derivation rule, except that it sanitizes its source so that the access class of the target can be strictly dominated by the least upper bound of the source access classes. Let *SR* be the set of all sanitization rules; because sanitization rules are classification constraints, $SR \subseteq CC$. For $S \in SR$, the following axiom states that the access class assigned by the constraint to the target data must be at least the least upper bound of the access classes of all data that can be inferred from the target about the source, but at most the least upper bound of the access classes of all data in the source:

*Sanitization Axiom:*

$$\forall y \in S.target,\ y.class \geq \oplus \{x.class | x \in S.source \wedge S.target \rightarrow x\},$$
$$\text{and } y.class \leq \oplus \{x.class | x \in S.source\}$$

where $y \rightarrow x$ means that $x$ can be inferred from $y$. Inference can be defined in several ways. The following uses classical information theory (the basics are in Denning [6]) to state that the relative equivocation (reduction in uncertainty) of $x$ given $y$ is at least $h$ for some threshold $h \in [0, 1]$:

$$y \rightarrow x = [requiv(y, x) \geq h],$$

where

$$requiv(y, x) = \frac{H(x) - H_y(x)}{H(x)};$$

$H(x)$ is the uncertainty of $x$ (in bits) and $H_y(x)$ is the uncertainty of $x$ given $y$ (equivocation). If $y$ discloses no information about $x$, then $H_y(x) = H(x)$, and $requiv(y, x) = 0$; if $y$ discloses the exact value of $x$, then $H_y(x) = 0$, and $requiv(y, x) = 1$; thus, the value of *requiv* is always between 0 and 1. Setting $h = 1$ implies that exact disclosure is required for inference. The relation $y \rightarrow x$ could also be formulated in terms of logic or confidence intervals. The proof that a sanitization axiom is satisfied requires a demonstration that the assigned access class is correct with respect to the given definition of inference.

The following is an example of a sanitization rule for attribute FLIGHTS.WEIGHT, which represents the sum of all payload weights for a given flight:

```
sanitization rule on FLIGHTS
  FLIGHTS.WEIGHT := sum(PAYLOAD.WEIGHT)
    where count(PAYLOAD.ITEM#) > 10
      and FLIGHTS.FLIGHT# = PAYLOAD.FLIGHT#
  class ⊕ {FLIGHTS.FLIGHT#.class, PAYLOAD.FLIGHT#.class, SECRET}
```

The above rule sanitizes the values of PAYLOAD. WEIGHT, thereby excluding the attribute PAYLOAD. WEIGHT from the access class specification (all data in this example are classified at the attribute level). The access class specification includes all other attributes in the source (even though the total weight discloses no information about flight numbers), and also states that the resulting access class will be at least SECRET. The clause "count(PAYLOAD.WEIGHT) > 10" ensures that a minimum number of items is on board each flight so that no single weight can be inferred from the sum.

A common form of sanitization involves removing the precision from sensitive data. Consider a relation schema:

```
    LOCATION(OBJECT, LONG, LAT,
             GROSS-LONG, GROSS-LAT)
```

where LONG and LAT give geographical longitude and latitude to 8 significant digits and GROSS-LONG and GROSS-LAT give only 2 digits of precision. Suppose that LONG (and LAT) are SECRET. The following shows how GROSS-LONG could be defined CONFIDENTIAL through a sanitization rule:

```
sanitization rule on LOCATION
   GROSS-LONG := round(LONG,6)
   class CONFIDENTIAL
```

where round$(x, d)$ rounds $x$ by removing $d$ digits of precision.

The following is another example of sanitization. Suppose that $A$ and $B$ are large SECRET numbers and that $C$ is the ciphertext encryption of plaintext $A$ under SECRET key $B$, where encryption is multiplication modulo a prime. Then $C$ could be released as UNCLASSIFIED through a sanitization rule.

### D. Consistency and Completeness

The set $\mathcal{CC}$ of all classification constraints must be complete and consistent. A set of classification constraints is said to be *complete* if an access class is defined for each element; it is *consistent* if no two constraints, both of which must be satisfied simultaneously, define conflicting classes. We are presently investigating the conditions under which completeness and consistency can be determined by analyzing the constraints with respect to the database schema, and algorithms for doing so.

