Proceedings of IEEE Computer Security Foundations Workshop VI, Franconia, NH, June 1990, pages 152-158.

A Formal Framework for Single Level Decomposition of Multilevel Relations

Sushil Jajodia* & Ravi Sandhu*

Department of Information and Software Systems Engineering
George Mason University, Fairfax, VA 22030

Abstract

In this paper, we consider multilevel relations in which security classifications are assigned at the granularity of individual data elements. Usually these multilevel relations exist only at the logical level. In reality, a multilevel relation is decomposed into a collection of single level base relations which are then physically stored in a database, and a recovery algorithm is used to reconstruct the original multilevel relation. In this paper we formalize the relationship that exists between the decomposition-independent filtered relations and the multilevel relations obtained from decomposed single level relations using the recovery algorithm. We state three requirements that must be met by any decomposition and recovery algorithms. In particular our algorithms in [7] meet these requirements.

1 INTRODUCTION

In recent years, there have been several efforts to build multilevel secure relational database management systems. A major issue is how access classes are assigned to data stored in relations. The proposals have ranged from assigning access class to relations, assigning access classes to individual tuples in a relation, or assigning access classes to individual attributes of a relation.

Unlike these proposals, in the SeaView (Secure Data Views) project security classifications are assigned to individual data elements of the tuples of a relation. See table 1. This project began as a joint effort by SRI International and Gemini Computers with the goal of designing and prototyping a multilevel secure relational database management system that satisfies the Trusted Computer System Evaluation Criteria for Class A1 [5]. SeaView researchers have considerably advanced the state of the art in multilevel database security and the project itself has moved to a prototype implementation phase using GEMSOS as the underlying TCB along with the ORACLE relational DBMS [9]. SeaView has been extensively described [4, 8, 9, 10, for instance].

In SeaView, subjects having different clearances see different versions of the multilevel relation. A user having a clearance at an access class sees only that data which lies at that access class or below. Thus, a user with Top Secret clearance will see the entire relation in table 1, while a user having Secret clearance will see the filtered relation given in table 2.

Multilevel relations in SeaView exist only at the logical level. In reality multilevel relations are decomposed into a collection of single level base relations which are then physically stored in the database. Completely transparent to users, multilevel relations can be reconstructed from these base relations on user demand. The practical advantages of being able to decompose and store a multilevel real relation by a collection of single level base relations are almost obvious. In particular the TCB can enforce mandatory controls with respect to the single level base relations which allows the DBMS to mostly run as an untrusted application on the TCB.

Although there have been some changes in SeaView definitions and concepts it has for the most part remained remarkably stable indicating that its foundation is a sound one. Unfortunately there are aspects of SeaView's decomposition of a multilevel relation into single level ones which have not been stated or analyzed with the same rigor devoted to its other aspects.

*This research was supported (partially) by the Center for Excellence in Command, Control, Communications, and Intelligence at George Mason University. The Center's general research program is sponsored by the Virginia Center for Innovative Technology, MITRE Corporation, the Defense Communications Agency, CECOM, PRC/ATI, ASD (C3I), TRW, AFCEA, and AFCEA NOVA.

A_1	C_1	A_2	C_2	A_3	C_3	TC
mad	S	17	S	x	S	S
foo	S	34	S	w	TS	TS
ark	TS	5	TS	y	TS	TS

Table 1: A Multilevel Relation R

A_1	C_1	A_2	C_2	A_3	C_3	TC
mad	S	17	S	x	S	S
foo	S	34	S	null	S	S

Table 2: A Secret Instance of R

As a result there are many subtle and nontrivial issues which have been overlooked. Some of these were pointed out by us in [7].

In this paper, we take a closer look at single level decomposition of multilevel relations. Since a multilevel relation is stored as single level relations, we need two algorithms.

1. A *decomposition* algorithm which breaks multilevel relations into single level relations.

2. A *recovery* algorithm to reconstruct original multilevel relations from single level ones.

How these two functions are related is formalized in our first requirement in Section 4. The requirement corresponds to the "lossless join property" in the standard relational theory.

