Formal Query Languages for Secure Relational Databases

MARIANNE WINSLETT
University of Illinois
and
KENNETH SMITH
Mitre Corporation
and
XIAOLEI QIAN
SRI International

The addition of stringent security specifications to the list of requirements for an application poses many new problems in DBMS design and implementation, as well as database design, use, and maintenance. Tight security requirements, such as those that result in silent masking or withholding of true information from a user or the introduction of false information into query answers, also raise fundamental questions about the meaning of the database and the semantics of accompanying query languages. In this paper, we propose a belief-based semantics for secure databases, which provides a semantics for databases that can "lie" about the state of the world, or about their knowledge about the state of the world, in order to preserve security. This kind of semantics can be used as a helpful retrofit for the proposals for a "multilevel secure" database model (a particularly stringent form of security), and may be useful for less restrictive security policies as well. We also propose a family of query languages for multilevel secure relational database applications, and base the semantics of those languages on our semantics for secure databases. Our query languages are free of the semantic problems associated with use of ordinary SQL in a multilevel secure context, and should be easy for users to understand and employ.

Categories and Subject Descriptors: H.2.0 [**Database Management**]: General—*security, integrity and protection*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Security

Additional Key Words and Phrases: Formal security models, information security, multilevel secure databases

This work has been supported by IBM, by NSF under a Presidential Young Investigator award, and by ARPA and Rome Laboratories under contract F30602-92-C-0140.

Authors' addresses: M. Winslett, Computer Science Department, University of Illinois, 1304 West Springfield Avenue, Urbana IL 61801; email: winslett@cs.uiuc.edu; K. Smith, Mitre Corporation, 7525 Colshire Drive, McLean, VA 22102; email: kps@mitre.org; X. Qian, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025, email: qian@csl.sri.com.

1. INTRODUCTION

Research into database security models is still in its infancy. Current security proposals can be broken into two groups, those that offer "discretionary" access controls (the usual type of security that comes with current-day file systems and databases) and those that impose "mandatory" controls. Recent work in role-based security for databases that need discretionary access controls is promising for many applications, although it may not by itself offer enough facilities for applications with heavy security needs. At the other end of the spectrum lies research into the tightest type of security: mandatory multilevel security. Most proposals for multilevel secure (MLS) relational databases have utilized syntactic integrity properties to control problems that arise in the presence of very tight security, such as polyinstantiation, pervasive ambiguity, and proliferation of tuples due to updates, with only partial success. We believe that many of the MLS problems can be resolved by directly addressing the question of what an MLS database *means*, rather than making syntactic adjustments to avoid semantic problems.

In the following section, we will first explain the relevance of our work for discretionary security. We then give an overview of the basic concepts of mandatory security, and explain the relevance of our work for mandatory security. Section 3 explains our formal semantics for secure databases. Sections 4 and 5 define query languages for secure databases, and show that they satisfy mandatory security. Finally, in Section 6, we summarize our results and describe future work.

2. RELATED WORK IN DATABASE SECURITY

2.1 Discretionary Security

In the past, database researchers have largely side-stepped the question of security, and commercial DBMSs have typically been able to satisfy their customers with relatively simple security facilities offered on a per-table granularity. However, this situation is likely to change in the future, as enterprises move from having local islands of database automation to having an enterprise-wide information backbone (composed of cooperating heterogeneous databases), and then to direct sharing of information between enterprises on the Internet. As the legitimate direct sharing of information increases, so does the potental for violation of privacy and inappropriate disclosure of corporate assets, and so does need for careful control of dissemination of information through security measures.

For example, consider the demands that will be placed on a hospital patient database in the relatively near future. Medical insurers in a fee-for-service scenario need the hospital's information on patient treatments in order to process claims and issue payments. In the future, direct sharing of that information could save both hospitals and insurers an enormous amount of money. However, an insurer should not have access to records of patients insured by other companies, or to other information not needed for processing its own claims.

In addition, hospitals routinely make public certain information about their patients. However, some patients may have reasons to request anonymity when they check into a hospital; for example, they may wish to avoid the scrutiny of the press. For these reasons, hospitals allow generally any patient to supply an alias, an assumed name, when they arrive at the hospital, or resort to other measures to produce a *cover story* for public consumption. A cover story is misinformation about the correct value of a set of attributes, promulgated at "lower" security levels in order to hide information known at "higher" levels. For example, a famous person being treated for a socially unacceptable ailment or procedure X might use an assumed name, or might have a more acceptable ailment or procedure Y, such as "exhaustion," be given as the cause of admission. For the cover story to hang together, however, all publicly available bits of information about the patient must be consistent. For example, if it is known that the patient is being treated by a specialist in X, or is on the X ward, then an announced diagnosis of Y will not be credible. Furthermore, the cover story is for external consumption only, and not for use inside the hospital in, for example, dispensing drugs, or in disclosure to insurance companies. Thus one version of reality is available to the public, another to hospital staff, and a subset of the latter information to outside agencies such as insurers.

When these different versions of reality are mixed together in a single database or collection of databases it is not so clear what the database really means. For example, the question *What is this patient's diagnosis*? should get different answers, depending on who is asking the question (e.g., the patient's nurse, versus a reporter). However, there is also a need to be able to ask questions relative to others' viewpoints; for example, the nurse might need to ask, *If a reporter asks for the diagnosis of this patient, what will the answer be*? to check for the presence and credibility of a cover story. The inverse question asked by a reporter, *If a nurse asks for the diagnosis of this patient, what will the answer be*? should be handled in yet another way. Our semantics for secure databases and our proposed extensions for secure query languages address these concerns by giving formal recognition to the possibility of multiple versions of reality housed in a single database (or in multiple cooperating databases).

Note that in general it is not possible to preserve privacy and maintain security by simply omitting information. For example, suppose a reporter gets a yes-or-no answer to the question *Is Jane Doe a patient at this hospital*? but in reply to the question *Is Michael Jackson a patient at this hospital*? is told *I am sorry, but I cannot divulge that information.* The absence of an answer, i.e., the omission of information, is sufficient in this case to give away information and probably compromise confidentiality. As a more abstract example, suppose a person is attempting to ascertain the value of attribute A of a tuple t in a relational database, but is not permitted to read the value. On attempted access, the DBMS might return an "access denied" message, but as described above, this approach may convey too much information about the value of t.A. As a alternative, the system might plant a cover story to hide the actual value of attribute A, by replacing the actual value of t.A by

a null value. However, this approach may also divulge too much information. The user, on seeing a null value for t.A, may try innocently or maliciously to update t to replace the null by a concrete value. If the update request is rejected, then the fact that the null really means "access denied" has been divulged. If the update request is accepted, then the DBMS has actually got to maintain two different values for t.A, so that the user will see the value for t.A that he or she expects, after the update has been committed. At that point, the DBMS is storing multiple versions of reality, and the exact meaning of the data in the system is unclear.[1]

We believe that the most promising approach to security for applications with strict security requirements is role-based security [Rabitti et al. 1991; Ting et al. 1992], in which access to data items is allowed or prohibited based on the role or roles in which a user is acting. Our eventual goal is to offer a sound semantic treatment of cover stories for a role-based system. In this paper, we take the first step toward that goal, by formalizing semantics and query languages for databases under mandatory security.

2.2 Mandatory Security

The proposals for a multilevel secure relational model [Jajodia and Sandhu 1991; Haigh et al. 1991; Denning et al. 1987] implement the policy of *mandatory* protection defined in Department of Defense [1985] and interpreted for computerized systems by Bell and LaPadula [1974]. Under mandatory protection, objects (data items) are assigned a security classification, and subjects (active processes, users) are assigned a security clearance. Classifications and clearances are both taken from a common domain of *access classes*, also known as *labels* or *levels*, which form a finite partially ordered set[2] that we will call the *security hierarchy* or, when clear from context, simply the *hierarchy*. For example, labels Top Secret (TS), Secret (S), Confidential (C), and Unclassified (U) are widely used. For two labels c_1 and c_2, if $c_1 > c_2$ in the partial order, we say c_1 is "higher than" or "above" c_2. If $c_1 \geq c_2$, we say c_1 *dominates* c_2.

The Bell-LaPadula model imposes the "no read up, no write down" restrictions on accesses by subjects. Subjects are only permitted to read from a level dominated by their own; subjects are only permitted to write to a level that dominates their own. This is sufficient to prevent subjects from passing information directly downward through the security hierarchy, as required by the mandatory policy.

MLS relational models (e.g., Jajodia and Sandhu [1991], Haigh et al. [1991], and Denning et al. [1987]) have implemented the no-read-up and

[1] The moral of this particular example may be that users should not have update authorization for attributes whose "real" values might not be visible to them.

[2] Some work in mandatory security places additional requirements on the poset, such as a requirement that it be a lattice [Denning 1976]. However, these properties are not needed in our work.

no-write-down restrictions by associating labels with the elements of a relation, such as tuples and fields. Only tuples with labels dominated by that of a user are visible to that user. Therefore different users see different versions of a relation, depending on their access class. A subject may only update, delete, or insert items having his or her own label.[3] Relations in which tuples are visible based on their access class are called MLS relations.

There are a number of similarities between roles and this system of labels and access restrictions. The security labels for subjects are somewhat like roles for users. Each of these "roles" has a fixed set of access privileges associated with it: users can write information labeled with their own label and read information with a label dominated by their own label. However, in practice, roles capture "need to know" information, and security labels capture an orthogonal kind of information. In a system offering both facilities, a user would only be able to read a data item if his or her label dominated that of the item and if he or she was in a role that had read authorization for the item. Furthermore, mandatory security involves additional considerations not present in a pure role-based system: beyond the obvious restrictions on queries and updates, mandatory security has a profound impact on the architecture of MLS DBMSs, because indirect means of downward information flow, called covert channels, must also be prevented. For example, if a service of the database is denied to a subject based on the presence of a tuple at a higher level, the subject can infer its existence, resulting in downward information flow. With respect to security, more is at stake than inferring the presence of a particular tuple. The success or failure of the service request can be used repeatedly to communicate one bit of information to the lower level. Therefore, *any* information visible at the high level can be sent through the channel.

The problem of *polyinstantiation* arises through the avoidance of a covert channel. If a user inserts a tuple with key k, a user from a lower level cannot be prevented from inserting a different tuple with key k at a later time, as refusing the later insertion would open a covert channel. As a result, MLS relations can contain multiple tuples with the same key value, known as polyinstantiated tuples. This problem has been addressed in previous models by means of syntactic integrity properties, which control the extent and form of polyinstantiation (e.g., Jajodia and Sandhu [1990; 1991] and Sandhu and Jajodia [1992]).[4]

We will begin our survey of related work in mandatory security by address-

[3] Many applications using mandatory security allow subjects to perform writes to data at higher levels. In a database context, however, writing-up is usually prohibited, as (at least in the straightforward implementation) it would allow subjects to completely overwrite data at higher levels.

[4] In our work, we seek a formal foundation for MLS databases that prevents polyinstantiation from becoming an issue. From a practical perspective, one could try to achieve this goal by designing the schema so that individuals authorized to update a relation or attribute have labels that allow them to see all the values that could ever appear in that relation, so that polyinstantiation is not needed.

ing two high-level questions: *what does "MLS data semantics" mean*? and *why is it important*? We then survey other uses of modal logic in security research, and then discuss issues relating to data semantics in MLS databases.