### E. Derivation Rules as an Inference Tool

Although derivation rules specify an access class for derived data, an access class for derived data may also be specified by a classification constraint that reflects what its perceived classification should be. The two constraints can then be analyzed for consistency to determine whether there is an inference problem associated with the derived data.

For example, consider again the derivation rule for PAYLOAD.WEIGHT, which we shall write in brief as PAYLOAD.WEIGHT = ITEM.WEIGHT * PAYLOAD.QTY, ignoring for the moment the fact that both ITEM.ITEM# and PAYLOAD.ITEM# are included in the source in order to do the join. Let us further abstract this to $C = A*B$. Consider the following classification constraints and derivation rules:

```
classification constraints on R
A, B
class SECRET
  C
  class TOP-SECRET

derivation rule on R
C := A * B
```

Here the user has added the additional classification constraint for $C$ (which is not needed for completeness because a label for $C$ is computed from the derivation rule). Now, because the first two classification constraints label $A$ and $B$ as SECRET, the derivation rule for $C$ will label $C$ as SECRET, thereby conflicting with the classification constraint for $C$, which specifies that $C$ is TOP-SECRET. This inconsistency reveals an inference problem: if $C$ is TOP-SECRET, then simply labeling it as TOP-SECRET is not enough because it can be derived from $A$ and $B$. The user might then redefine the constraints so that at least one of $A$ and $B$ is TOP-SECRET.

Returning to our Flight database, the situation is somewhat more complicated because the join attributes ITEM.ITEM# and PAYLOAD.ITEM# are also in the source for PAYLOAD.WEIGHT. If either of the join attributes are classified higher than ITEM.WEIGHT and PAYLOAD.QTY, then PAYLOAD.WEIGHT will be forced to the higher access class. This could be avoided by expressing the derivation as the following sanitization rule (and demonstrating that ITEM# cannot be inferred from PAYLOAD.WEIGHT):

```
sanitization rule on PAYLOAD
  PAYLOAD.WEIGHT := PAYLOAD.QTY * ITEM.WEIGHT
    where PAYLOAD.ITEM# = ITEM.ITEM#
  class ⊕ {ITEM.WEIGHT.class, PAYLOAD.QTY.class}
```

lation, be stored at relation-low. This means that a user knows in advance that there may be invisible data associated with an attribute.

Mandatory security, namely the simple security and *-properties [20], will be enforced in the reference monitor (see Section VI).

## IV. Views for Accessing Data

Data in the database are accessed through access views, which control database retrieval and update.

### A. Access Views

All accesses to the database for retrieval, update, insert, and delete are controlled by a set $AV$ of views called *access views*. Because access views are generally referred to as "views" in the literature, henceforth we shall refer to them simply as views. A view specifies some subset of the database and can also specify computations. The view specification has a classification, and the view has an associated authorization list for discretionary security.

For a view $V \in AV$, the security requirements are stated in terms of two axioms. The first deals with mandatory security, and the second with discretionary security.

### B. Filtering Axiom

The first axiom for access views specifies the filtering requirement: that all data bound to the source of $V$ (when the view is evaluated) are dominated by the login access class of the user $U$, denoted $U.class$. As previously noted, restricting the classification of the source data is essential in order to prevent user inferences [17]–[19].

*Filtering Axiom:*

$$U.class \geq x.class, \quad \forall x \in V.source.$$

In practice, the filtering logically removes all data from the view source that is not dominated by $U.class$ as follows:

1) If no value for an attribute is dominated by $U.class$, then the attribute is deleted.

2) If no value for an entire record is dominated by $U.class$, then the record is deleted.

3) If, after removing attributes and records, an element remains that is not dominated by $U.class$, the element is replaced by *nil* (meaning "undefined") and assigned the access class UNCLASSIFIED.

Note that the replacement of classified data by *nil* values does not completely hide the data in that the existence of data at higher access classes is disclosed (although it may not be always possible to distinguish between hidden data and other undefined data). However, such hiding does not provide a leakage path, because we require that the relation schema, including a specification for the range of access classes that can be assigned to elements in the relation, be stored at relation-low.