We need to require more in a multilevel world since a multilevel relation at an access class induces a family of relation instances, one at each access class in the security lattice. Likewise, when a multilevel relation is decomposed into single level relations, these single level relations are themselves partitioned among groups of relations, one group (possibly empty) corresponding to each descending access class in the security lattice. Thus a relation instance at an access class c can be obtained in two different ways.

1. Directly from a higher level multilevel relation instance by filtering out data not dominated by c.

2. Indirectly from a higher level multilevel relation instance by (i) decomposing at the higher level into an equivalent collection of single level relations, (ii) casting aside those single level relations not dominated by c, and (iii) finally reconstructing the multilevel relation instance at c from the collection in step (ii).

It is obvious that these two ways of arriving at the c instance of a multilevel relation must yield identical results, otherwise our decomposition and recovery is simply incorrect.

There are conversely two different ways of arriving at a collection of single level relations equivalent to a multilevel relation instance at access class c.

1. Directly decomposing the c instance of the multilevel relation.

2. Indirectly from the decomposition of some c' multilevel relation instance for $c' > c$ by (i) decomposing at the c' instance into its equivalent collection of single level relations, and (ii) casting aside those single level relations not dominated by c.

From a security perspective these two methods must give us precisely the same result. Otherwise we will have interference from c' to c opening up covert channels for leakage of information.

In this paper, we formalize the relationship that exists between the decomposition-independent filtered relations and the multilevel relations obtained from the appropriate single level relations. We state two additional requirements in Section 6 corresponding to the two requirements informally outlined above. Our decomposition and recovery algorithms of [7] are examples of algorithms that meet all three requirements.

2 BASIC CONCEPTS

The standard relational model [1, 2, 3] is concerned with data without security classifications. Data are stored in relations which have well defined mathematical properties.

A *relation scheme* R is a collection of attributes names $A_1, A_2, \ldots, A_n$ where each A_i corresponds to some *domain* D_i which is a set of values.

A *relation* over R is a set of distinct *tuples* of the form $(a_1, a_2, \ldots, a_n)$ where each *element* a_i is a value in domain D_i.

Not all possible relations are meaningful in an application; only those that satisfy certain integrity constraints (usually entity and referential integrity constraints defined below) are considered valid.

Let X and Y denote sets of one or more of the attributes A_i in a relation scheme R. We say Y is *functionally dependent* on X, written $X \rightarrow Y$, if given any relation over R, it is not possible to have two tuples in the relation with the same values for X but different values for Y. A *candidate key* of a relation scheme (or relation) is a minimal set of attributes on which all other attributes are functionally dependent. It is minimal in the sense that no attribute can be discarded without destroying this property. It is guaranteed that a candidate key always exists, since in the absence of any functional dependencies it consists of the entire set of attributes. There can be more than one candidate key for a relation with a given collection of functional dependencies.

The *primary key* of a relation is one of its candidate keys which has been specifically designated as such. The primary key serves the purpose of selecting a specific tuple from a relation instance as well as of linking relations together.

The standard relational model incorporates two integrity rules, called *entity integrity* and *referential integrity*. Our focus is on the former rule since the latter is not relevant to the topic of this paper. Entity integrity simply requires that no tuple in a relation instance can have null values for any of the primary key attributes. This property guarantees that each tuple will be uniquely identifiable.

Since we wish to introduce *nulls* in multilevel relations, we need to define some notions and notations used with relations whose tuples may contain nulls. It is well known that null values in relational databases result in tricky problems. Fortunately, nulls in multilevel relations arise due to security considerations in a specific manner which allows us to deal with them cleanly and rigorously for the problems considered in this paper.