2.2.1 *Data Modeling versus Semantics of Data.* When we say that we are working on the problem of semantics for MLS databases, people often think that we are working on the problem of data modeling for MLS applications. For this reason, we will explain the difference between semantics of data and data modeling, and show the utility of studying semantics of data.

We believe that the current situation in the MLS community is analogous to the state of research on null values in relational databases in the late 1970s. The early work in the database community on the problem of null values consisted largely of a stream of proposals and counterproposals for what the result of a query should be when the database contained nulls. These early proposals were operational in nature—they explained how a query against such a database should be evaluated. Confusion reigned for quite a few years, with each approach appearing to have its own semantic problems.

Then researchers realized that the "bugs" in proposals stemmed from a lack of understanding of what null values *meant*. People realized that nulls were not just additional values in an attribute domain: a new definition was needed for what a database containing nulls meant, before anyone could talk about how queries could be evaluated, or what updates might mean. Several solid proposals for the semantics of different kinds of nulls were quickly put forward, and the theoretical understanding of null values quickly solidified. Today null values are a "mature" research topic in the database world.

Quite separate from the question of what nulls meant was the issue of semantic data modeling for applications that may have nulls. Work in that area addressed the question of when and where applications should allow nulls to appear as values, and offered means of expressing limits on the appearance of nulls as annotations to an entity-relationship or relational schema. This work was clearly useful, and its completion did not hinge on the development of a semantics for nulls.

In our opinion, the current state of affairs in research into databases obeying mandatory security has overtones of the long-ago null values scenario. There has been a great deal of examination of the question of what should constitute a legal instance of an MLS relational schema, and many operational semantics for SQL operations over MLS databases have been proposed. No obvious winner of the MLS "data model wars" has appeared, and the models themselves have undergone steady revision over the years. We believe that the main source of difficulties, as with null values, has been a lack of examnation of the question of what an MLS database really means. A null is not an ordinary value in an attribute domain, and neither is a security label. From our point of view, the introduction of security labels radically changes the meaning of a database, and a suitable semantics is needed before other questions will find thoroughly satisfactory answers.

In parallel with the MLS data model wars, research has also been addressing the problem of semantic data modeling for MLS applications, through, for example, additions to ER modeling paradigms to allow expression of the need for a cover story for an attribute (e.g., Smith [1990] and Thuraisingham [1992]). This work has been moving along smoothly; semantic data modeling for MLS databases and semantics of MLS databases are two different problems that can be addressed in parallel, just as the questions of schema-level modeling of nulls and the meaning of nulls are separate.

2.2.2 *Modal Logic in Database Security Research.* We know of several other efforts to use modal logic to help with security issues in databases. Modal logic is just a tool, not a solution approach, and so these other authors have used modal logic to explore questions different from those that interest us here. We briefly survey these efforts below.

Glasgow et al. [1992] use a modal logic of knowledge as the basis for specifying and reasoning about secure distributed systems. Their paper considers multilevel security in general, and applies their techniques to a database example: the question of describing Seaview's policy of mandatory security and integrity. Their interest is in security policy description, rather than in the database concerns of query languages addressed in this paper.

Bonatti et al. [1992] use a modal logic of knowledge, belief, secrecy, and allowability to address an aspect of the inference problem [Akl and Denning 1987; Su and Ozsoyoglu 1987; Binns 1992a; 1992b; 1993]. Bonatti et al. do not consider multilevel security in the sense that we do here—their propositional facts are either secret or nonsecret—but their techniques could certainly be applied to MLS relational databases. Under Bonatti et al.'s approach, the DBMS knows which facts are to be kept secret from a user, and also knows the inference rules and pieces of knowledge that would allow a user to put two facts together and conclude a third. The job of the DBMS is to keep a transcript of the interactions between the DBMS and the user, and to answer user queries using the following strategy: if answering an incoming query truthfully would allow a user to conclude a secret fact correctly (based on previous information given out, plus the current answer), then the DBMS can either refuse to answer or, if a refusal would give away secret information, lie. Bonatti et al.'s approach is very interesting; in an MLS context, its dangerous assumptions are that the user knows no more than the DBMS does about possible inferences, and that the user does not share information with other users who have had separate dialogs with the DBMS (as answers to the same query might be different in different dialogs).

An earlier but very similar piece of work to Bonnati et al.'s was conducted by Sicherman et al. [1983]. They also consider the problem of keeping facts secret, but the only strategy that the system is allowed to use is that of keeping silent. The authors use a nonmodal formalization of the problem.

Cuppens [1992] has used a modal logic of allowability and knowledge to investigate the *aggregation* problem. This problem arises when individual data items in a grouping are classified at one level, but the grouping as a whole is classified at a higher level; the canonical example is an organiza-

tional phone book which is to be kept secret, but whose individual entries can be given out to those who request them. Cuppens' approach has much in common with the Bonatti and Glasgow papers described above, as he uses modal logic to give a nice formalization to the otherwise nasty problem of aggregation. As in the other papers described above, Cuppens' interests do not extend to query languages. Our work does not address either the inference problem or the aggregation problem.

2.2.3 *The Need for Semantics in Secure Databases.* We believe that any proposal for a secure relational model, under either discretionary or mandatory security, that presents different users with different versions of reality should include the following elements:

(1) A formal definition of the security properties that databases under this model will possess.
(2) A definition of the syntax of databases under the model.
(3) A semantics for databases under the model that can be used to reduce a database containing multiple versions of reality to a set of single-level databases, each database representing the beliefs about the state of the world held by the users at a particular security level.

In other words, given a secure database, one should be able to answer the question *what do the subjects having role/label l believe to be the current state of the world*? In any ordinary (single-level) relational database, this question has a trivial answer: the users believe that the state of the world is exactly that given by the tuples in the database. Our approach to secure databases rests on two assumptions: first that all the subjects at a particular level can agree on the current state of the world, restricted to the database schema, and second, that it is possible to set out, in a formal language, a policy specifying exactly how to construct that state of the world by combining beliefs which are explicitly stated by subjects at that level, plus selected pieces of information drawn from the world states believed in at lower levels.

In the remainder of this section, we will show how lack of such a semantics leads to problems in databases that store multiple versions of reality, including current MLS models. In our examples, we will use the recent model of Jajodia and Sandhu [1991]; the problems we describe are present in other MLS models as well. Under the model of Jajodia and Sandhu [1991], security labels are attached to individual fields of a tuple, and a subject is only able to read a particular tuple field if the subject's label dominates that of the tuple field.

Semantic Ambiguity. An MLS relation does not always have a single semantic interpretation. For example, Figure 1 shows an instance of the relation scheme SOD (Starship, Objective, Destination), where Starship is the key, as an MLS relation under the model of Jajodia and Sandhu [1991]. Consider the two polyinstantiated tuples sharing the attribute value 'Enterprise'. These tuples can be interpreted in at least two ways:

—They represent two different levels of secure understanding of a single starship named Enterprise: the first tuple gives the perspective of an

Starship	Starship Class	Objective	Objective Class	Destination	Destination Class	Tuple Class
Enterprise	U	Exploration	U	Vulcan	U	U
Enterprise	C	Diplomacy	C	Romulus	C	C

Fig. 1. One schema and instance for the secure SOD relation.

unclassified subject, and the second tuple gives the perspective of a confidential subject.

—Two entirely different starships exist, both (unfortunately) with the name Enterprise; one tuple refers to each ship.

One might like to know whether C subjects believe that there is one Enterprise or two, or are unsure.

Additionally, there is the question of what S subjects believe about the Enterprise—do they agree automatically with beliefs from the C level, the U level, or both? The question becomes more difficult if, for example, the two tuples in the example came from parallel levels C_1 and C_2 that are directly dominated by S. If C_1 and C_2 hold conflicting beliefs, then what is the policy that determines the beliefs of S subjects?

Query Ambiguity. In Figure 1, when a C subject asks:

```
SELECT  Destination
FROM    SOD
WHERE   Starship = 'Enterprise';
```

what should be the answer? One possibility is 'Romulus', because that is the value associated with that user's security level. Another answer could be 'Romulus' and 'Vulcan', because that is the exhaustive list of all values associated with destinations of Enterprises in the database. Current MLS proposals do not address the question of whose beliefs are to be consulted when answering a query; typically they include all visible beliefs when processing a query.

In the case of a more complicated query, the ambiguity becomes more acute: current proposals will join together two tuples with different security labels, even in the common case where it is clear that no subject at any level would believe that the joined tuple reflects the state of the world correctly. For example, let us consider the question of which starships are conducting exploration on Romulus. This query can be written as

$$\textit{Ships Exploring Romulus} = \pi_{Starship}\sigma_{Objective = \text{'Exploration'} \wedge Destination = \text{'Romulus'}}(SOD).$$

One might also express the query as

$$\textit{More Ships Exploring Romulus} = \pi_{Starship}\sigma_{Objective = \text{'Exploration'}}(SOD) \cap \pi_{Starship}\sigma_{Destination = \text{'Romulus'}}(SOD),$$

Starship	Starship Class	Objective	Objective Class	Destination	Destination Class	Tuple Class
Enterprise	l	Exploration	l	Vulcan	U	l
Enterprise	C	Diplomacy	C	Romulus	C	C
Enterprise	l	Exploration	l	Pluto	TS	TS
Enterprise	C	Diplomacy	C	Pluto	TS	TS

Fig. 2. Tuple proliferation upon update.

a phrasing which might come about through the use of 4GL code and view definitions. Under ordinary relational algebra semantics, the answer to *ShipsExploringRomulus* is the empty set, but the answer to the "equivalent" query *MoreShipsExploringRomulus* is 'Enterprise', even though *no one* believes that the Enterprise is conducting exploration on Romulus.

Should these dubiously joined or intersected tuples contribute to a query answer? In current proposals for a MLS relational model, the semantics given for the query language is limited to a description of the "filter function" used to screen out higher-level information that the user should not be allowed to see. There is no built-in mechanism for specifying whose beliefs a query Q should be evaluated over. Thus Q will generally be evaluated over a mish-mash of information from different levels, and the answer to Q might not be believable from any single subject's perspective. Even if a user wished to spell out conditions in Q, so that Q would only be evaluated over his or her beliefs about the state of the world, this would not be possible because there is no definition of exactly what those beliefs are, nor any easy way of specifying them.

Intuitively, if a user from level l asks a query Q using ordinary SQL, we would like to answer Q with respect to l's beliefs about the current state of the world. It should also be possible for an l-level user to ask questions about what lower levels believe to be the state of the world. In other words, an S-user should be able to ask *Where is the starship Enterprise going*? and be told that the Enterprise is heading to Pluto, if that is what S people believe. The user should also be able to ask *Where do Unclassified users think that the Enterprise is going*? and receive a (possibly different) answer. Our proposed query languages will make these queries easy to pose.

Proliferation of Tuples Due to Updates. In Denning et al. [1987] an update can introduce a number of new polyinstantiated tuples that is exponential in the number of security classes involved. Jajodia and Sandhu [1991] eliminate much but not all of this proliferation, as in the following update to Figure 1 by a *TS*-user:

```
UPDATE  SOD
SET     Destination = 'Pluto'
WHERE   Starship = 'Enterprise';
```

As Figure 2 shows, the number of Enterprise tuples doubles after this update is performed. If the two Enterprise tuples refer to different secure under-

standings of *one* ship, this update can be interpreted to mean that the *TS*-user is adding a still-more-secure understanding about the destination of the Enterprise. Doubling the number of tuples does not seem necessary to add this understanding.[5]

The common theme in the problems just described is the lack of semantics underlying the MLS relational model. Without a semantics, syntactic issues cannot be resolved through connection to the semantics. For example, polyinstantiation has posed such a thorny issue because it is not clear what polyinstantiation *means*.

