NCSC TECHNICAL REPORT - 005
Volume 3/5
Library No. S-243,039

## FOREWARD

This report is the third of five companion documents to the *Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria*. The companion documents address topics that are important to the design and development of secure database management systems, and are written for database vendors, system designers, evaluators, and researchers. This report addresses polyinstantiation issues in multilevel secure database management systems.

[illegible]

Keith F. Brewster
Acting Chief, Partnerships and Processes

May 1996

## ACKNOWLEDGMENTS

The National Computer Security Center extends special recognition to the authors of this document. The initial version was written by Barbara Blaustein, Sushil Jajodia, and Alfred Paradise of the MITRE Corporation. The final version was written by Gary Smith, Larry Halme, and David Wichers of Arca Systems, Inc.

The documents in this series were produced under the guidance of Shawn P. O'Brien of the National Security Agency, LouAnna Notargiacomo and Barbara Blaustein of the MITRE Corporation, and David Wichers of Arca Systems, Inc.

We wish to thank the members of the information security community who enthusiastically gave of their time and technical expertise in reviewing these documents and in providing valuable comments and suggestions.

## TABLE OF CONTENTS

## LIST OF FIGURES

# SECTION 1

# INTRODUCTION

This document is the third volume in the series of companion documents to the *Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria* [TDI 91; DoD 85]. This document examines polyinstantiation issues in multilevel secure (MLS) database management systems and summarizes the research to date in this area.

## 1.1 BACKGROUND AND PURPOSE

In 1991 the National Computer Security Center published the *Trusted Database Management System Interpretation* (TDI) of the *Trusted Computer System Evaluation Criteria* (TCSEC). The TDI, however, does not address many topics that are important to the design and development of secure database management systems (DBMSs). These topics (such as inference, aggregation, and database integrity) are being addressed by ongoing research and development. Since specific techniques in these topic areas had not yet gained broad acceptance, the topics were considered inappropriate for inclusion in the TDI.

The TDI is being supplemented by a series of companion documents to address these issues specific to secure DBMSs. Each companion document focuses on one topic by describing the problem, discussing the issues, and summarizing the research that has been done to date. The intent of the series is to make it clear to DBMS vendors, system designers, evaluators, and researchers what the issues are, the current approaches, their pros and cons, how they relate to a TCSEC/TDI evaluation, and what specific areas require additional research. Although some guidance may be presented, nothing contained within these documents should be interpreted as criteria.

These documents assume the reader understands basic DBMS concepts and relational database terminology. A background in security sufficient to use the TDI and TCSEC is also assumed; however, fundamentals are discussed whenever a common understanding is important to the discussion.

## 1.2 SCOPE

This document addresses polyinstantiation issues in multilevel secure DBMSs. It is the third of five volumes in the series of TDI companion documents, which includes the following documents:

- *Inference and Aggregation Issues in Secure Database Management Systems* [Inference 96]
- *Entity and Referential Integrity Issues in Multilevel Secure Database Management Systems* [Entity 96]
- *Polyinstantiation Issues in Multilevel Secure Database Management Systems*
- *Auditing Issues in Secure Database Management Systems* [Audit 96]
- *Discretionary Access Control Issues in High Assurance Secure Database Management Systems* [DAC 96]

This series of documents uses terminology from the relational model to provide a common basis for understanding the concepts presented. For most of the topics covered in this series, the concepts presented should apply to most database modeling paradigms, depending on the specifics of each model. This document specifically addresses relational DBMSs.

This document is related to the documents *Inference and Aggregation Issues in Secure Database Management Systems* [Inference 96] and *Entity and Referential Integrity Issues in Multilevel Secure Database Management Systems* [Entity 96]. Much of the discussion of the relationship between enforcement of integrity constraints and multilevel security centers on the inference channels which integrity constraints can introduce. In most cases, the enforcement of any integrity constraint referring to multilevel data will create a signaling channel [Meadows 88]. One way to avoid these channels caused by enforcing entity integrity is to use polyinstantiation.

## 1.3 INTRODUCTION TO POLYNISTANTIATION

The goal of mandatory security is to prevent the unauthorized disclosure of data by prohibiting users (or automated programs working on behalf of users) from accessing data for which they are not cleared. MLS DBMSs utilize mandatory access controls (MAC) based on the access class of the subject that is acting on behalf of (or in the name of) some named, accountable user. However, low-level users may be able to infer high-level information from only low-level data, real world knowledge of what the database is modeling, and knowledge of how the database functions. Besides preventing direct access to high data values, the TCB must also inhibit users from drawing inferences from the perceived existence (or absence) of high data, or from any noticeable effects from the manipulation of this data. This section introduces these concepts, which are examined in greater detail in later sections of this document.