We use a single type of null value. By a *general relation* we mean a relation over one or more attributes that are allowed to have null values. Henceforth we will understand relation to mean general relation and will use the latter term only for added emphasis where appropriate. Let G be a general relation on attributes $A_1, \ldots, A_n$. Let t and s be two tuples in an instance of G. We say t *subsumes* s if for every attribute A_i, either $t[A_i] = s[A_i]$ or $s[A_i] =$ null. That is t and s agree everywhere except possibly for some attributes where s has a null value and t a non-null value. G is said to be *subsumption free* if it does not contain two tuples such that one subsumes the other. Finally, the nature of functional dependencies with null values also needs clarification. Let X and Y be subsets of $A_1, \ldots, A_n$. A tuple t is X-total if it has no null value for attributes in X. We say the *null-valued functional dependency* (NFD) $X \rightarrow Y$ is satisfied by G if for all X-total tuples $t, t' \in G$ such that $t[X] = t'[X]$, we have that $t[Y] = t'[Y]$. Note that $t[Y]$ and $t'[Y]$ may contain nulls, and nulls are equal only to other nulls. Henceforth we understand the term functional dependency to mean NFD.

In the sequel, we assume that unless otherwise stated all relations are made subsumption free by exhaustive elimination of subsumed tuples.

3 MULTILEVEL RELATIONS

Moving on to a multilevel world, we define *multilevel relations* by extending the definitions given in the previous section for the standard relational model. The extension it turns out is not straight-forward. Unlike the standard relational model where there is a single relation corresponding to each relation scheme, a multilevel relation scheme has different instances at different access classes. Thus, the notion of a key is inherently more complex than for a standard relation. While in a standard relation the definition of candidate keys is based on that of functional dependencies, in a multilevel setting the concept of functional dependencies is itself clouded because a relation instance is now a collection of sets of tuples rather than a single set of tuples. Rather than trying to resolve this complex issue here, we follow the lead of SeaView and assume there is a user specified primary key AK consisting of a subset of the data attributes A_i. This is called the *apparent primary key* of the multilevel relation scheme. Henceforth we understand the term primary key as synonymous with apparent primary key.

In order to simplify the notation, we use A_1 instead of AK from now on. It should be understood, however, that in general A_1 will consist of multiple attributes. Our definition now consists of two parts:

Definition 1 [MULTILEVEL RELATION SCHEME] A state-invariant multilevel relation scheme

$$R(A_1, C_2, A_2, C_2, \ldots, A_n, C_n, TC)$$

where each A_i is as before an attribute over domain D_i, each C_i is a *classification attribute* for A_i and TC is the *tuple-class* attribute. The domain of C_i is specified by a range $[L_i, H_i]$ which defines a sub-lattice of access classes ranging from L_i up to H_i. The domain of TC is $[\text{lub}\{L_i\}, \text{lub}\{H_i\}]$. $\square$

Definition 2 [RELATION INSTANCES] A collection of state-dependent *relation instances*

$$R_c(A_1, C_2, A_2, C_2, \ldots, A_n, C_n, TC)$$

one for each access class c in the given lattice. Each instance is a subsumption free set of distinct tuples of the form $(a_1, c_1, a_2, c_2, \ldots, a_n, c_n, tc)$ where each $a_i \in D_i$, $c \geq c_i$ and $tc = \text{lub}\{c_i\}$. Moreover, if a_i is not null then $c_i \in [L_i, H_i]$. We require that c_i be defined even if a_i is null, i.e., a classification attribute cannot be null. Since tc is computed from the other classification attributes from now on we will include it or omit it as convenient. □

The multiple relation instances are, of course, related; each instance is intended to represent the version of reality appropriate for each access class. Roughly speaking, each element $t[A_i]$ in a tuple t is visible in instances at access class $t[C_i]$ or higher; $t[A_i]$ is replaced by a null value in an instance at a lower access class. We will give a more formal description using the filter function in Section 5.

It seems appropriate to consider the semantic of null values in tuples. A null value has two interpretations: the first corresponds to the usual semantics in the standard relational theory depending on the context and the second corresponds to security considerations. Thus, a null value could be interpreted as an unknown value which exists but is not recorded (for whatever reason), as a nonexistent value (such as an unassigned phone number), or as an inapplicable value (such as maiden name of a male employee). In the security context a null could also mean that a value, if it exists, cannot be seen at that access class.