Our earlier work on this problem appeared in Smith and Winslett [1992a], where Smith and Winslett introduced the idea of database interpretations (with a somewhat different definition than that used in this paper). In that paper we also gave a syntax for multilevel SQL, without cross-level joins. In this paper we have improved the syntax for multilevel SQL, changed the intended meaning of some operations, permitted cross-level joins, and given a formal semantics for the resulting queries. We have also addressed the question of other formal query languages, such as secure relational algebra, and given a semantics for them.

In other related work of ours [Qian 1994a; 1994b], Qian brings a belief-based perspective to bear on the problem of multilevel integrity constraints. In Qian and Lunt [1992], we also show how to retrofit our semantics to one particular class of MLS relational models.

3. A SEMANTICS FOR SECURE RELATIONAL DATABASES

In this paper we distinguish between the semantic and syntactic aspects of secure databases. This section describes a semantics for secure databases. In particular, we define an *interpretation* (in the logic sense) of a secure database, which draws on Kripke models [Chellas 1980] with a simple, nonlogic-based presentation. Our database interpretations are simple and easy to understand, and can be used to give meaning to syntactic features and to operations. Our interpretations can serve as a *lingua franca* underlying others' syntax, providing a means of information interchange between secure databases that use different syntax through a common understanding of the meaning of that syntax. In this section, we will present our approach using the terminology of multilevel secure databases, i.e., mandatory security; however, we intend this general approach to also be extensible to the role-based systems of the future.

[5] In the SeaView system [Denning et al. 1987], all these tuples are not stored directly in the database. Rather, at the physical schema level, each relation is decomposed into a set of fragments, so that each fragment contains only information at a single level (plus the key). Then at run-time, all the fragments whose labels are dominated by that of the current subject are joined together in a many-way join. The subject is presented with that entire relation, and it is up to the subject to determine which tuples in the relation are actually true of the world, which are believed only by others, and which are not believed by anyone and are merely part of the aftermath of joining together information originating at different levels.

Let us present first a formal underpinning for ordinary (single-level) relational databases. We will then extend that definition to the secure case.

3.1 Formalizing Ordinary Relational Databases

Assume we are given a finite set of named domains, $D_1, \ldots, D_d$, each consisting of a set of values. Then a *relational database schema* takes the form

$$\{R_1(A_{11}: D_{11}, \ldots, A_{1n_1}: D_{1n_1}), \ldots, R_m(A_{m1}: D_{m1}, \ldots, A_{mn_m}: D_{mn_m})\},$$

where each R_i is the name of a relation; A_{ij} is the name of a unique attribute of relation R_i; and D_{ij} is the domain of attribute A_{ij} (that is, one of $D_1, \ldots, D_d$). A *relational database instance* corresponding to this schema is an assignment of a finite subset of $D_{i1} \times \cdots \times D_{in_i}$ to each relation R_i of the schema. We will often use the term *relational database* to refer to the combination of a schema and an instance; the meaning will be clear from context.

A database schema may also include information about integrity constraints that the database must satisfy, such as key and referential integrity constraints. These constraints can be described using an extension of the query language, or by special-purpose constructs. We will not dictate a particular form for expressing integrity constraints, because they will not be a matter of concern in this paper.

Because we are interested in secure databases, we must add a final set of features to the database schema, to describe relevant facts about security. Intuitively, we want the schema to describe the security hierarchy of the application domain, and to say what the security label is of the information in the current database. We will do this by requiring that the schema contain several special relations.

First, we require the existence of a domain called *Labels*, whose values are all the different security class names in the security hierarchy. We assume that a partial order is defined on *Labels*, so that $l \geq l'$ holds iff l dominates l'. Then, we require the schema to contain two unary relations, *Anyone*(*label* : *Labels*) and *Self*(*label* : *Labels*). In the ordinary relational databases formalized in this section, all database users have the same security clearance l, and the instance for *Self* must be $\{\langle l \rangle\}$. In other words, *Self* gives the level of the subjects whose complete beliefs are contained in the database. We will say that the database *has label* l. Similarly, the instance for *Anyone* must contain all security classes l' such that $l \geq l'$.[6]

Note that our formal treatment does not allow null values, just as ordinary relational algebra omits consideration of nulls. Null values may be included in the formal treatment by formalizing them in one of the standard manners

[6]Although *Self* could be computed on demand from *Anyone* using *max*, we will include *Self* explicitly in the schema, as it will not ever be updated and is very useful in queries.

(e.g., Zaniolo [1984], Liu and Sunderraman [1990], and Imielinski and Lipski [1984].[7]

3.2 Formalizing MLS Relational Databases

Under our semantics, intuitively the interpretation of an MLS database is a set of ordinary relational databases, one database for each label in the security hierarchy.[8] The databases all share the same schema,[9] and each database is tagged with its label. Additionally there is a binary relationship between databases in the interpretation, which holds exactly when the label of the first database dominates the second, according to the security hierarchy. From the properties of the security hierarchy, it follows immediately that the binary relationship is reflexive, antisymmetric, and transitive.

Superficially, this approach may look like database-level labeling of information; however, that perspective is incorrect. The label on a database indicates the label of the *subjects* who believe that the contents of the database describe the state of the world accurately. The information in the database may have come from sources at many different levels, and thus may have many different security classifications. For example, an S subject may agree with some U beliefs, such as an unclassified list of zip codes. Although the zip codes have security classification U, both U and S subjects believe that the zip code information is correct, and both will include the zip codes in the database of beliefs at their level. For applications where it is important for users to see the actual classification of information, we would expect to see an additional column or columns in the relational schema, so that subjects having clearance l could see the classification l' of each piece of information. The issue of tracking classifications of information is an interesting topic in its own right, and it would be possible to include direct support for it in a semantics like ours, but as it is orthogonal to our concerns in this paper, we will not consider it further.

The database tagged with a particular label contains the total *beliefs* of the subjects having that label about the state of the world reflected in the

[7] We have omitted nulls because they cause a divergence between syntax and semantics of ordinary relational databases. In particular, under most formal models for nulls, a single (syntactic) database with nulls corresponds to a set of the database instances defined above—one instance for each possible choice of values for its nulls—rather than to a single instance. This complicates the definition of relational algebra, and means that a single (syntactic) MLS database with nulls would correspond to a set of ordinary relational instances, with one *or more* instances per level. By omitting nulls, an MLS database in our model will correspond to a set of ordinary relational instances, with exactly one instance per security class. Because nulls are omitted in the formal treatment, we hope to keep the formalities more palatable to the reader not already familiar with the complications of null values, without depriving other readers of any new twists on the treatment of nulls.

[8] There is the potential for confusion between the syntactic MLS database and its interpretation as a set of databases. We use the term "database" to refer to a database at a level of the interpretation.

[9] If different schemas are desirable at different levels, this can be accomplished using techniques similar to those used in the SeaView project [Denning et al. 1987]. The issue of different schemas at different levels is orthogonal to our proposal for semantics.

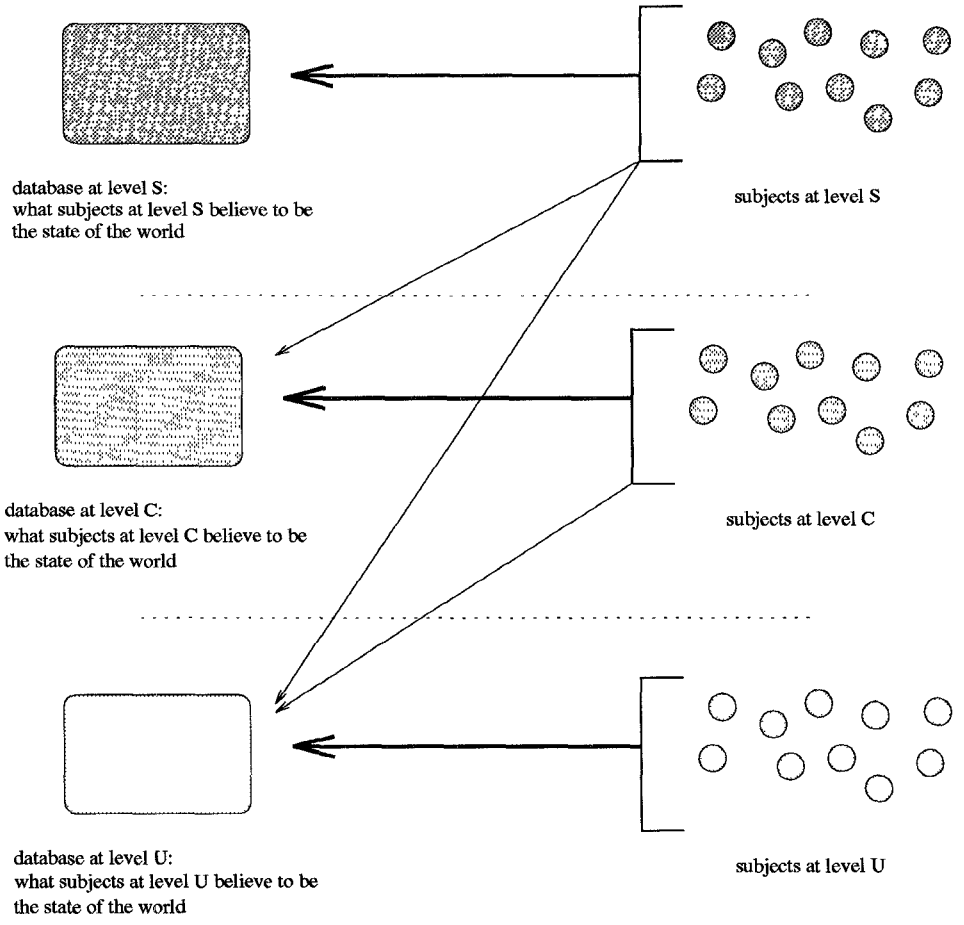

Fig. 3. Relationships between subjects and databases of different levels.

schema.[10] We use the term "belief" because subjects with different labels may make different statements about the value of the same attribute for the same entity.

The binary relationships between databases in an interpretation are motivated by Figure 3, which shows subjects and databases for three linearly ordered levels: S, C, and U.

A subject *believes* only the contents of the database at its own level, as represented by the thick arrow in the figure from a circle (a subject) to a box (the database) at the same level. The subjects of each level *see* what they and the subjects of each lower level believe, as represented by the thin arrows from subjects at one level to the databases at lower levels. A subject may see

[10] We use the term "belief" in an intuitive manner here, not in the technical sense of modal logic models of belief, which are not necessary in this simple application.

Database Interpretation at level S:

SOD

Starship	*Objective*	*Destination*
Enterprise	Diplomacy	Romulus

Self

Label
S

Anyone

Label
S
C
l

Database Interpretation at level C:

SOD

Starship	*Objective*	*Destination*
Enterprise	Diplomacy	Romulus

Self

Label
C

Anyone

Label
C
l

Database Interpretation at level l:

SOD

Starship	*Objective*	*Destination*
Enterprise	Exploration	Vulcan

Self

Label
l

Anyone

Label
l

Fig. 4. The interpretation of a starship's database.

many tuples that it does not itself believe. The information about which databases are visible from which levels is embedded in *Self* and *Anyone*.