The user's clearance, of course, must dominate the login access class:

*Login Access Class Axiom:*

$$U.clearance \geq U.class.$$

A user may update multilevel data through a single view. For example, a TOP-SECRET user could update a tuple containing both SECRET and TOP-SECRET attributes. Although permitting multilevel updates is highly desirable, it also introduces some risk. One way of controlling this risk is to limit the range over which a multilevel update can be performed (e.g., to two adjacent access classes); another is to require that all such updates originate from the user via a "trusted path" (rather than from software).

When a query result is displayed to the user, the display will include the following as convenience labels: the security markings for all source data; the labels computed by all derivation rules; and a label for the query result as an aggregate, computed as the least upper bound of the access classes for all data accessed during the processing of the query. Because the system adheres to the *-property, these are convenience labels only, and the query result must be protected at the access class of the user (which may be higher than any of the computed labels). This is because the database subjects running on behalf of the user inherit the user's access class and, therefore, have access to data in that access class.

### C. Discretionary Access

The second axiom on access views states that $U$ can perform an operation $o$ on $V$ only if $o$ is specifically authorized to $U$ on $V$:

*Discretionary Access Axiom:*

$$access(U, V, o) \supset auth(U, V, o).$$

In addition to the above axioms, we require that all views used for update, insert, and delete be well-defined—i.e., that the data in the target can be mapped back onto the source. In general, this requires that the target for each such view be stored (rather than derived) data and include a key for each relation named.

Views need not be revoked when a user's clearance changes (revocation is needed only if the discretionary policy changes), since the reference monitor, which enforces mandatory security, will not return any data to a user through a view unless the access class of the user dominates the access class of the data, and will not acknowledge the existence of a view to a user unless the access class of the user dominates the access class of the view specification. To illustrate, consider the following SECRET view, which retrieves the flight number and date of all flights:

```
view FLIGHT-DATES
   FLIGHT# := FLIGHTS.FLIGHT#
   DATE := FLIGHTS.DATE
   class SECRET
   auth retrieve
```

Because the view definition is SECRET, users with clearances less than SECRET will not have access to the view. Moreover, SECRET users retrieving data through the view will not have access to any TOP-SECRET data. In particular, under the classification constraints on FLIGHTS given earlier, all flights to Iran will be invisible to SECRET users. The "auth" specification states which operations are permitted to which users; for simplicity, we have stated operations only. A user authorized to use this view could issue a retrieval against it—for example,

```
retrieve DATE from FLIGHT-DATES
      where FLIGHT# = 1735
```

A view can contain predicates that depend on arbitrary states of the database. For example, suppose that our database includes a single-record relation STATE with attribute TIME giving the current time, and a relation USER with attribute UID giving the user identifier for each logged-in user, and an attribute CONNECT giving the type of login connection for the user. Letting "*" denote the user id of the user applying a view, the preceding view could be written as follows in order to require that all accesses be from local terminals during the hours 8:00 AM to 5:00 PM:

```
view FLIGHT-DATES
  FLIGHT# := FLIGHTS.FLIGHT#
  DATE := FLIGHTS.DATE
    where STATE.TIME >= 0800
      and STATE.TIME <= 1700
      and USER.CONNECT = 'local'
      and USER.UID = *
class SECRET
auth retrieve
```

Note that we include all access predicates in the view rather than expressing them separately through special access rules as is done in INGRES [7] and by Bonyun [21].

For many applications, users want data presented graphically or statistically, or they want some complex calculation to be performed over the data. Users may compose one or more views and use numeric and statistical operators and operators for formatting and display. Many of these operators (e.g., those used for formatting) will be invisible to the user and implicit in the *retrieve* command. In general, users are free to form queries across views for which they are authorized.

## V. Aggregation

We now turn to the *aggregation problem*, that is, restricting access to collections of data beyond that required for the individual elements. The aggregation problem arises when data are brought together through associations to provide a larger context. Although one could in principle define an aggregate to be an arbitrary set of data, the cases of practical interest seem to fall into the following categories:

*1) Qualitative Associations:* These correspond to associations among different attributes of the schema or among different instances of the same attribute, and fall into two subcategories:

*a) Entity Associations:* Here, it is the association between different entities (as represented by relations) that is sensitive. Examples are the association between the source and receiver of a transmission, and between a flight and item (meaning the item is in the flight's payload).