Suppose that a multilevel database relation with element level labeling is constructed to summarize potentially sensitive flight mission information, as depicted in Figure 1.1. This relation's schema specifies a primary key of Flight ID and three attributes that describe the Origin, Destination, and Cargo of each flight. Associated with each element is a security label. Flight personnel on the ground, who are cleared to Secret, may need to be privy to Flight ID and Origin/Destination information. However, it may be desired that the nature of the flight manifest be at times hidden to all but Top Secret users. For example, Flight 007's cargo of alien spacecraft debris may be considered Top Secret. However, ground personnel may learn weight and size specifics, and may even be witness to the real-world loading process. The absence of data in the Cargo field would increase the likelihood of low users correctly inferring the presence of a Top Secret cargo with resulting natural curiosity as to its nature. It is possible that such an inference would threaten the success of the mission. Instead, a "cover story" could intentionally be created so that low users are placated by the plausible, but erroneous statement that the cargo consists of weather balloon debris. The question then becomes how to store and manage this conflicting information in the database.

| Flight ID | Origin | Destination | Cargo |
|---|---|---|---|
| . . . | | | |
| 007 S | Roswell S | Area 51 S | Weather Balloon Debris S |
| 007 S | Roswell S | Area 51 S | Alien Spacecraft Debris TS |
| . . . | | | |

**Figure 1.1: Introduction to Polyinstantiation**

A broader issue than cover stories that arises with a multilevel database is how the DBMS should address attempts by subjects to insert or change multilevel database objects. A low-level user may attempt to modify a tuple that, unbeknownst to him, already contains high data. If the database were to permit this, the low-level user would be able to change data of which that user is not even aware. This is likely to be undesirable. If instead, the database were to notify the user of the collision, the user would be able to infer that high data exists, and a downward signaling channel would result.[1]

In a like manner, if a high user attempts to insert data in a field which already contains low data, there are a similar set of inadequate approaches. The database can permit the update, but this would violate confidentiality with a direct write down from a high subject. The database can prohibit the subject from making the update, but it would be impractical to always disallow updates by high users. Finally, the database can perform the update, but automatically raise the classification of the data to the higher level of the update. However, since this would effectively make this field disappear from the low-level user's perspective, a downward signaling channel would again be created.

In order to avoid the drawbacks of these update approaches, an alternative is for the database to support multiple instances of records with the same primary key at different levels. Polyinstantiation is a solution that controls inference and signaling channels, and that also offers a way to implement cover stories. Polyinstantiation permits the creation of multiple instances of data records -- a low user sees the tuple associated with a particular primary key populated with one set of element values; a high user may see the “same” tuple with perhaps different values for some of the elements or multiple tuples of different levels with different data values. By supporting multiple instances, the database could allow a low subject to insert data and not change the data in the existing higher level record. Likewise, a high subject is permitted to insert data without disclosing it to low subjects

Polyinstantiation expands the notion of primary key to include the security label so that more than one tuple may possess the same “apparent primary key” if they are at different security levels. By its nature, polyinstantiation thus necessitates the violation of integrity across levels: Users at different security levels querying on a record with the same apparent primary key will not be

1. When automated means are used to query and update the database, signaling channels become a greater concern, both for speed of signaling and for possible hidden Trojan horse software. A user may rightfully be deemed trustworthy to access a database at multiple levels, but if the application software he uses contains hidden code which exploits signaling channels, confidentiality enforcement may be surreptitiously bypassed.

ensured of viewing the same data. The DBMS ensures database integrity only with respect to the data at a given level. Integrity across levels cannot be enforced if multilevel confidentiality is strictly enforced.

The impetus for supporting polyinstantiation is the attempt to reconcile conflicting requirements within an MLS DBMS -- the fundamental conflict between data integrity (which is not required by the TCSEC) and confidentiality (which is required by the TCSEC). Confidentiality, or secrecy, refers to protecting the data from unauthorized disclosure, while integrity refers to protecting the data from unauthorized alteration or destruction.