Formally, an MLS database consists of a relational schema, as defined above, plus an interpretation:

Definition 3.2.1. An MLS interpretation I over schema S is defined by

$$I = \bigcup_{l \in Labels} I_l,$$

where each I_l is an ordinary relational database over S with label l. We will say that I_l is the database at level l or the interpretation at level l.

For example, Figure 4 shows an interpretation that might correspond to the syntax presented in Figure 1.

Database access privileges for subjects are summarized below:

(1) *Update Access:* a database update request (insert, delete, update) from a subject can only alter data in the interpretation at the subject's own level.

(2) *Read Access:* a query from a subject at level l can access data from exactly those databases whose label is dominated by l.

The read access rule corresponds to the Bell-LaPadula simple property: a subject can retrieve exactly what it can see that someone believes. The

update access rule states that a subject can change its own beliefs, but no one else's. The update access property is stronger than the Bell-LaPadula ★-property; the latter is semantically tailored to message-passing systems, where it would not violate security to allow a lower level to append a message to a higher level's message queue. We will consider queries and updates in more detail in Section 5, including the ability to specify in a query just whose beliefs the query is to be evaluated against.

It is important to note that the beliefs held by the subjects at level l, as described in the interpretation, are not cumulative. In other words, subjects at level l do not have to believe anything that is believed at the levels below them:[11] each level of the interpretation holds the complete beliefs of the subjects at that level. Often, current MLS relational model proposals seem to assume, without ever actually stating so, that tuples assigned lower-level labels are sometimes believed by subjects at higher levels, i.e., that beliefs are cumulative in some way. One of our goals in this work is to make all assumptions regarding cumulativeness explicit, in the form of integrity constraints. Without such a provision, it is not possible to state unambiguously what beliefs are held by the subjects at each level.

We anticipate that integrity constraints[12] will be used extensively to relate information believed by subjects at different levels. The integrity constraints will form an official policy for the database, stating in a declarative fashion how beliefs held at higher levels must relate to those held at lower levels. For example, a typical integrity constraint might say that the subjects at level l agree with all the beliefs (tuples) in relation R at the levels immediately below l', except when such an agreement would lead to a key conflict or referential integrity violation at level l [Qian 1994a]. Or, a constraint might state flatly that all levels must agree completely with the beliefs held at level U regarding, e.g., the ZipCode and LocalPlanets relations. One can use these cross-level constraints to restrict the amount and form of polyinstantiation, and the permissible uses of cover stories.

In an MLS context, the constraints can be enforced in a variety of ways, without violating mandatory security [Costich and McDermott 1992]. Database-integrated production rules, such as those implemented within Starburst [Widom and Finkelstein 1990] and Postgres [Stonebraker et al. 1988], have recently been integrated into the MLS database framework [Smith and Winslett 1992b]. We anticipate that triggered rules will be the

[11] An integrity constraint may require subjects at level l to hold some of the same beliefs as are held at a lower level l', as described later. In this case, part of the interpretation at level l' will be duplicated at level l, so each level of the interpretation still holds the complete beliefs of the subjects at that level.

[12] Our earlier discussion of integrity constraints considered only single-level integrity constraints, such as are found in ordinary relational databases. Here we are referring to cross-level integrity constraints, which can be defined using the secure query language constructs in this paper. For now, we ask the reader to assume the existence of a language with which one could express cross-level constraints.

enforcement mechanism of choice for many rules [Ceri and Widom 1990], while some particularly common types of rules may have special-purpose built-in constructs available at the syntactic level to declare and enforce them (much as single-level relational systems usually have special support for declaring and enforcing ordinary relational keys). We anticipate that secure integrity constraints and enforcement policies will be a lively area of future research. The big challenge in this area is to determine whether a set of constraints has an enforcement policy that will not violate security by denying service to a user based on information not visible to the user (such as the violation of a constraint at a higher level). Much attention has already been paid by others to the problem of enforcing MLS key constraints in proposals for an MLS relational model. For approaches to handling more complex constraints, we refer the reader to our related paper [Qian 1994b]. In the current paper, we simply assume that any integrity constraints for the database have been compiled into an enforcement policy that will never violate mandatory security.

Our approach to integrity constraints differs sharply from that of most proposals for an MLS relational model, which build a variety of integrity constraints directly into their model (e.g., "no entity polyinstantiation," "at most one cover story," etc.). We do not believe that any one set of built-in integrity constraints will be necessary and sufficient for all applications; each of the MLS proposals to date has chosen a different set of built-in constraints. By not including a required set of integrity constraints in our definition of an interpretation, we make it possible for other authors to adopt our semantics to give meaning to their MLS syntax: the authors need only express their built-in constraints as integrity constraints, and supply a function to map each instance of a database under their model to an interpretation.

The assumption of independence of beliefs held at different levels (limited only by cross-level integrity constraints) does not have any implications for the manner in which the database is represented syntactically. For example, the chosen syntax may use attribute-level labeling. As mentioned earlier, any current proposal P for an MLS relational model can be given a semantics, by supplying a mapping from each database under proposal P to one of our database interpretations. For example, Qian [1994a] uses this approach to give semantics to a proposal that uses tuple-level labeling.

Just as our semantics does not restrict the choice of database syntax, neither does it restrict the choice of database implementation. We anticipate that most of the information believed at a particular level will also be believed at the levels above it, and therefore a storage architecture that avoids redundant storage of information will prove most appropriate from that point of view (although other concerns may mitigate in favor of redundant storage; see Froscher and Meadows [1989]). The repetition of shared beliefs that is inherent in our semantics need not carry over into a redundant storage representation. We believe that the semantic-level redundancy (i.e., the ability to query a complete and consistent world view, shared by all subjects with a particular label) is needed to give the user an easy way of understanding the content of the database.

4. FORMAL QUERY LANGUAGE

In this section we present secure relational algebra, a formal query language for use with the database interpretations defined in the previous section. Syntactically, secure relational algebra is ordinary relational algebra, plus an additional symbol **B**, which can be thought of as meaning "believes." Then we show how our technique of extension can be applied to any other formal query language, including logic-based query languages.

In the remainder of the paper, we will use a, b, c to denote constants; A, B, C to denote attribute names; l to denote a constant that is a label; E to denote a secure relational algebra expression (defined below); Q to denote a secure query; ϕ to denote a selection condition; and R to denote a relation name. Any of the above may be subscripted.

4.1 Secure Relational Algebra: Syntax and Semantics

We will use an inductive definition to define a *secure relational algebra expression* (*query expression* for short). First, R is a query expression, and so is the constant relation containing the m tuples $\{\langle c_{11}, \ldots, c_{1n}\rangle, \ldots, \langle c_{m1}, \ldots, c_{mn}\rangle\}$. If E_1 and E_2 are query expressions, then so are the following:

(1) *Cartesian Product.* $(E_1 \times E_2)$;

(2) *Union.* $(E_1 \cup E_2)$, where E_1 and E_2 both have arity n and have the same underlying domain for their ith attribute, $1 \leq i \leq n$;

(3) *Difference.* $(E_1 - E_2)$, where E_1 and E_2 both have arity n and have the same underlying domain for their ith attribute, $1 \leq i \leq n$;

(4) *Projection.* $(E_1[A_1, \ldots, A_n])$, where each A_i is an unambiguous[13] reference to an attribute;

(5) *Selection.* $(E_1[\phi])$, where ϕ is a selection condition (defined below);

(6) *Level Shift.* $(\mathbf{B}[E_1]E_2)$, where E_1 is a query expression whose result is a unary relation over *Labels*.

Items (1) through (5) are the usual definition of relational algebra (see, e.g., Ullman [1988] and Korth and Silberschatz [1991]), and we will not say much about them here. Item (6) is new; intuitively, $\mathbf{B}[E_1]E_2$ poses the question contained in E_2 to the database at the level(s) determined by E_1.

Using an inductive definition, the selection condition ϕ occurring in $E[\phi]$ can be, as usual, any of the following:

(1) $(t_1\ op\ t_2)$, where *op* is any one of $=$, $<$, $>$, $\leq$, $\geq$, $\neq$, and each of t_1 and t_2 is a constant or an unambiguous reference to an attribute of E;[14]

[13] A_i is an unambiguous reference to an attribute if A_i is an integer between 1 and the arity of E_1. As syntactic sugar, in our examples we will instead use references of the form A, where A is the name of an attribute, or $R.A$, where R is a relation name and A is the name of an attribute, wherever possible.

[14] We have assumed here that all comparators are defined across all domains. If this is not the case, one must restrict the syntax by requiring that t_1 and t_2 come from comparable partially ordered domains.

(2) $(\phi_1 \wedge \phi_2)$;
(3) $(\phi_1 \vee \phi_2)$;
(4) $\neg \phi$.

From time to time, we will omit some of the required parentheses in query expressions, using the usual precedence conventions.

We next define the interpretations (intuitively, the meaning) of query expressions, by reducing them to ordinary relational algebra manipulations. The interpretation of a query expression E at level l (written $|E|_l$) is a relation defined as follows:

—*Base Cases.* $|R|_l$ is the instance of R at level l. Similarly, $|\{\langle c_{11}, \ldots, c_{1n}\rangle, \ldots, \langle c_{m1}, \ldots, c_{mn}\rangle\}|_l$ is $\{\langle c_{11}, \ldots, c_{1n}\rangle, \ldots, \langle c_{m1}, \ldots, c_{mn}\rangle\}$.
—*Cartesian Product.* $|(E_1 \times E_2)|_l = |E_1|_l \times |E_2|_l$.
—*Union.* $|(E_1 \cup E_2)|_l = |E_1|_l \cup |E_2|_l$.
—*Difference.* $|(E_1 - E_2)|_l = |E_1|_l - |E_2|_l$.
—*Projection.* $|(E_1[A_1, \ldots, A_n])|_l = |E_1|_l[A_1, \ldots, A_n]$.
—*Selection.* $|(E_1[\phi])|_l = |E_1|_l[\phi]$.
—*Level Shift.*

$$|(\mathbf{B}[E_1]E_2)|_l = \bigcup_{\langle l'\rangle \in |E_1 \cap Anyone|_l} |E_2|_{l'}.$$

When $|E_1 \cap Anyone|_l$ is the empty set,[15] i.e., there are no security labels that satisfy E_1 and are dominated by l, then the interpretation is the empty relation, of the same arity as E_2.[16]

How are these definitions different from the usual relational algebra? First, we look at the interpretation of a query expression *at a specific database interpretation level*; in ordinary relational algebra there is only one level to consider. Second, our new level shift operator[17] asks for the query to be interpreted at each of a set of levels. The syntactic restrictions given earlier for projection, selection, union, and set difference guarantee in the usual way that the interpretation of an expression is always well defined. Our treatment

[15] The intersection operator is defined as $R \cap S = R - (R - S)$, as usual

[16] An alternative, which we do not pursue here, is to let l' range over all of $|E_1|_l$ and say that the interpretation is undefined if

$$|E_1|_l \not\subseteq |Anyone|_l.$$

Under this approach, a run-time error will be generated if a user tries an illegal level shift. Note that a level shift expression that only makes use of *Anyone* and *Self* (i.e., is posed in a manner relative to the level of the issuer) could never cause a run-time error.