Similar to the standard relational model, not all relation instances R_c are valid in an application; only those that satisfy certain integrity constraints are valid. For now we assume that there is a set F which specifies all constraints, and we enforce these constraints in all valid instances. Since different models have proposed different sets of constraints [11], for now we choose to not be explicit about the contents of the set F. We will state in Section 6 some properties we require of decomposition and recovery algorithms. The point we wish to emphasize is that these requirements are sensible, independent of the exact choice of F or even the exact decomposition and recovery algorithms.

4 LOSSLESS DECOMPOSITION

Since we store a multilevel relation R_c as a collection of single level relations, it is reasonable to require that the information contained in R_c must be equivalent to the information contents of the single level relations. We formalize the notion of equivalence as follows.

The Decomposition Function

Let $R_c(A_1, C_1, A_2, C_2, \ldots, A_n, C_n)$ be a multilevel relation. We assume that there is a *decomposition* function ρ which takes a multilevel relation R_c and yields a collection $\mathbf{R}_c$ of single level relations $\{\hat{R}_{j,c_j} : j = j_1, \ldots, j_n\}$, where the access class of $\hat{R}_{j,c_j}$ is c_j. In other words,

$$\rho(R_c) = \mathbf{R}_c = \{\hat{R}_{j,c_j} : j = j_1, \ldots, j_n\}$$

We wish to be as general as possible in developing our framework, so we will not constrain the $\hat{R}$ relations very much. These relations are single level in the sense that they should satisfy all requirements of traditional relational theory. The attributes of these relations are simply data attributes, and there is no formal concept of a classification attribute. That is, when some C_i does figure as an attribute of $\hat{R}$ it is formally treated as a piece of data just as an A_j attribute would be. One would expect the attributes of the $\hat{R}$'s to be subsets of the attributes of R_c, but we do not make this a requirement.

It is reasonable to require that if we start with two different multilevel relations R_c and S_c, then ρ will yield different collections of single level relations. That is we require that ρ be a one-to-one function: if $R_c \neq S_c$, then $\rho(R_c) \neq \rho(S_c)$.

The Recovery Function

Next, we wish to have each tuple in R_c be somehow recoverable from the tuples in the collection $\rho(R_c) = \{\hat{R}_{j,c_j} : j = j_1, \ldots, j_n\}$. Thus, we require a *recovery* function τ which takes as input the single level $\hat{R}$ relations and reconstructs the multilevel relation R_c. In other words, we require that

$$\tau(\{\hat{R}_{j,c_j} : j = j_1, \ldots, j_n\}) = R_c$$

Lossless Requirement

Obviously these two functions ρ and τ have to be related; one is an "inverse" of the other, as defined

$$R_c \longrightarrow \boxed{\mathbf{R}_c = \rho(R_c)} \longrightarrow \mathbf{R}_c$$

$$R_c \longleftarrow \boxed{R_c = \tau(\mathbf{R}_c)} \longleftarrow \mathbf{R}_c$$

Figure 1: Requirement 1

below. This gives us the first of our three requirements for decomposition and recovery functions.

Requirement 1 Let ρ and τ denote the decomposition and recovery functions, respectively. For any multilevel relation R_c, $\tau \circ \rho(R_c) = R_c$. $\Box$

This requirement can be visualized as shown in figure 1. The proof outlined in [4] for the proposed SeaView decomposition described there amounts to proving requirement 1.

Requirement 1 corresponds to the "lossless join property" in the standard relational theory. If we end up with more tuples during reconstruction than what we had originally in R_c, then we have lost some information. Our first requirement guarantees that when we decompose tuples in a multilevel relation into smaller tuples, only the original tuples can be recovered; unwanted combinations never occur.

One might attempt to extend requirement 1 to include its converse, i.e., $\rho \circ \tau(\mathbf{R}_c) = \mathbf{R}_c$. This is however too strong because the decomposition ρ may not be onto. So the domain of τ will have elements outside the range of ρ. The converse of requirement 1 should hold for those $\mathbf{R_C}$ in the range of $\rho(R_c)$. This will follow from the requirements we will be formulating.