*b) Property Associations:* Here, associations among some of the properties (as represented by attributes) of a single entity are more sensitive than the properties taken separately. An example is an association between longitude and latitude for an object, which might be more sensitive than either coordinate alone.

*2) Quantitative Associations:* These correspond to size-based associations where any collection of more than $n$ tuples over the same attributes is considered to be more sensitive than a single tuple. For example, an insurance agent might be allowed to retrieve the dollar value of individual insurance policies, but is not allowed to retrieve more than $n$ of them. Or, a particular cashier might be authorized to write checks drawn on the company's account, but may not write checks totaling more than $n$ dollars in any one day.

In the following sections, we discuss three methods for dealing with aggregation problems: 1) database design, 2) sanitization, and 3) discretionary aggregation constraints.

### A. Database Design

Many qualitative associations can be handled through database design. Consider first associations among entity types, where each entity type is represented by a relation. Here, we can represent the association either by a separate relation that contains join attributes for both relations, or by join attributes that are stored in either or both of the entity relations. In general, these join attributes will be primary or secondary keys, but entities can be related by non-key attributes as well. If the join attributes are stored in the entity relations, then the association can be protected by classifying at least one of each pair of attributes used for joining the relations at the higher access class. Where the association is represented entirely by a separate relation (i.e., no meaningful associations can be made by joining the relations for the separate entities), the entire relation may be classified up.

To illustrate how an association can be protected, suppose in the Flight database that all flights and items are classified SECRET, but that the association between flights and items is to be regarded as TOP-SECRET. Since this association is represented entirely by the PAYLOAD relation (no meaningful associations can be made by joining the attributes in FLIGHTS and ITEM), the problem is readily solved by classifying the entire PAYLOAD relation TOP-SECRET. (Classifying only PAYLOAD.ITEM# as TOP-SECRET would not solve the problem since the result of PAYLOAD.WEIGHT/PAYLOAD.QTY could be joined with ITEM.WEIGHT to get ITEM#).

Associations among different instances of the same entity type can be similarly handled by classifying the association at the higher access class. To illustrate, suppose that knowing that employees Smith, Jones, and Davis are working together on a project may reveal SECRET information about the mission of the project, although other project and employee information may be CONFIDENTIAL. We can store employee information in a CONFIDENTIAL relation EMPLOYEES(EMPLOYEE-ID, NAME, DEPT-NO, TITLE), and we can store project information in a CONFIDENTIAL relation PROJECTS(PROJECT-ID, PROJECT-NAME, BUDGET). EMPLOYEES includes tuples < 12345, Smith, 15, Cartographer >, < 14000, Jones, 9, Driver >, and < 10800, Davis, 11, Explosives-Expert >. PROJECTS includes tuples < 21, Product-X, 250000 > and < 19, Special-Deliveries, 500000 >. The association of employees and the projects they are assigned to can be stored in the relation WORKS-ON(PROJECT-ID, EMPLOYEE-ID), where classification can be at the tuple level. If WORKS-ON has all three of the tuples < 19, 12345 >, < 19, 14000 >, and < 19, 10800 > (indicating that Smith, Jones, and Davis are all working together on the Special-Deliveries project), then these three tuples would be classified SECRET, whereas the other tuples in WORKS-ON would be classified CONFIDENTIAL (unless there were other projects that Smith, Jones, and Davis work together on).

### *B. Sanitization*

Sanitization rules are required for most property associations and quantitative associations. With respect to property associations, we note that simply assigning a higher access class to one or more of the attributes forming the aggregate often does not address the problem. For example, consider a relation $R(ID, A, B, C)$ with key $ID$, where $ID$, $A$, $B$, and $C$ are each SECRET, but the association $(A, B)$ is TOP-SECRET. We could split up $A$ and $B$ by storing one—say $A$—in a separate relation $(ID, A)$, with the join attribute $ID$ labeled as TOP-SECRET. However, this would also prevent making SECRET associations between $A$ and $ID$ and $A$ and $C$. We have the same problem if we simply classify attribute $A$ higher in the original relation $R$. Such cases are better dealt with as sanitization problems, using sanitization rules. With this approach, we protect the aggregate at the higher access class, and require a sanitization rule to release subsets of the aggregate to users at lower access classes. (In this case, we require a proof that the sanitization is sufficiently lossy with respect to the whole rather than to the individual elements.) This has the additional advantage that the reference monitor can provide mandatory security for the aggregate, even though the reference monitor does not know about aggregate definitions made at the view layer. If the aggregate were not protected at the higher access class, then the aggregate would be vulnerable to unauthorized disclosure if the view manager could be circumvented.