[17] We call our secure query laguages *modal*, due to the presence of the level shift operator, which is called a *modality*. Semantics for modal languages are usually defined in terms of Kripke models [Chellas 1980], that is, models consisting of interrelated possible worlds. In our context, a Kripke model is a set of databases related to one another by the "dominates" relationship. Modal operators are usually defined in terms of relationships between possible worlds, just as the meaning of **B** depends on the $\leq$ relationship between levels.

of level shifts masks out any levels that are not visible from l, so that the user cannot shift to a higher level from a lower one and violate security.

If l is a security level, then $\langle l \rangle$ is a tuple containing l, and $\{\langle l \rangle\}$ is a relation containing the single tuple $\langle l \rangle$. A *query* (*from level* l) is a formula of the form $\mathbf{B}[\{\langle l \rangle\}]E$, where E is a query expression. (As syntactic sugar, we will write $\mathbf{B}[l]E$ as shorthand for $\mathbf{B}[\{\langle l \rangle\}]E$ when there is no possibility of confusion.) Intuitively, when a subject at level l poses a question E, we change the question to $\mathbf{B}[l]E$ (i.e., ask the question relative to l's beliefs) and evaluate that as a query. By having $\mathbf{B}[l]$ prepended automatically, the same expression can be typed in by a subject at any level, to find out the query answer from the perspective of that level: no tailoring is needed to adapt the expression to the current subject's level. The DBMS uses its knowledge to prepend the correct level, rather than relying on an untrustworthy source for that information.

We can now define the interpretation of a query $Q = \mathbf{B}[l]E$. The interpretation of Q, written $|Q|$, is $|E|_l$. If E does not contain any level shifts, then Q can be thought of as an ordinary relational query, applied to the database instance at level l.

4.2 Secure Relational Algebra: Examples

We now give a series of examples to show the kinds of queries that can be asked using secure relational algebra. Our example schema is *SOD*(*starship*, *objective*, *destination*). These questions can be posed by a subject at any level, using identical syntax; the DBMS will prepend $\mathbf{B}[l]$, where l is the level of the subject, before answering the query, so that the query will automatically be answered by examining the beliefs of the issuer.

(1) Find out (what I believe to be) the destination of the Enterprise.

$$(SOD[\,starship = \text{'}Enterprise\text{'}])[\,destination\,]$$

The syntax used here is that of an ordinary relational algebra query.

(2) List all starships that anyone believes to exist.

$$\mathbf{B}[\,Anyone\,](SOD[\,starship\,])$$

At each visible level, we determine the list of starships believed in at that level, and take the union over all levels.

(3) Find all levels where someone believes that the Enterprise exists.

$$\mathbf{B}[\,Anyone\,](((SOD[\,starship = \text{'}Enterprise\text{'}]) \times Self)[\,label\,])$$

The approach taken here is to check for the existence of the Enterprise at each level, retaining only the name of the level when the Enterprise is found. Only levels visible to the issuer will be checked.

(4) Find all starships that I do not believe in, but a lower level does.

$$(\mathbf{B}[\,Anyone - Self\,](SOD[\,starship\,])) - (\mathbf{B}[\,Self\,](SOD[\,starship\,]))$$

This example uses a query expression, *Anyone* − *Self*, to pick out the set of lower levels and find what starships are believed to exist there.[18] A set difference is used to remove any starships that are also believed in at the level of the issuer.

(5) Find all starships that every level believes to exist. Queries with the word *every* are clumsy to pose in relational algebra, and this clumsiness carries over to secure relational algebra; this query would be much easier to pose in an SQL-style language than it is in relational algebra. As a first step toward the solution, we will declare a view or a temporary relation (using new syntax) that gives an upper bound on the answer: the set of all starships that the issuer believes to exist.

$$MyStarships(starship) = (SOD[starship])$$

Next, we will determine which of these ships some other level does not believe in.

$$NotBelievedIn(starship) = (((MyStarships \times Anyone) - (\mathbf{B}[Anyone]((SOD[starship]) \times Self)))[starship])$$

From *NotBelievedIn*, we can create a list of the starships all levels believe to exist.

$$EveryoneAgrees(starship) = (MyStarships - NotBelievedIn)$$

(6) Find all starships that level C thinks that every level believes to exist.

$$\mathbf{B}[C]EveryoneAgrees$$

This query forms a relation containing just the level C, and uses that relation to pick out the levels where the query will be evaluated. If asked by a user at a level that does not dominate C, such as level U, the interpretation of the query is the empty relation.

4.3 Secure Relational Algebra: Properties

It will be easy to show that queries obey mandatory security (i.e., do not read up). First, let us formalize the definition of a secure query language.

Consider an ordinary relational algebra query Q, with arity n, over a given database schema S. Q can be thought of as a *function* from the set of all instances of S to the set of all n-ary relation instances. In other words, given a database instance I, $Q(I)$ is an n-ary relation instance, the answer to the query.

In the case of an MLS database, a query Q is a function that maps each database interpretation I (containing one ordinary database instance I_l per

[18] The expression *Anyone* in place of *Anyone-Self* would give the same answer in this example. Similarly, **B**[*Self*](*SOD*[*starship*]) could be replaced by *SOD*[*starship*] without changing the answer to the query.

label l) to an n-ary relation $Q(I)$. We will write $I|_{\leq l}$ for the set of ordinary database instances $I_j \in I$ such that $j \leq l$. To be secure, Q's mapping must not make use of information that is above the level of the issuer of Q.

Definition 4.3.1. Let S be a schema, l a security level, I and J database interpretations over S, and Q a query over S posed by subjects at level l. A query language *obeys mandatory security* iff for all choices of S, l, Q, I, and J,

$$Q(I) = Q(J) \quad \text{whenever} \quad I|_{\leq l} = J|_{\leq l}.$$

PROPOSITION 4.3.2. *The secure relational algebra query language obeys mandatory security.*

Note that an *implementation* of a query language that satisfies Definition 4.3.1 might violate mandatory security, because the implementation might allow covert channels or have other security-related bugs.

PROOF. Let E be a query expression over schema S to be evaluated at level l. Suppose that E does not contain any operations (project, select, difference, union, level shift, product). Then $|E|_l$ is defined solely in terms of I_l, that is, solely in terms of the interpretation at level l. Therefore $|E|_l$ is the same no matter what data are contained at higher levels.

Suppose now that E contains $n > 0$ operations. $|E|_l$ has an outermost operation, either union, difference, product, selection, projection, or level shift. Consider the case where the outermost operation is not level shift. The argument(s) of the outermost operation each have $n - 1$ or fewer operations, so by the inductive hypothesis, we can assume that their interpretations at level l are defined solely in terms of the interpretations at level l and lower. Thus $|E|_l$ is also defined solely in terms of those interpretations.

Suppose now that the outermost operation of E is a level shift: $\mathbf{B}[E_1]E_2$. Then by definition, $|\mathbf{B}[E_1]E_2|_l = \bigcup_{l'} |E_2|_{l'}$, where $l' \in |E_1 \cap Anyone|_l$. Because *Anyone* contains only the labels $\leq l$, l' can only be the label of a database interpretation at level l or lower. Further, by the inductive hypothesis, we can assume that $|E_1|_l$ (i.e., the values that l' can assume) is the same no matter what information is present at interpretations above level l. By the same hypothesis, we can assume that $|E_2|_{l'}$ is the same no matter what information is present at interpretations above level l'. Since $l' \leq l$, we have that $|E|_l$ is the same no matter what the interpretations are of levels higher than l.

We conclude that the function represented by the query $Q = \mathbf{B}[l]E$ is independent of the levels of the interpretation above l, and therefore that secure relational algebra obeys mandatory security. □

The proof of this proposition brings out an important point regarding the composability of secure relational algebra. Suppose a query Q is to be evaluated at level l, and Q contains a subexpression E that is to be evaluated at a level $l' < l$. In other words, Q contains a level shift operation that shifts the focus of the query to the database at a level other than that of the issuer. In this case, the proof of the proposition shows us that $|E|_{l'}$ is

independent of information above level l'. In other words, the innermost clauses of E cannot refer to information at a level higher than l', even though the query issuer is allowed to see level $l > l'$. This property allows queries to be evaluated in a manner that is not context sensitive, i.e., it is not necessary to remember the level of the issuer, or levels previously visited, when evaluating inner portions of a query.

Updates are usually omitted from discussions of formal query languages. However, it is easy to see how to adapt a formal update language to the MLS world: since, as we mentioned earlier, MLS subjects can only write to the database at their own level, an update request received from a user can be applied to the database at his or her level, using ordinary relational update semantics. The presence of multiple levels does not affect the interpretation of the update, except in choosing which tuples are to be updated. In the next section we will show how this approach can be applied to SQL.

In a secure environment, it can be very useful to know what information in a query answer can be passed on to a lower-level user. For example, suppose an S subject asks a query Q, and then wants to communicate the results to a U subject. How can the S user know what part of the answer to Q can be shared with the U subject, without violating security? Under our approach, such a decision is easy. The S subject should first ask the query $\mathbf{B}[U]Q$, which means, "What would a U subject get as the answer to my original query Q?" and then communicate the answer to the U subject. From the point of view of the query language, the S subject need not worry about leaking secret information by mistake when this approach is used. Under other proposed approaches to MLS queries, it is not clear how an S user could safely communicate information in a query answer to a U user, except by logging in again at the U level and reposing the query. Of course, even with our formal query language, the system as a whole is only as secure as its implementation, and achieving a leak-free implementation of any MLS system is at best a daunting task.

Query languages are usually thought of as something to be applied to the syntactic form of a database or knowledge base, producing a syntactic result. Meaning is assigned to a query language by defining the result of a query in terms of the interpretation of the database or knowledge base, as well as in terms of the database or knowledge base syntax (and showing that the two definitions are equivalent). In our case, this would mean defining a query language for use with one or more of the proposals for MLS database syntax, and then showing what the answers should be for queries in that language, by defining their effect on the interpretation of the database. We have not followed that approach in this section, because there are widely divergent opinions regarding the appropriate syntax for an MLS database. Different choices of syntax may, in our opinion, require different choices of query language syntax. Thus we have bypassed the question of database syntax and gone directly to the problem of interpreting queries, in order to present a language with some universal utility. A query language designed for a particular MLS synax can be given meaning by showing how it maps to one of our formal languages.

4.4 Secure Relational Algebra: Identities

It is natural to ask what identities hold for the **B** operator in secure relational algebra, for use in generating and optimizing query plans. Given query expressions E_1 and E_2, we write $E_1 = E_2$ if for every database interpretation and level l, $|E_1|_l = |E_2|_l$. Proposition 4.4.1 lists the most important identities that hold.