It is important to note that the TDI does not explicitly mandate that polyinstantiation be supported (or implemented). It only requires that confidentiality be enforced correctly, and that an evaluated DBMS address in some manner the threat to confidentiality that can arise from integrity constraints defined over data at more than one security level. Enforcing integrity constraints can open channels by which high-level information can be transmitted down to low-level users, either through some form of signaling, or through inference. Signaling channels can covertly transfer information through permitting either detectable changes to a low-level storage object to reflect high-level data (storage channel), or timing variations, detectable at the low-level which reflect the encoding of high-level data (timing channel). Guidelines on acceptable bandwidths for covert channels are given in Section 8 of the TCSEC. The potential speed of signaling makes attention to these channels important for even TCSEC Class B1 DBMS products to address as an unacceptable design flaw. Inference channels permit low-level users to infer high-level information from only low-level data and knowledge about the real world and how the DBMS works. Polyinstantiation is an approach to enforcing confidentiality while retaining some data integrity. In the wider database community, integrity refers more generally to the correctness, accuracy, and internal consistency of data. The issue becomes one of stretching the traditional non-secure data model, with its built-in data integrity safeguards, to accommodate the needs of the MLS environment.

The topic of polyinstantiation remains contentious because it introduces additional complexity into a database and it increases confusion as to what in a database reflects correctly modeled real-world values. For example, a polyinstantiated real-world entity may be modeled by multiple database tuples, and cases can easily arise where no single one of these tuples will contain all the correct element values to properly describe the real-world entity. Further, the relational theory of normalization is at odds with polyinstantiation enforcement: for example, a relation with polyinstantiated elements is not even in first normal form. This document reviews polyinstantiation research, examines these concerns, and summarizes different approaches to address the underlying database integrity-secrecy conflict.

## 1.4 AUDIENCES OF THIS DOCUMENT

This document is targeted at four primary audiences: the security research community, database application developers/system integrators, trusted product vendors, and product evaluators. In general, this document is intended to present a basis for understanding the necessity for supporting polyinstantiation or a suitable alternative in MLS DBMSs. Implemented approaches and ongoing research are examined. Members of the specific audiences should expect to get the following from this document:

Researcher

This document describes the basic issues associated with polyinstantiation. Important research contributions are discussed as various topics are examined. By presenting current theory and debate, this discussion will help the research community understand the scope of the issue and highlight polyinstantiation approaches and alternatives. For additional relevant work, the researcher should consult two associated TDI companion documents: *Inference and Aggregation Issues in Secure Database Management Systems* [Inference 96] and *Entity and Referential Integrity Issues in Multilevel Secure Database Management Systems* [Entity 96].

Database Application Developer/System Integrator

This document highlights the potential hazards and added complexity of management caused by polyinstantiation in MLS applications. It describes techniques to aid the application developer to minimize the occurrence of polyinstantiation and facilitate the efficient locating and cleanup of undesired polyinstantiation aftereffects. Intentional polyinstantiation in the form of cover stories is introduced, and the issues associated with cover story support are examined.

Trusted Product Vendor

This document describes the conflict between integrity and secrecy. It examines approaches to polyinstantiation enforcement in an MLS database and the benefits and drawbacks of these approaches. This is discussed in the context of tuple level as well as element level labeling. Approaches to polyinstantiation adopted by currently evaluated commercial MLS DBMS products are examined.

Evaluator

This document presents an understanding of polyinstantiation issues to facilitate evaluation of a candidate MLS DBMSs implementation of polyinstantiation or an alternative.

## 1.5 ORGANIZATION OF THIS DOCUMENT

The organization of the remainder of this document is as follows:

- Section 2 provides background by defining terminology and notation adopted by this document, and by introducing concepts basic to the discussion of polyinstantiation.
- Section 3 describes the issues associated with polyinstantiation in more detail and discusses the architectural considerations that affect polyinstantiation.
- Section 4 introduces an example and uses it to examine a number of different polyinstantiation and polyinstantiation-avoidance approaches.
- Section 5 presents the polyinstantiation practices of commercially available DBMS products.
- Section 6 summarizes the polyinstantiation issues which were presented in the document.

# SECTION 2

# BACKGROUND

This section provides background necessary for discussing polyinstantiation. Terminology and notation used by this report are introduced in Section 2.1. Section 2.2 compares polyinstantiation enforced at the granularity of a tuple versus at the granularity of an element. Section 2.3 compares the triggering of polyinstantiation due to an action on the part of a user at the high level versus by one at the low level.