Similarly, quantitative associations can be handled by sanitization. To illustrate, consider the familiar "phone book" problem, where the entries in the phone book for a well-known agency are at one access class, but the entire phone book, or even a set of more than $n$ entries from the book, is at a higher access class. Here we protect the entire phone book at the higher access class (its actual classification) and require sanitization to release an individual phone number to users at lower access classes. This corresponds to how the phone book is treated in the manual world.

### *C. Aggregation Constraints*

Aggregation constraints are used for qualitative and quantitative associations of a discretionary nature when the aggregation problem cannot be solved through design alone.

An *aggregation constraint* $W$ is a view that is defined by an attribute set, $W.attset$, and a count, $W.count$, which specifies a lower bound on the number of tuples over the attributes in $W.attset$ required to form the aggregate. A relation *aggregate-restrict* specifies what users are individually or collectively restricted access to an aggregate. In particular, a user $U$ is restricted access to an aggregate $W$ if $U$ belongs to a group $G$ where *aggregate-restrict* $(G, W)$. Restrictions are placed on individual users by forming singleton groups.

We assume that the scope of each aggregation constraint is minimal. This means that a minimal set of attributes is specified (i.e., any smaller set of attributes does not cause an aggregation problem), and, for quantitative aggregation, a minimal number of records is specified.

Access to an aggregate $W$ is controlled through additional restrictions on access views. For a given set of access views $VS$, we say that *$VS$ covers $W$* if and only if the set of attributes named in $VS$ includes all attributes in $W.attset$. A set of view instantiations, denoted $VI$, *contains* $W$, written $VI \supseteq W$, if and only if the views in $VI$ cover $W$ and the number of tuples returned by $VI$ is at least $W.count$. The following axiom then states that if a group of users $G$ has collectively accessed an aggregate $W$, then it must not be the case that *aggregate-restrict* $(G, W)$:

*Aggregate Access Axiom:*

$$access(G, VI) \wedge VI \supseteq W \supset \neg\, aggregate\text{-}restrict(G, W).$$

As an example, consider the following aggregation constraint, which defines FLIGHT-AGG to consist of any 10 or more complete tuples in FLIGHTS:

```
aggregation constraint FLIGHT-AGG
   FLIGHTS.ALL
   count = 10
   restrict {Smith}, {Jones, Young}
```

The aggregate is restricted to Smith individually, and to Jones and Young collectively (but not to Smith and Jones collectively, for example). Neither of these two groups would be allowed to retrieve more than 10 records over time from any set of views over the entire FLIGHTS relation. However, because all attributes in FLIGHTS are needed to form the aggregate, the users could retrieve an unlimited number of records from a view that accesses, say, only the FLIGHT# and DEST fields. Records accumulated over time might be excluded from the aggregate when they become outdated (perhaps after some predetermined elapsed time).

One might want aggregation restrictions to apply to single view instantiations only, and not to accumulations over time. This option might be specified in an aggregation constraint.

Because aggregation constraints are discretionary access controls, they have the limitations of any other discretionary controls. For example, they provide little protection against Trojan Horses or collusion between users.

## VI. System Design

The preceding sections have outlined a high-level policy model for a secure database system based on views. The axioms state the security conditions that must be satisfied by the system and each instance of the database. In this section, we give a rough sketch of a tentative system design to support our view concepts.

We decompose the system into a set of $k$ *layers*. Each layer $i$ builds on lower layers and implements a *security policy* $P_i$, which is a set of triples $(U, I, o)$ specifying that user $U$ can perform operation $o$ on information $I$. Each layer has the property that its policy further constrains access by requiring that additional axioms be satisfied; thus, $P_i \subseteq P_{i-1}$ for $i = 2, \cdots, k$. Our formal model will parallel the design layers.

The following is a possible decomposition that takes advantage of existing kernel technology. The layers are listed from top to bottom, with the boundaries for the trusted computing base (TCB) and reference monitor as shown in Fig. 1.