Before we can state our other two requirements, we need to define precisely how various instances of a multilevel relation scheme are related. This is done using the filter function, defined in the next section.

5 FILTERED INSTANCES

In this section, we formally define a *filter function* which maps a multilevel relation to different instances, one for each descending access class in the security lattice. The filter function limits each user to that portion of the multilevel relation for which he or she holds a clearance.

Definition 3 [Filter Function] Given the c-instance R_c of a multilevel relation the filter function σ produces the c'-instance $R_{c'} = \sigma(R_c, c')$ for $c' \leq c$. A tuple $t' \in R_{c'}$ if and only if t' can be derived from some $t \in R_c$ as follows:

$$\begin{aligned} t'[A_1, C_1] &= t[A_1, C_1] \\ t'[A_i, C_i] &= \begin{cases} t[A_i, C_i] & \text{if } t[C_i] \leq c' \\ < \text{null}, c_1 > & \text{otherwise} \end{cases} \quad \text{for } 1 < i \leq n \end{aligned}$$

$\Box$

The following properties of σ are easily verified.

1. $\sigma(R_c, c) = R_c$.
2. For $c'' < c' < c$, $\sigma(\sigma(R_c, c'), c'') = \sigma(R_c, c'')$

The first property states that filtering a relation instance at its own level has no effect. The second states that filtering twice successively at descending levels has the same effect as filtering directly to the second level. Both properties are natural ones to expect of a filter function.

Now, we can use σ to describe how the various instances R_c of a relation scheme are related. Requiring the instances of a multilevel relation to be related by σ gives the inter-instance property of SeaView.

Definition 4 [Inter-Instance Property] Let R be a relation scheme, and let R_c and $R_{c'}$ be two relation instances of R such that $c' < c$. Then we have that $\sigma(R_c, c') = R_{c'}$. $\Box$

6 FORMAL FRAMEWORK

As we have seen, we can view a multilevel relation R_c in two different ways: One way to view R_c is as a collection of instances at different access classes by application of the filter function. The other way to view R_c is as a family $\mathbf{R}_c$ of single level relations obtained using the decomposition algorithm. In this section, we show how these different views fit together in our formal framework. First we require one more definition.

Definition 5 [Projection Function] Let $\mathbf{R}_c = \{\hat{R}_{j,c_j} : j = j_1, \ldots, j_n\}$ be a collection of single level relations such that the access class of $\hat{R}_{j,c_j}$ is c_j. Given an access class c', we define the *projection* of $\mathbf{R}_c$ at access class c' as follows:

$$\mathbf{R}_{c'} = \pi(\mathbf{R}_c, c') = \{\hat{R}_{k,c_k} : \hat{R}_{k,c_k} \in \mathbf{R}_c \wedge c_k \leq c'\} \quad \Box$$

Note that the projection function is a given. Its definition captures the mandatory access control for reads as applied to the storage objects containing the $\hat{R}$ relations. Note that $\mathbf{R}_{c'} \subseteq \mathbf{R}_c$ and $\mathbf{R}_{c'}$ may possibly be empty.

Now we can state our remaining two requirements.

Requirement 2 Let R_c be a multilevel relation, and let c' be an access class such that $c' \leq c$. As before, let σ denote the filter function, and ρ and τ denote the decomposition and recovery functions, respectively. Let $\mathbf{R}_c = \{\hat{R}_{j,c_j} : j = j_1, \ldots, j_n\}$ where each $\hat{R}_{c_j}$ is a single level relation at access class c_j.

We can derive the c'-view of R_c in one of the following two ways:

1. Directly from R_c by applying the filter function σ to R_c to obtain the c'-instance: $R_{c'} = \sigma(R_c, c')$.
2. From decomposed single level relations $\mathbf{R}_c = \{\hat{R}_{j,c_j} : j = j_1, \ldots, j_n\}$ by
   (a) first selecting those relations which are at or below access class c' by means of $\pi(\mathbf{R}_c, c')$, and then
   (b) applying the recovery function τ to this latter collection of single level relations.