PROPOSITION 4.4.1. *Let R be a constant relation (i.e., an explicit list of tuples), and let A be an attribute of expression E_1. Then the following identities hold:*

$$\mathbf{B}[E_1](E_2[A_1, \ldots, A_n]) = (\mathbf{B}[E_1]E_2)[A_1, \ldots, A_n] \tag{1}$$

$$\mathbf{B}[E_1](E_2[\phi]) = (\mathbf{B}[E_1]E_2)[\phi] \tag{2}$$

$$\mathbf{B}[E_1](E_2 \cup E_3) = \mathbf{B}[E_1]E_2 \cup \mathbf{B}[E_1]E_3 \tag{3}$$

$$\mathbf{B}[E_1](E_2 \times E_3) \subseteq \mathbf{B}[E_1]E_2 \times \mathbf{B}[E_1]E_3 \tag{4}$$

$$\mathbf{B}[E_1](E_2 - E_3) \supseteq \mathbf{B}[E_1]E_2 - \mathbf{B}[E_1]E_3 \tag{5}$$

$$\mathbf{B}[E_1 \cup E_2]E_3 = \mathbf{B}[E_1]E_3 \cup \mathbf{B}[E_2]E_3 \tag{6}$$

$$\mathbf{B}[Self]E_1 = E_1 \tag{7}$$

$$\mathbf{B}[E_1]R = \begin{cases} R & \text{if } E_1 \cap Anyone \text{ is nonempty} \\ \{\ \} & \text{otherwise} \end{cases}$$

$$= R \ltimes (R \times (E_1 \cap Anyone)) \tag{8}$$

$$\mathbf{B}[E_1](\mathbf{B}[R]E_3) = \mathbf{B}[R]E_3 \tag{9}$$

$$\mathbf{B}[E_1](\mathbf{B}[Anyone]E_2) = \mathbf{B}[Anyone \ltimes_{label \le E_1\ A} E_1]E_2 \tag{10}$$

The proof of Proposition 4.4.1 appears in the Appendix.

Note that Proposition 4.4.1 does not include identities involving projection, selection, difference, or cartesian product as top-level operations in the first argument to the **B** operator. Since the first argument to **B** must always evaluate to a unary relation, cartesian product should not appear as a top-level operation in the first argument of **B**. Due to the unary requirement, projection cannot be moved outside the first argument either. We do not know of any means of moving the evaluation of a top-level selection operation or difference outside the first argument of **B**.

Also note that Proposition 4.4.1 includes few identities involving nested **B** operators. For example, one cannot in general simplify the expression $\mathbf{B}[E_1](\mathbf{B}[E_2]E_3)$ to an expression involving only one **B** operator, because the result of evaluating E_2 at the level of the issuer of the query may be very different from the result of evaluating E_2 at the levels specified in E_1.

4.5 Other Secure Query Languages

At the beginning of this section, we promised to show how to use our approach to extend any formal query language to work in a secure context. Suppose that we have an ordinary (single-level) query language with a

well-defined semantics, such as relational algebra or calculus, or even a formalized subset of SQL. We will assume that the semantics takes the form of a mapping that takes a query expression and a database instance and produces a relation. The secure version of this query language will be defined inductively, as usual; but it will contain one additional operator not present in the single-level version, the level shift operation. Let E_1 and E_2 be two expressions over the query language, such that E_1 will evaluate to a unary relation. Then $\mathbf{B}[E_1]E_2$ is an expression in the secure version of the query language. To assign meaning to the extended query language, the interpretation of a query expression at level l is defined as in the original query language, except that (1) for operations other than the level shift, the interpretation must be computed with respect to the database at level l; and (2) the interpretation of a level shift operation at level l is

$$|(\mathbf{B}[E_1]E_2)|_l = \bigcup_{\langle l' \rangle \in |E_1 \cap Anyone|_l} |E_2|_{l'},$$

which is the same interpretation of level shifts as we used for MLS relational algebra.

Our modal query language is a bit of an odd duck in the modal logic world, because we allow, in some sense, queries over modalities. Axiomatization of our language might be an entertaining exercise for a theoretically minded colleague. We do not intend to investigate the question of axiomatization, as a model theory for the language is all we need for our purposes.

5. SECURE SQL

In this section, we present an extension of SQL to allow easy expression of secure queries. From a theoretical perspective, such an exploration may seem unnecessary, for two reasons. First, the previous section already showed how to extend a query language by adding a **B** operator, so what more is there to say? Second, under all MLS database syntax proposals, one can just use ordinary SQL on the syntactic form of an MLS database. Now that we have given some meaning to these databases, we could still allow the user to use ordinary SQL, and we could reject those queries that could not be given an unambiguous interpretation. Then the user would not need to learn any new SQL constructs.

While these arguments are valid to some extent, an explicit extension to SQL is needed for pragmatic considerations. It is too hard for a user to write ordinary SQL queries that have an unambiguous interpretation. Also, the **B** operation is not simple enough for users to grasp easily. Thus we present what we hope to be a more palatable SQL syntax, called Secure SQL.

For simplicity, we will assume that our dialect of SQL allows constant relations to appear in queries, so that connectives like IN and > ANY, as well as EXISTS, can appear with a constant relation as an argument, as well as a subquery.

In the following notation, the rectangular brackets denote optional items, and * denotes zero or more repetitions. We will not consider all of the

constructs of SQL, but we think that the means of extension to all of SQL will be clear to the reader.

5.1 Select

The SELECT statement has the following general form:

```
SELECT          A_i [, A_j]*
FROM            R_k [ , R_m]*
[BELIEVED BY    S]
WHERE           P;
```

The SELECT, FROM, and WHERE clauses are ordinary SQL syntax with the ordinary SQL syntactic restrictions. Nested SELECTs are allowed, as in ordinary SQL, and may have BELIEVED BY clauses of their own. In the BELIEVED BY clause, S is a nested query that must evaluate to a set of security levels. If the level of the subject issuing the query does not dominate some security level in the evaluated S, then that level is eliminated from the result of evaluating S. If the BELIEVED BY clause is absent, S is set to the level of the issuer of the query, so that the query accesses only those tuples that the subject believes.

In order to provide a bit of syntactic sugar to keep BELIEVED BY clauses short for the most common kinds of queries, we will also allow S to contain a nonempty list of security levels and relation names, with the obvious meaning. Examples appear after the discussion of Secure SQL semantics.

Intuitively, the result of the SELECT operation can be determined from the database interpretation as follows. First, evaluate the (outermost) BELIEVED BY clause at the database interpretation at the level of the issuer of the query. Second, evaluate the WHERE clause separately at each level mentioned in the result of evaluating the BELIEVED BY clause. The results are unioned and presented to the issuer of the query.

More formally, let us suppose that we have a mapping that gives the interpretation of ordinary SQL queries—i.e., given an ordinary database and SQL query, this mapping determines the relation that is the answer to the query. We will write this mapping as $\|Q\|$, when Q is an ordinary SQL query. The awkward point in specifying $\|Q\|$ is in handling nested subqueries that share tuple variables with outer queries ("correlated queries"); correlated queries are orthogonal to our interest in security, and for that reason we will give a formal definition of interpretations only for noncorrelated[19] queries.

To make our task easier, if noncorrelated SQL query Q contains a nested query Q', we will assume that

$$\|Q\| = \|Q.(\|Q'\|)\|,$$

where $Q.(\|Q'\|)$ is the query obtained by textually substituting the relation $\|Q'\|$ for the subquery Q' in Q. We will say that Q' is *immediately nested* inside of Q if no other query is nested between Q and Q'.

[19] The correlated case can be handled by defining the interpretation of a parameterized query, and extending ordinary SQL to allow parameterized constant relations to appear in queries.

SOD Interpretation at level S:

Starship	*Objective*	*Destination*
Enterprise	Diplomacy	Romulus
Nighthawk	Warfare	Venus

SOD Interpretation at level $C1$

Starship	*Objective*	*Destination*
Enterprise	Diplomacy	Romulus

SOD Interpretation at level $C2$

Starship	*Objective*	*Destination*
Nighthawk	Warfare	Venus
Blackjack	Mining	Pluto

SOD Interpretation at level U:

Starship	*Objective*	*Destination*
Enterprise	Exploration	Vulcan

Fig 5 An interpretation of a SOD relation.

Our task now is to extend the mapping $\|Q\|$ to include our new BELIEVED BY construct, which corresponds to the **B** operator discussed in the previous section. We will write the extended mapping as $|Q|$. Let us assume that all syntactic sugar has been removed before interpretation. If a noncorrelated Secure SQL query Q is issued by a subject at level l, then $|Q| = |Q|_l$, the interpretation of Q at level l. In turn, $|Q|_l$ is defined as follows:

(1) If Q contains the immediately nested clause BELIEVED BY Q_1, then

$$|Q| = \bigcup_{\langle l' \rangle \in (|Q_1|_l \cap |Any|_l)} |Q_2|_{l'},$$

where *Any* is short for 'SELECT * FROM Anyone', and Q_2 is formed from Q by removing the BELIEVED BY clause. When $|Q_1|_l \cap |Any|_l$ is the empty set, i.e., there are no security labels that satisfy Q_1 and are dominated by l, then $|Q|$ is the empty relation, of the same arity as Q_2.

(2) If Q does not contain an immediately nested BELIEVED BY clause, but does contain the immediately nested subquery Q', then $|Q|_l = |Q.(|Q'|_l)|_l$.

(3) If Q contains no immediately nested BELIEVED BY clauses or subqueries, then $|Q|_l$ is $\|Q\|$ evaluated over the database interpretation at level l.

For example, consider Figure 5, assuming a hierarchy of four labels with U at the bottom, $C1$ and $C2$ dominating U and incomparable to one another,

and S dominating all. To list the beliefs of all visible levels about the destination of starship Enterprise, we use

```
SELECT        Destination, Label
FROM          SOD, Self
BELIEVED BY   Anyone
WHERE         Starship = 'Enterprise';
```

which returns $\{\langle Vulcan, U\rangle, \langle Romulus, C1\rangle, \langle Romulus, S\rangle\}$ for a user at level S, $\{\langle Vulcan, U\rangle\}$ for U and $C2$ subjects, and $\{\langle Vulcan, U\rangle, \langle Romulus, C1\rangle\}$ for $C1$ subjects.

Cross-level joins can be expressed by using a subquery with a BELIEVED BY clause. Cross-level joins can be useful as metaqueries, i.e., queries that compare what is believed at different levels. To make expression of these queries a bit simpler, we define a view that returns all the level labels that are strictly below the current level:

```
CREATE VIEW  AnyoneBelowMe(Label) AS
             SELECT  Label
             FROM    Anyone
             WHERE   Label NOT IN
                     (SELECT   *
                     FROM      Self);
```

For example, suppose a $C2$ subject wishes to find out which starship entities are believed in at the $C2$ level but not at any lower levels. The appropriate query is

```
SELECT   Starship
FROM     SOD
WHERE    Starship NOT IN
         (SELECT          Starship
         FROM             SOD
         BELIEVED BY      AnyoneBelowMe);
```

which evaluates to $\{\langle Nighthawk\rangle, \langle Blackjack\rangle\}$ for the database interpretation given in Figure 5. The same query would still be well defined if posed by users from other levels, and would return the empty relation.

An S subject can also use a BELIEVED BY clause to gather information that can safely be shared with a subject at a lower level. For example, the S user can find out what starships are known at the U level, by asking, 'SELECT Starship FROM SOD BELIEVED BY U'. Under other MLS proposals, especially those that rely on attribute-level labeling, it can be extremely awkward to pose a complex query at level S whose answer is guaranteed to be safe to share with a U subject, and it can be dangerous to rely on "advisory" labels attached to query answers.