Layer 1 is a *security kernel* implementing the Bell and LaPadula model [20]. The kernel supports mandatory security (secrecy and integrity) for single-level objects (such as files, etc.), subjects (users, processes, etc.), process and memory management, and other resource managers. With respect to the view model, this layer supports the underlying physical representation of the database, user clearances, and login access class. The kernel will be used to implement the higher layers. We envisage using the GEMSOS security kernel [22] in our design.

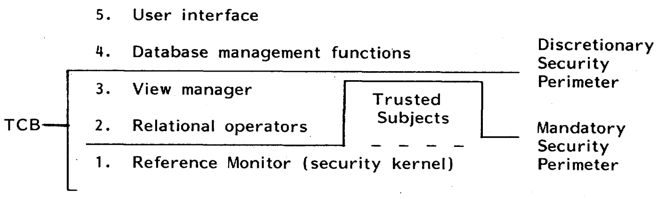

5. User interface
4. Database management functions
3. View manager
2. Relational operators
1. Reference Monitor (security kernel)
Trusted Subjects
TCB
Discretionary Security Perimeter
Mandatory Security Perimeter

Fig. 1. System decomposition.

The reference monitor enforces filtering. By filtering all data passed from this layer to higher (untrusted) layers in the database system, a Trojan Horse in the query processor cannot leak high-level data by encoding it in a low-level response [18].

Layer 2 implements single-level base relations. This layer provides the relational operators as well as other numeric or symbolic operators provided by the database system (e.g., for computing statistics) and implements the access methods, including index structures.

Layer 3, the view manager, implements views. This layer provides the mapping of multilevel views onto the single-level base relations of Layer 2, and decomposes transactions on multilevel views into single-level transactions on the single-level base relations of Layer 2. Layer 3 supports classification constraints, including derivation and sanitization rules, aggregation constraints, and discretionary access controls (e.g., access-control lists or user/group/world vectors). It is responsible for checking that a set of constraints is complete and consistent, and for assigning access classes to the data (through calls on Layer 1).

As shown in the figure, we have identified distinct security perimeters with respect to mandatory and discretionary security. The TCB encompasses both. Although classification and aggregation constraints are within the TCB, they are outside the reference monitor because they are potentially quite complex and rely to some extent on the correct implementation of the relational operators.

The mandatory security perimeter includes the reference monitor as well as "trusted subjects" to perform downgrades arising from sanitization and rule-based classification (specifically, to allow data to be assigned an access class that is lower than that of the user entering the data). The trusted subjects will require a demonstration that information at a high access class cannot leak through them to a low access class. For downgrades arising from the classification constraints, user confirmation of the access classes to be assigned to data may also be required for increased assurance. For sanitization, we may restrict the data that can be retrieved and limit the amount by which the data can be marked down. We will also have to show that the sanitization rules lose sufficient information about the source.

The discretionary security perimeter includes everything through Layer 3, the view manager. Although Layer 3 cannot compromise the *mandatory* secrecy and integrity provided by Layer 1, it is still a part of the TCB. (Everything above Layer 3 is untrusted in the sense of the TCB, although user application environments at Layer 5 may themselves enforce more refined application policies.)

Because Layer 3 is constrained by the reference monitor of Layer 1, which ensures enforcement of the mandatory policy, the view manager will not require the same degree of assurance as the reference monitor.

Layer 4 provides data management functions that are not needed by the lower layers. Although we are not yet sure what functions can reside in this layer, possible candidates are transaction management, parsing and query optimization, and distributed database management. Our goal is to place as much of the database system as possible outside the TCB.

Layer 5 provides the user interface. It corresponds to the application layer.

Each successive layer in the TCB (through Layer 3) introduces further axioms, thereby restricting while enriching the policy of the system. Although the higher layers cannot give increased access to the data beyond that given by the reference monitor, by supporting classification down to the element level, rule-based assignment of security labels, sanitization, derivation rules, and aggregation constraints, the higher layers can provide increased support for security while allowing data to be classified at its true access class and not become over-classified.