Our requirement is that both ways produce identical results. That is,

$$\sigma(R_c, c') = \tau \circ \pi(\rho(R_c), c') \qquad \square$$

We can describe the above situation in terms of the diagram given in figure 1. Our second requirement is that this diagram must be commutative. Requirement 2 is a generalization of requirement 1, which is necessary in the context of multilevel relations. Figure 1 is a special case of figure 2 when $c = c'$, in which event both σ and π are trivially the identity transformation.

We regard requirement 2 as stating the correctness criteria for the decomposition and recovery algorithms. Our next requirement, illustrated in figure 3, states the security criteria. It is based on the supposition that $\mathbf{R}_{c'}$ will be visible to c' subjects. Therefore it must appear as though $\mathbf{R}_{c'}$ was created directly at c' rather than being derived by projection from $\mathbf{R}_c$. Otherwise we have interference from higher to lower security classes.

Requirement 3 Let $R_{c'}$ be a multilevel relation such that $R_{c'} = \sigma(R_c, c')$, and let ρ denote the decomposition function. We can decompose $R_{c'}$ into a set of single level relations in two different ways:

$$\begin{tikzcd}
R_c \arrow[r] \arrow[d] & \boxed{\mathbf{R}_c = \rho(R_c)} \arrow[r] & \mathbf{R}_c \arrow[d] \\
\boxed{R_{c'} = \sigma(R_c, c')} \arrow[d] & & \boxed{\mathbf{R}_{c'} = \pi(\mathbf{R_c}, c')} \arrow[d] \\
R_{c'} & \boxed{R_{c'} = \tau(\mathbf{R}_{c'})} \arrow[l] & \mathbf{R}_{c'} \arrow[l]
\end{tikzcd}$$

Figure 2: Requirement 2

$$\begin{tikzcd}
R_c \arrow[r] \arrow[d] & \boxed{\mathbf{R}_c = \rho(R_c)} \arrow[r] & \mathbf{R}_c \arrow[d] \\
\boxed{R_{c'} = \sigma(R_c, c')} \arrow[d] & & \boxed{\mathbf{R}_{c'} = \pi(\mathbf{R_c}, c')} \arrow[d] \\
R_{c'} \arrow[r] & \boxed{\mathbf{R}_{c'} = \rho(R_{c'})} \arrow[r] & \mathbf{R}_{c'}
\end{tikzcd}$$

Figure 3: Requirement 3

1. Apply the decomposition function ρ to $R_{c'}$ directly to obtain a collection of single level relations $\mathbf{S}_{c'} = \{\hat{S}_{k,c_k} : k = k_1, \ldots, k_m\}$ where each $\hat{S}_{k,c_k}$ is a single level relation at access class c_k.

2. First decompose R_c into single level relations $\{\hat{R}_{j,c_j} : j = j_1, \ldots, j_n\}$ by applying the decomposition function ρ to R_c, and then select those relations from $\rho(R_c)$ which are at or below access class c' to get $\mathbf{R}_{c'}$.

Our second requirement is that both ways produce identical results. That is

$$\rho \circ \sigma(R_c, c') = \pi(\rho(R_c), c') \qquad \square$$

In other words our third requirement is that the diagram of figure 3 must be commutative.

To make requirement 3 more concrete consider the hypothetical decomposition shown in table 3(a), where the multilevel relation R_{S} is mapped by ρ to the collection $\mathbf{R}_{\mathrm{S}}$ of single level relations $\hat{R}_{1,\mathrm{S}}$, $\hat{R}_{2,\mathrm{U}}$ and $\hat{R}_{3,\mathrm{U}}$. Each of these $\hat{R}$'s consists of a single tuple. Projecting $\mathbf{R}_{\mathrm{S}}$ to get $\mathbf{R}_{\mathrm{U}}$ leaves us with $\hat{R}_{2,\mathrm{U}}$ and $\hat{R}_{3,\mathrm{U}}$ as shown in table 3(b). Say that our hypothetical recovery algorithm gives us R_{U} as shown there. This postulated recovery is very plausible since the tuple in $\hat{R}_{3,\mathrm{U}}$ is simply subsumed by that in $\hat{R}_{2,\mathrm{U}}$. Yet it is difficult to think of a decomposition which will yield $\mathbf{R}_{\mathrm{U}}$ of table 3(b) from R_{U}. The decomposition of table 3(c) is far more plausible. But then tables 3(b) and 3(c) collectively violate requirement 3.