Suppose now that the S subject wishes to find out what starships are believed in at lower levels but not at the S level. A first guess at the appropriate query is

```
SELECT        Starship
FROM          SOD
BELIEVED BY   Anyone
```

```
WHERE       Starship NOT IN
            (SELECT         Starship
            FROM            SOD
            BELIEVED BY     Self);
```

but this query will not return the desired answer, because Self in the inner query will be evaluated separately over each level visible to the S user, rather than always evaluating to S as the user intended. In other words, the scoping rules will prevent an inner query from referring to a level not visible at the level of the outer query. This problem is an artifact of SQL, not of BELIEVED BY; SQL forces certain types of queries to be expressed with a certain form of nesting. One possible solution is to form a temporary relation at the level of the query issuer, and break the query into two parts. Our preferred way out of the difficulty, however, is to introduce INTERSECT and MINUS operations that connect two queries, just like the UNION keyword in ordinary SQL. Then the example above can be expressed as

```
(SELECT         Starship
FROM            Starships
BELIEVED BY     Anyone)
MINUS
(SELECT         Starship
FROM            Starships
BELIEVED BY     Self);
```

5.2 Insert

The secure SQL INSERT statement has the following general form.

```
INSERT
INTO    R[(A_i[ , A_j]*)]
VALUES(a_i[ , a_j]*);
```

In the inserted tuple, attribute A_i is to be given the value a_i. One can also use an INSERT command with the subquery syntax allowed in ordinary SQL. Under this approach, Q below is an ordinary Secure SQL query.

```
INSERT
  INTO R[(A_i[ , A_j]*)]
  Q;
```

In either case, a set is formed of tuples to be inserted into the database interpretation at the level of the requestor. If the VALUES clause is used, the user has specified the tuples directly; if the subquery form is used, then the set of tuples to be inserted is given by the interpretation of the subquery, defined in the previous section. Although subqueries may of course contain BELIEVED BY clauses, the INSERT command does not include such a clause of its own, because a subject can only insert tuples into the interpretation at the subject's own level. Key attributes must be assigned nonnull values, else the request is rejected, as is usual in SQL. Any omitted nonkey attributes of R are assigned a null.

For example, an S subject can assert agreemet with $C2$-level beliefs about the Blackjack, using the following insertion.

```
INSERT
INTO    SOD
        SELECT        *
        FROM          SOD
        BELIEVED BY   C2
        WHERE         Starship = 'Blackjack';
```

A user's INSERT request (or UPDATE, or DELETE) might possibly lead to an integrity constraint violation, thus triggering actions of the enforcement policy associated with the integrity constraint. For example, the INSERT request might cause the triggering and execution of rules at the user's level, or at higher levels. In this paper, we assume that the execution of such rules has been shown to guarantee mandatory security (e.g., through preanalysis and run-time multilevel transactions [Costich and McDermott 1992]), and focus our attention on the update requests themselves.

5.3 Update

The UPDATE statement has the following general form.

```
UPDATE  R
SET     (A_i = a_i[ , A_j = a_j]*)
WHERE   P;
```

As is usual in SQL, the SET clause cannot contain key attributes. As with INSERT, an UPDATE clause has no BELIEVED BY clause of its own, although it may contain a subquery with a BELIEVED BY clause.

To interpret an UPDATE request, begin by replacing the UPDATE and SET clauses by SELECT * FROM R, creating a query Q. The tuples in $|Q|_l$ are the tuples of the interpretation at level l that will be updated. The attribute values of those tuples are to be altered according to the SET clause, setting attribute $A_i = a_i$.

If a subject would like to take a set of tuples from a lower level of the database, modify them, and add them to the subject's own database, then this can be done by a transaction with two Secure SQL commands: an INSERT followed by an UPDATE.

For example, consider relation SOD of Figure 5 again. If an S subject issues the command

```
UPDATE  SOD
SET     Destination = 'Earth'
WHERE   Starship IN
        (SELECT          Starship
        FROM             SOD
        BELIEVED BY      Anyone
        WHERE            Destination = 'Vulcan');
```

meaning "Every starship of mine that anyone (whose beliefs I can see) believes is headed to Vulcan, I now believe is headed to Earth." When issued by an S subject, this update changes the destination of the Enterprise to

SOD Interpretation at level S:

Starship	*Objective*	*Destination*
Enterprise	Diplomacy	Earth
Nighthawk	Warfare	Venus

SOD Interpretation at level $C1$:

Starship	*Objective*	*Destination*
Enterprise	Diplomacy	Romulus

SOD Interpretation at level $C2$:

Starship	*Objective*	*Destination*
Nighthawk	Warfare	Venus
Blackjack	Mining	Pluto

SOD Interpretation at level U:

Starship	*Objective*	*Destination*
Enterprise	Exploration	Vulcan

Fig. 6. An interpretation of a SOD relation: Rerouting to Earth.

'Earth', because U subjects think that the Enterprise is headed to Vulcan. The updated database is given in Figure 6; note that there are no polyinstantiated tuples in the result.

Our semantics permits the scope of an update to be narrowed to a single entity or relationship by specifying its key in the WHERE clause. This is a semantic solution to the problem of tuple proliferation: when a key is specified in p, at most one tuple will be added to the interpretation of a database at a level. For example, if an S subject issues the command

```
UPDATE  SOD
SET     Destination = 'Earth'
WHERE   Starship = 'Enterprise';
```

against Figure 5, the result is the same as in the previous update: no tuples are added to the database, and only one tuple is altered.

5.4 Delete

The Secure SQL DELETE statement has the following general form.

```
DELETE
FROM    R
WHERE   P
```

A DELETE request says that the issuer no longer believes in the existence of the entities or relationships qualified by P. Tuples are selected from the

database interpretation at the level of the issuer, as in an ordinary MLS query (i.e., replace DELETE by SELECT * and execute the resulting query). Then the selected tuples are removed from the database interpretation at that level. No BELIEVED BY clause is needed for DELETE because a subject can only retract belief in entities and relationships that exist at the subject's own level. (Subqueries in the WHERE clause of a deletion may of course contain BELIEVED BY clauses.) Beliefs held at other levels about the deleted entity or relationship will persist.

For example, the following request will delete all starships that are not believed in at any lower level. If issued by a $C2$ subject, the request would delete the Blackjack from Figure 5.

```
DELETE
FROM    SOD
WHERE   Starship NOT IN
        (SELECT         Starship
        FROM            SOD
        BELIEVED BY     AnyoneBelowMe);
```

6. CONCLUSIONS

The goal of our research project is to provide a simple and natural semantics for databases that contain cover stories. In this paper, we have presented a general semantics for the simplest kind of these databases, MLS relational databases, and explored the question of suitable MLS query languages. Our semantics is based on Kripke-like database interpretations, which express the beliefs held at different security levels about the state of the world. Problems present in proposals for MLS relational model syntax, such as ambiguous null values, meaningless queries, and tuple proliferation under updates, can be resolved by supplying a semantics for those proposals, i.e., by giving a mapping from the syntax of those proposals to our database interpretations. We believe that our semantics is useful and general, and that it will be a useful retrofit for other models with existing implementations [Qian 1994a].

We also define a modal relational algebra suitable for use with our semantics, and show how to turn any formal query language into a well-defined MLS query language. We then consider the problem of finding a good MLS extension for SQL, and give definitions for Secure SQL SELECT, INSERT, DELETE, and UPDATE, deriving the effect of each operation from its effect on the semantic interpretation of the relations involved.

In our future work in this area, we plan to extend this work to the more complex hierarchies that we expect to find in discretionary security. We are also examining issues that arise when attempting to share data between secure databases that have different security hierarchies or different MLS models. Recently we have also been examining the question of integrity constraints and secure enforcement mechanisms for integrity constraints, with a goal of delineating families of safe integrity constraints, and their possible enforcement policies.

APPENDIX

Proof of Proposition 4.4.1

We present the proofs ordered by formula number.

(1)

$$
\begin{aligned}
&|\mathbf{B}[E_1](E_2[A_1, \ldots, A_n])|_l \\
&\quad = \bigcup_{\langle l' \rangle \in |E_1 \cap Anyone|_l} |E_2[A_1, \ldots, A_n]|_{l'} \\
&\quad = \bigcup_{\langle l' \rangle \in |E_1 \cap Anyone|_l} (|E_2|_{l'}[A_1, \ldots, A_n]) \\
&\quad = \left(\bigcup_{\langle l' \rangle \in |E_1 \cap Anyone|_l} |E_2|_{l'} \right)[A_1, \ldots, A_n] \\
&\qquad \text{(by commutativity of ordinary union and projection)} \\
&\quad = |(\mathbf{B}[E_1]E_2)|_l[A_1, \ldots, A_n] \\
&\quad = |(\mathbf{B}[E_1]E_2)[A_1, \ldots, A_n]|_l
\end{aligned}
$$

(2) Same proof outline as for previous identity.

(3)

$$
\begin{aligned}
|\mathbf{B}[E_1](E_2 \cup E_3)|_l &= \bigcup_{\langle l' \rangle \in |E_1 \cap Anyone|_l} |E_2 \cup E_3|_{l'} \\
&= \bigcup_{\langle l' \rangle \in |E_1 \cap Anyone|_l} (|E_2|_{l'} \cup |E_3|_{l'}) \\
&= \bigcup_{\langle l' \rangle \in |E_1 \cap Anyone|_l} |E_2|_{l'} \cup \bigcup_{\langle l' \rangle \in |E_1 \cap Anyone|_l} |E_3|_{l'} \\
&\qquad \text{(by associativity of union)} \\
&= |\mathbf{B}[E_1]E_2|_l \cup |\mathbf{B}[E_1]E_3|_l \\
&= |\mathbf{B}[E_1]E_2 \cup \mathbf{B}[E_1]E_3|_l
\end{aligned}
$$

(4)

$$
\begin{aligned}
|\mathbf{B}[E_1](E_2 \times E_3)|_l &= \bigcup_{\langle l' \rangle \in |E_1 \cap Anyone|_l} |E_2 \times E_3|_{l'} \\
&= \bigcup_{\langle l' \rangle \in |E_1 \cap Anyone|_l} (|E_2|_{l'} \times |E_3|_{l'}) \\
&\subseteq \bigcup_{\langle l' \rangle \in |E_1 \cap Anyone|_l} |E_2|_{l'} \times \bigcup_{\langle l' \rangle \in |E_1 \cap Anyone|_l} |E_3|_{l'} \\
&\qquad \left(\text{because } \bigcup_\iota (R_\iota \times S_\iota) \subseteq \bigcup_\iota R_\iota \times \bigcup_\iota S_\iota\right) \\
&= |\mathbf{B}[E_1]E_2|_l \times |\mathbf{B}[E_1]E_3|_l \\
&= |\mathbf{B}[E_1]E_2 \times \mathbf{B}[E_1]E_3|_l
\end{aligned}
$$

(5) Same proof outline as for the previous identity.