## VII. Conclusions

The concepts described in this paper represent a starting point for a model that reaches into new areas of database security. By introducing rule-based classification to constrain the access classes of the data, the model provides a framework for addressing some inference problems, aggregation, sanitization, context- and content-dependent classification, and dynamic classification. There are, however, many difficult problems that must be solved before these concepts can be realized in an actual system.

We are now formalizing some of these concepts as a security model. We expect that in the process of formalization, the model will undergo revisions and enhancements as we find better ways of expressing the basic concepts and uncover implementation problems. Our goal is to develop a model and system design that can be implemented in a three to five year period.

## References

[1] *Department of Defense Trusted Computer System Evaluation Criteria*, Dep. Defense, National Computer Security Center, Standard DOD 5200.28-STD, 1985.

[2] E. B. Fernandez, R. C. Summers, and C. Wood, *Database Security and Integrity*. Reading, MA: Addison-Wesley, 1981.

[3] E. F. Codd, ''A relational model for large shared data banks,'' *Commun. ACM*, vol. 13, no. 6, pp. 377-387, June 1970.

[4] P. P. Griffiths and B. W. Wade, ''An authorization mechanism for a relational database system,'' *ACM Trans. Database Syst.*, vol. 1, no. 3, pp. 242-255, Sept. 1976.

[5] C. J. Date, *An Introduction to Database Systems*, vol. II. Reading, MA: Addison-Wesley, 1983.

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

[7] M. Stonebraker and E. Wong, ''Access control in a relational data base management system by query modification,'' in *Proc. 1974 ACM Annu. Conf.*, Nov. 1974, pp. 180-186.

[8] P. G. Neumann, R. S. Boyer, R. J. Feiertag, K. N. Levitt, and L. Robinson, ''A provably secure operating system: The system, its applications, and proofs,'' Comput. Sci. Lab., SRI International, Menlo Park, CA, Tech. Rep. CSL-116, 2nd ed., May 1980.

[9] B. G. Claybrook, ''Using views in a multilevel secure database management system,'' in *Proc. 1983 Symp. Security and Privacy*, IEEE Comput. Soc., Apr. 1983, pp. 4-17.

[10] D. E. Denning, ''Multilevel secure database systems: Requirements and model,'' NAS/AFSB Summer Study on Multilevel Database Management Security, Working Paper, June 1982.

[11] Committee on Multilevel Data Management Security, ''Multilevel data management security,'' Air Force Studies Board, National Research Council, National Academy Press, Tech. Rep. 1983 (for official use only).

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

[13] K. J. Biba, ''Integrity considerations for secure computer systems,'' USAF Electronic Systems Division, Bedford, MA, Tech. Rep. ESD-TR-76-372, Apr. 1977.

[14] R. R. Schell and D. E. Denning, ''Integrity in trusted database systems,'' in *Proc. 9th Nat. Comput. Security Conf.*, 1986.

[15] T. H. Hinke and M. Schaefer, ''Secure data management system,'' System Development Corp., Tech. Rep. RADC-TR-75-266, Nov. 1975.

[16] T. F. Lunt, D. E. Denning, R. R. Schell, and M. Heckman, ''Polyinstantiation in a secure relational database system,'' SRI International, Tech. Rep., May 1986.

[17] D. E. Denning, ''Cryptographic checksums for multilevel data security,'' in *Proc. 1984 Symp. Security and Privacy*, IEEE Comput. Soc., 1984, pp. 52-61.

[18] D. E. Denning, ''Commutative filters for reducing inference threats in multilevel database systems,'' in *Proc. 1985 Symp. Security and Privacy*, IEEE Comput. Soc., 1985.

[19] R. D. Graubart, ''The integrity-lock approach to secure database management,'' in *Proc. 1984 Symp. Security and Privacy*, IEEE Comput. Soc., 1984, pp. 62-74.

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

[21] D. A. Bonyun, ''Rules as the basis of access control in database management systems,'' in *Proc. 7th DOD/NBS Comput. Security Conf.*, 1984, pp. 38-47.

[22] R. R. Schell and T. F. Tao, ''Microcomputer-based trusted systems for communication and workstation applications,'' in *Proc. 7th DOD/NBS Comput. Security Conf.*, Sept. 1984, pp. 277-290.