The decomposition and recovery algorithms of [7] are examples of algorithms that satisfy all three requirements. An outline for the proof of requirement 2 is given in [7] and requirement 3 can be similarly proved. Requirement 1 is of course a special case of requirement 2. The proof outlined in [4] for the proposed SeaView decomposition described there amounts to proving requirement 1. Since the SeaView algorithms are based on the outer join operation it will require greater effort to prove requirements 2 and 3 as compared with proving these for the algorithms of [7].

7 CONCLUSION

In this paper we have provided a conceptual framework for dealing with multilevel real relations as a collection of single level base relations. We have formalized the relationship that exists between the decomposition-independent filtered relations and the multilevel relations obtained from decomposed single level relations using the recovery algorithm. We state three requirements that must be met by any decomposition and recovery algorithms. Our decomposition and recovery algorithms in [7] meet these requirements.

In terms of future work much remains to be done. The efficiency of the recovery algorithm is clearly crucial to the query response time. It is therefore important to consider further optimizations to our recovery algorithm of [7]. Since we decompose a multilevel real relation as a collection of single-level base relations, it remains to show that an update to a multilevel relation can be correctly translated into equivalent updates to base relations, and conversely. This will provide a formal basis for the updatability of multilevel relations *vis-a-vis* base relations. A formal consideration of updates is also necessary to show that the data model does not contain covert channels.

Acknowledgement

We are indebted to John Campbell, Joe Giordano, and Howard Stainer for their support and encouragement, making this work possible.

References

[1] Codd, E.F. "A Relational Model of Data for Large Shared Data Banks." *Communications of ACM* 13(6): (1970).

[2] Codd, E.F. "Extending the Relational Database Model to Capture More Meaning." *ACM Transactions on Database Systems* 4(4): (1979).

[3] Date, C.J. *An Introduction to Database Systems.* Volume I, Addison-Wesley, fourth edition (1986).

[4] Denning, D.E., Lunt, T.F., Schell, R.R., Shockley, W.R. and Heckman, M. "The SeaView Security Model." *IEEE Symposium on Security and Privacy*, 218-233 (1988).

[5] Department of Defense National Computer Security Center. *Department of Defense Trusted Computer Systems Evaluation Criteria.* DoD 5200.28-STD, (1985).

[6] Gajnak, G.E. "Some Results from the Entity-Relationship Multilevel Secure DBMS Project." *Aerospace Computer Security Applications Conference*, 66-71 (1988).

[7] Jajodia, S. and Sandhu, R.S. "Polyinstantiation Integrity in Multilevel Relations." *IEEE Symposium on Security and Privacy*, Oakland, California, May 1990, to appear.

[8] Lunt, T.F., Denning, D.E., Schell, R.R. Heckman, M. and Shockley, W.R. "Element-Level Classification with A1 Assurance." *Computers & Security*, Feb. 1988.

[9] Lunt, T.F., Schell, R.R., Shockley, W.R., Heckman, M. and Warren, D. "A Near-Term Design for the SeaView Multilevel Database System." *IEEE Symposium on Security and Privacy*, 234-244 (1988).

[10] Lunt, T.F., Denning, D.E., Schell, R.R. Heckman, M. and Shockley, W.R. "Secure Distributed Data Views. Volume 2: The SeaView Formal Security Policy Model." SRI-CSL-88-15 (1989).

[11] Sandhu, R. S., Jajodia, S. and Lunt, T. "A New Polyinstantiation Integrity Constraint for Multilevel Relations." *IEEE Workshop on Computer Security Foundations*, Franconia, New Hampshire, June 1990, to appear.