## 2.1 TERMINOLOGY AND NOTATION

MLS DBMSs utilize MAC to prevent the unauthorized disclosure of high-level data to low-level users. In an MLS DBMS, it is necessary to hide the actions (inserts, deletes, updates) of high subjects from low subjects, and thereby prevent signaling channels that could disclose high-level data. It is also important to prevent low-level user actions from overwriting high-level data in the DBMS. Inference is another threat to MAC policy: low-level users may be able to infer high-level information from only low-level data, real world knowledge, and knowledge about how the DBMS works, such as what actions it normally allows or disallows. One method to reduce or eliminate some potential inference channels is to upgrade the classification of some key data element in the inferential chain and thus remove access to it by the low-level user [Inference 96]. However, other inferential chains to the high-level information may be available using an alternate path or too much information may have already been disseminated and thus no longer be explicitly represented in (controlled by) the database [Garvey 91]. Polyinstantiation (and support for cover stories) addresses these concerns.

The term *polyinstantiation* was coined by the Secure Data Views (SeaView) project and refers to the simultaneous existence within an MLS DBMS of multiple data objects with the same name, where the multiple instantiations are distinguished by their security level [Denning 87]. The *relational data model* is the formalism used in describing polyinstantiation. We assume readers possess basic familiarity with the relational data model [Date 81, 83]. To illustrate the notation we use throughout this paper, we give some brief definitions pertinent to MLS DBMSs and discussion of polyinstantiation approaches.

A standard relational database can be perceived by its users as a collection of relations. Relations are composed of tuples. A tuple ($v_1$, $v_2$, ..., $v_k$) has k components where the i-th component is $v_i$. A relation may be viewed as a table, with rows called tuples, and the columns called attributes. A relation has well-defined mathematical properties and consists of a relation scheme (which defines the attributes and name of the relation) and a relation instance. A *relation schema R* ($A_1$, ..., $A_n$) consists of a relation name *R* and a set of attribute names $A_1$, ..., $A_n$. An *instance* R of the schema *R* consists of a set of *tuples t*, each of which contains a single value *t*[$A_i$] for each attribute $A_i$.[2]

In an MLS environment, relations may consist of values at different classification levels. These

2. In the literature, the term *entity* is used more generally for tuple. Since all the approaches described in this document are based on the relational model, we use *tuple* throughout the document.

classification levels are ordered according to a *security lattice*. The reader should be familiar with classification levels and the standard models of secure information flow [Fernandez 81, Denning 82]. Our examples will refer to the levels Top Secret (TS), Secret (S), Confidential (C), and Unclassified (U) as applied to both data and to user sessions. For example, an "S-user" is a user performing actions on the database from a session operating at the Secret level.

A *multilevel relation schema* $R_c$ is an augmented version of a relation schema $R$ ($A_1$,...,$A_n$ in which security labels are associated with certain attributes, as constrained in the multilevel schema. These are maintained by the DBMS TCB for each tuple. In a system that requires all values in a single tuple to be uniformly classified, $R_c$ may simply have a security label, TC, which is associated with each tuple.[3] If values within a tuple may be classified at different levels, then $R_c$ includes security labels $C_1$,...,$C_n$, associated with each one of the attributes $A_1$,...,$A_n$, respectively, within a tuple. (If it is known that certain attributes will always be classified at the same level within a single tuple, fewer security labels may be stored.) The inclusion of security labels $C_1$,...,$C_n$ represents the most general case and is therefore used in this document.

The basic model for multilevel relations needs to be defined with a MAC policy in mind. The MAC policy for MLS databases is often based on the Bell-LaPadula model [Bell 76], which is stated in terms of subjects and objects. A subject is an active entity, such as a process that can request access to objects, whereas an object[4] is a passive entity, such as a record, a file, or a field within a record. Every subject is assigned a clearance level and every object a classification level. Classification levels and clearances are collectively referred to as *security levels*, and form a lattice. Each security level has two components: a hierarchical component and a set (possibly empty) of unordered categories. A security level $c_1$ is said to *dominate* security level $c_2$, in the induced partial order, if (1) the hierarchical component of $c_1$ is greater than or equal to that of $c_2$, and (2) all categories in $c_2$ are included in those of $c_1$. A security level $c_1$ *strictly dominates* security level $c_2$, in the partial order, if (1) $c_1$ dominates $c_2$, and (2) $c_1$ does not equal $c_2$.

The following are two necessary conditions in the Bell-LaPadula model [Bell 76, DoD 85]:

1. *The Simple Security Property or "No Read Up":* A subject can only read objects at a security level dominated by the subject's level, and

2. *The *-Property (Star Property) or "No Write Down":* A subject can only write objects at a security level that dominates the subject's level.