**Dorothy E. Denning** received the B.A. and M.A. degrees in mathematics from the University of Michigan, Ann Arbor, and the Ph.D. degree in computer science from Purdue University, West Lafayette, IN.

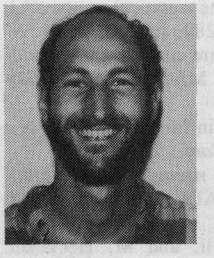
She is a Senior Computer Scientist at SRI International, Menlo Park, CA. Her current research interests include databases, systems, and security. Before coming to SRI, she was an Associate Professor of Computer Science at Purdue University.

Dr. Denning has been President of the International Association of Cryptologic Research and Chairman of the ACM Special Interest Group on Operating Systems (SIGOPS), and is author of *Cryptography and Data Security* (Addison-Wesley, 1982).

**Selim G. Akl** (S'74-M'78-SM'85) received the B.Sc. and M.Sc. degrees in electrical engineering from the University of Alexandria, Alexandria, Egypt, in 1971 and 1975, respectively, and the Ph.D. degree in computer science from McGill University, Montreal, P.Q., Canada, in 1978.

He is currently an Associate Professor of Computing and Information Science at Queen's University, Kingston, Ont., Canada. His research interests are primarily in the area of algorithm design and analysis, in particular, for problems in computer security and parallel computation.

Dr. Akl is Editor of the *IACR Newsletter*, published by the International Association for Cryptologic Research, author of *Parallel Sorting Algorithms* (Academic), and coauthor of *The Convex Hull Problem* (Plenum, to be published). He is a founding member of the Canadian Applied Mathematics Society, and a member of the International Association for Cryptologic Research, the IEEE Computer Society, and the Association for Computing Machinery.



**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 currently is responsible for the design and implementation of system security and system administrator capabilities for the Gemini Multiprocessing Secure Operating System, in addition to participating actively in the Multilevel Secure Data Views Project.

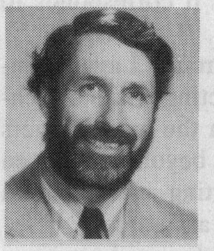
**Teresa F. Lunt** received the A.B. degree from Princeton University, Princeton, NJ, in 1976 and the M.A. degree in applied mathematics from Indiana University, Bloomington, in 1979.

She worked at The MITRE Corporation for four years and later at SYTEK's Data Security Division for two years before joining the Computer Science Laboratory at SRI in early 1986. She has worked on audit trail analysis, automated security guards, security models, and formal verification of secure systems. Her current work is in secure databases and in real-time intrusion detection expert systems.

**Matthew Morgenstern** received the B.S. and M.S. degrees in electrical engineering and computer science from Columbia University, New York, NY, the Master's degree in computer science and management, and the Ph.D. degree in computer science in 1976, both from the Massachusetts Institute of Technology, Cambridge.

Currently he is Senior Computer Scientist at SRI International, Menlo Park, CA, where he established and is project leader for the Intelligent Database Systems program. He also has served as Member of Technical Staff at Bell Laboratories, Assistant Professor of Computer Science at Rutgers University, New Brunswick, NJ, and Research Computer Scientist at USC's Information Sciences Institute, Los Angeles. His research and publications include database design, knowledge-based systems, expert systems, security, software development environments, user interfaces, and automatic programming.

Dr. Morgenstern is a member of Sigma Xi, Tau Beta Pi, Eta Kappa Nu, and the Association for Computing Machinery.



**Peter G. Neumann** (S'56–M'61–SM'85) received the A.B., S.M., and Ph.D. degrees from Harvard University, Cambridge, MA, in 1954, 1955, and 1961, respectively, and the Dr. rerum naturarum from the Technische Hochschule, Darmstadt, West Germany, in 1960.

He is Staff Scientist in the Computer Science Laboratory at SRI International, Menlo Park, CA, where he has been since 1971. He has been active in the computer field since 1953. His current research interests include computer systems, security, system vulnerabilities, and software engineering.

Dr. Neumann is Editor of the ACM SIGSOFT *Software Engineering Notes*, Chairman of the ACM Committee on Computers and Public Policy, and Coordinator of the on-line Forum on Risks to the Public in the Use of Computers.

**Roger R. Schell** (S'68–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 system programmer, and introduced the security kernel technology.