(6)

$$
\begin{aligned}
|\mathbf{B}[E_1 \cup E_2]E_3|_l &= \bigcup_{\langle l' \rangle \in |(E_1 \cup E_2) \cap Anyone|_l} |E_3|_{l'} \\
&= \bigcup_{\langle l' \rangle \in (|E_1 \cap Anyone|_l)} |E_3|_{l'} \cup \bigcup_{\langle l' \rangle \in (|E_2 \cap Anyone|_l)} |E_3|_{l'} \\
&= |\mathbf{B}[E_1]E_3|_l \cup |\mathbf{B}[E_1]E_3|_l \\
&= |\mathbf{B}[E_1]E_3 \cup \mathbf{B}[E_1]E_3|_l
\end{aligned}
$$

(7)

$$
\begin{aligned}
|\mathbf{B}[Self]E_1|_l &= \bigcup_{\langle l' \rangle \in |Self \cap Anyone|_l} |E_1|_{l'} \\
&= \bigcup_{l' = l} |E_1|_{l'} \\
&= |E_1|_l
\end{aligned}
$$

(8)

$$
\begin{aligned}
|\mathbf{B}[E_1]R|_l &= \bigcup_{\langle l' \rangle \in |E_1 \cap Anyone|_l} |R|_{l'} \\
&= \bigcup_{\langle l' \rangle \in |E_1 \cap Anyone|_l} R \\
&= \begin{cases} R & \text{if } |E_1 \cap Anyone|_l \text{ is nonempty} \\ \{\ \} & \text{otherwise} \end{cases} \\
&= |R \ltimes (R \times (E_1 \cap Anyone))|_l
\end{aligned}
$$

(9)

$$
\begin{aligned}
|\mathbf{B}[E_1](\mathbf{B}[R]E_3)|_l &= \bigcup_{\langle l' \rangle \in |E_1 \cap Anyone|_l} |\mathbf{B}[R]E_3|_{l'} \\
&= \bigcup_{\langle l' \rangle \in |E_1 \cap Anyone|_l} \bigcup_{\langle l'' \rangle \in |R \cap Anyone|_{l'}} |E_3|_{l''} \\
&= \bigcup_{\langle l' \rangle \in |E_1 \cap Anyone|_l} \bigcup_{l'' | \langle l'' \rangle \in R \wedge l'' \le l'} |E_3|_{l''} \\
&= \bigcup_{l'' | \exists l' \ \langle l' \rangle \in |E_1 \cap Anyone|_l \wedge \langle l'' \rangle \in R \wedge l'' \le l'} |E_3|_{l''} \\
&= \bigcup_{\langle l'' \rangle \in |R|_l \cap (|Anyone|_l \ltimes_{label \le E_1.A} |E_1|_l)} |E_3|_{l''} \\
&= \bigcup_{\langle l'' \rangle \in |R \cap (Anyone \ltimes_{label \le E_1.A} E_1)|_l} |E_3|_{l''} \\
&= \left|\mathbf{B}\left[R \cap (Anyone \ltimes_{label \le E_1.A} E_1)\right] E_3\right|_l
\end{aligned}
$$

(10)

$$
\begin{aligned}
|\mathbf{B}[E_1](\mathbf{B}[Anyone]E_2)|_l &= \bigcup_{\langle l' \rangle \in |E_1 \cap Anyone|_l} |\mathbf{B}[Anyone]E_2|_{l'} \\
&= \bigcup_{\langle l' \rangle \in |E_1 \cap Anyone|_l} \bigcup_{\langle l'' \rangle \in |Anyone|_{l'}} |E_2|_{l''} \\
&= \bigcup_{\langle l' \rangle \in |E_1 \cap Anyone|_l} \bigcup_{l'' | l'' \leq l'} |E_2|_{l''} \\
&= \bigcup_{l'' | \exists l' \, \langle l' \rangle \in |E_1 \cap Anyone|_l \wedge l'' \leq l'} |E_2|_{l''} \\
&= \bigcup_{\langle l'' \rangle \in |Anyone|_l \ltimes_{label \preceq E_1 A} |E_1|_l} |E_2|_{l''} \\
&= \bigcup_{\langle l'' \rangle \in |Anyone \ltimes_{label \preceq E_1 A} E_1|_l} |E_2|_{l''} \\
&= |\mathbf{B}[Anyone \ltimes_{label \preceq E_1 A} E_1]E_2|_l
\end{aligned}
$$

ACKNOWLEDGMENTS

The authors greatly appreciate the support of Sushil Jajodia. We are also greatly indebted for discussions with Tom Garvey, Arthur Keller, Teresa Lunt, and Bhavani Thuraisingham, and the comments of the anonymous referees.

REFERENCES

AKL, S. AND DENNING, D 1987. Checking classification constraints for consistency and completeness. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy* (Oakland, Calif. April). IEEE Computer Society, Washington, D.C., 196–201.

BELL, D. E. AND LAPADULA, L. J. 1974. Secure computer systems: Mathematical foundations and model. Tech. Rep. MITRE Corporation, McLean, Va.

BINNS, L. 1993. Inference and cover stories. In *Database Security, VI Status and Prospects*, North-Holland, Amsterdam, 169–178.

BINNS, L. 1992a. Inference through polyinstantiation In *Research Directions in Database Security, IV*, R. Burns, Ed. Mitre Tech Rep. M92B0000 118, Sept 1992, 74–84.

BINNS, L. 1992b Inference through secondary path analysis. In *Database Security, VI: Status and Prospects*. North-Holland, Amsterdam, 195–209.

BONATTI, P., KRAUS, S., AND SUBRAHMANIAN, V. 1992. Declarative foundations of secure deductive databases. In the *International Conference on Database Theory*. Springer-Verlag, Berlin

CERI, S. AND WIDOM, J. 1990. Deriving production rules for constraint maintenance In *Proceedings of the 16th International Conference on Very Large Data Bases* (August). Morgan Kauffmann, Palo Alto, Calif., 566–577.

CHELLAS, B. F. 1980. *Modal Logic*. Cambridge University Press, Cambridge, Mass

COSTICH, O. AND MCDERMOTT, J. 1992 A multilevel transaction problem for multilevel secure database systems and its solution for the replicated architecture. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy* (Oakland, May). IEEE Computer Society, Washington, D.C., 192–203.

CUPPENS, F 1992 A modal logic framework to solve aggregation problems. In *Database Security V: Status and Prospects*, C Landwehr and S. Jajodia, Eds. North-Holland, Amsterdam, 315–332

DENNING, D. 1976. A lattice model of secure information flow. *Commun. ACM 19*, 5, 236–243.

DENNING, D., LUNT, T., SCHELL, R., HECKMAN, M., AND SHOCKLEY, W. 1987. A multilevel relational data model. In *Proceedings of the IEEE Symposium on Research in Security and Privacy* (Oakland, April). IEEE, New York, 220–234.

DEPARTMENT OF DEFENSE. 1985. Department of Defense Trusted Computer System Evaluation Criteria. DOD 5200.28-STD, National Computer Security Center, Washington, D.C.

FROSCHER, J. N. AND MEADOWS, C. 1989. Achieving a trusted database management system using parallelism. In *Database Security II: Status and Prospects*. North Holland, Amsterdam.

GLASGOW, J., MACEWEN, G., AND PANANGADEN, P. 1992. A logic for reasoning about security. *ACM Trans. Comput. Syst. 10*, 3 (Aug.).

HAIGH, J., O'BRIEN, R., AND THOMSEN, D. 1991. The LDV secure relational DBMS model. In *Database Security, IV: Status and Prospects*, S. Jajodia and C. Landwehr, Eds. North-Holland, Amsterdam, 265–279.

JAJODIA, S. AND SANDHU, R. 1991. Toward a multilevel secure relational data model. In *Proceedings: ACM SIGMOD* (Denver, Colo., May). ACM, New York, 50–59.

JAJODIA, S. AND SANDHU, R. 1990. Polyinstantiation integrity in multilevel relations. In *Proceedings: IEEE Symposium on Research in Security and Privacy* (Oakland, May). IEEE, New York, 104–115.

KORTH, H. F. AND SILBERSCHATZ, A. 1991. *Database System Concepts*. McGraw-Hill, New York.

IMIELINSKI, T. AND LIPSKI, W. 1984. Incomplete information in relational databases. *J. ACM 31*, 4 (Oct.).

LIU, K.-C. AND SUNDERRAMAN, R. 1990 Indefinite and maybe information in relational databases. *ACM Trans. Database Syst. 15*, 1 (Mar.), 1–39.

QIAN, X. 1994a. A model-theoretic semantics of the multilevel secure relational model. In *Proceedings of the International Conference on Extending Database Technology* (Cambridge, UK, Mar.). Springer-Verlag, Berlin.

QIAN, X. 1994b. Inference channel-free integrity constraints in multilevel relational databases. In *Proceedings of the IEEE Symposium on Research in Security and Privacy* (Oakland, May). IEEE, New York.

QIAN, X. AND LUNT, T. 1992. Tuple-level vs element-level classification. In *Proceedings of the 6th IFIP Working Conference on Database Security* (Vancouver, BC, Aug.). IFIP, 311–324.

RABITTI, F., BERTINO, E., KIM, W., AND WOELK, D. 1991. A model of authorization for next-generation database systems. *ACM Trans. Database Syst. 16*, 1 (Mar.).

SANDHU, R. AND JAJODIA, S. 1992. Polyinstantiation for cover stories. In *Proceedings of the European Symposium on Research in Computer Security* (Toulouse, France, Nov.). Springer-Verlag, Berlin, 307–328.

SICHERMAN, G. L., DE JONGE, W., AND VAN DE RIET, R. P. 1983. Answering queries without revealing secrets. *ACM Trans. Database Syst. 8*, 1 (Mar), 41–59.

SMITH, G. 1990 The modeling and representation of security semantics for database applications. Ph.D. thesis, George Mason Univ., Fairfax, Va.

SMITH, K. AND WINSLETT, M. 1992a. Entity modeling in the MLS relational model. In *Proceedings of the 18th International Conference on Very Large Data Bases*. (August). VLDB Endowment, 199–210.

SMITH, K. AND WINSLETT, M. 1992b. Multilevel secure rules: Integrating the multilevel secure and active data models. In *Proceedings of the 6th IFIP Working Conference on Database Security* (Vancouver, BC, Aug.). IFIP, 33–58.

STONEBRAKER, M., HANSON, E. N., AND POTAMIANOS, S. 1988. The POSTGRES rule manager. *IEEE Trans. Softw. Eng. 14*, 7 (July), 897–907.

SU, T. AND OZSOYOGLU, G. 1987. Data dependencies and inference control in multilevel relational database systems. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy* (Oakland, May). IEEE Computer Society Press, Los Alamitos, Calif. 202–211

THURAISINGHAM, B. 1992. Handling security constraints during multilevel database design. In *Research Directions in Database Security, IV*, Rae Burns, Ed. Mitre Tech Rep. M92B0000 118, Mitre Corp., McLean, Va.

TING, T., DEMURJIAN, S., AND HU, M. 1992. A specification methodology for user-role based

security in an object-oriented design model: Experience with a health care application. In *Proceedings of the 6th IFIP Workshop on Database Security* (Vancouver, BC, Canada). IFIP.

ULLMAN, J. D. 1988. *Principles of Database and Knowledge Base Systems*. Vol 1. Computer Science Press, Rockville, Md

WIDOM, J. AND FINKELSTEIN, S. J. 1990. A syntax and semantics for set-oriented production rules in relational database systems In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Atlantic City, N.J., May). ACM, New York, 259–270.

ZANIOLO, C. 1984. Database relations with null values. *J. Comput. Syst. Sci. 1*, 28 (Feb.), 142–166.

Received October 1993; revised April 1994; accepted April 1994