To apply these concepts to a DBMS, it is necessary to determine the granularity of the objects protected by MAC, i.e., the storage objects. Security levels are then associated with these storage objects. Work on MLS databases has focused on four choices for assigning security levels to data stored in a relation. One can assign security levels to entire relations, to individual tuples (rows) of

---

3. The Tuple Class and attribute security labels are sometimes loosely presented in the literature as attributes themselves. However, an attribute is normally considered to be part of the schema, so a user would presumably have direct access to it and its contents. If a label is a user accessible attribute, then protecting and preserving its semantic integrity (i.e., the integrity of its relationship to its associated attributes) becomes problematic.
4. "object", as used in Bell and LaPadula model is not the same as an object-oriented DBMS (OODBMS) where objects are active containers of information.

a relation, to individual attributes (columns) of a relation, or to individual elements of a relation. Much of the research on polyinstantiation has examined the case where security levels are assigned to individual data elements stored in relations. However, since tuple level labeling is used in all MLS DBMS products evaluated to date, the concept of polyinstantiation must also be considered in the (simplified) context of tuple level labeling. Polyinstantiation is not an issue when all values of a given attribute are uniformly classified as is the case with relation level or attribute level labeling [Hinke 75].

A *candidate key* of a relation schema is a minimal set of attributes that uniquely determine the other attributes. There may be a number of candidate keys for a relation schema *R*, i.e., there may be distinct sets of attributes that could be chosen to be a key for the relation. One candidate key is identified as the *primary key*. The notion of a primary key is a fundamental concept in the world of single-level relational databases. The primary key is used to facilitate storage and retrieval, and maintain the integrity of relations. Entity integrity requires that the primary key serve as a unique identifier of each tuple in the relation and that it does not contain a null value.

While the notion of a primary key is simple and well understood for classical (single-level) relations, it does not have a straightforward extension to multilevel relations. A primary key's uniqueness requirement can create signaling channels [Meadows 88]. One approach to avoiding these channels involves augmenting the user-defined primary key with security labels associated with the primary key attributes. The concept of an *apparent primary key* was introduced by Denning et al., to refer to the unaugmented user-defined primary key [Denning 87].

A multilevel relation is said to be *polyinstantiated* when it contains two or more tuples with the same apparent primary key values. Therefore, the *real primary key* (i.e., the minimal set of attributes unique in each tuple) of the multilevel relation is obtained by additionally considering the security labels associated with the attributes of the apparent primary key. The exact manner in which this is done is closely related to the precise polyinstantiation behavior of the relation [Cuppens 92]. The discussion of different approaches to resolving the polyinstantiation problem provided in Sections 4 and 5 includes descriptions of how real primary keys are defined.

## 2.2 TUPLE VERSUS ELEMENT POLYINSTANTIATION

Two types of polyinstantiation may be considered. Tuple polyinstantiation refers to polyinstantiation at the granularity of the data tuple (which normally represents a realworld entity). Element polyinstantiation refers to polyinstantiation at the granularity of a data element.[5]

A relation with tuple polyinstantiation contains multiple tuples with the same apparent primary key values, but with different access class values. As an example, consider the relation Starship-Objective-Destination (SOD) illustrated in Figure 2.1. The named “starship” is tasked with performing a particular mission objective at a specified destination in the galaxy. In this relation, it may be considered important to the success of a mission that a particular objective or destination not be accessible to users not cleared up to a certain security level. It may even be desired that the existence of the craft itself remain secret to uncleared users.

5. With the derived values approach discussed in Section 4.3, one can effectively achieve the flexibility of element polyinstantiation through polyinstantiation at the granularity of a tuple.

| Starship | | Objective | | Destination | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Talos | U | U |
| Enterprise | S | Spying | S | Rigel | S | S |

**Figure 2.1: A Multilevel Relation With Tuple Polyinstantiation**

Here we demonstrate the SOD relation using an element-level labeling database in which each element has a classification as well as a value. In an element-level labeling database, the classification of a tuple is computed to be the least upper bound of the classifications of the individual data elements in the tuple. This computed value is denoted in our examples within a bolded column to distinguish it from data which must be stored in the database. The attribute Starship is the apparent primary key of SOD. This example and many of those following are drawn from Jajodia et al. [Jajodia 94].

The relation given in Figure 2.1 contains two tuples for a starship that has the same name, resulting in tuple polyinstantiation. The apparent primary key values are identical in both tuples; however, these values have different classification levels. These tuples can be regarded as pertaining to two different real-world entities (e.g., if the existence of the secret Enterprise starship is unknown to a U-user who then independently chooses to christen an unclassified starship “Enterprise”). It can also be regarded as pertaining to two representations of a single real-world entity (e.g., an unclassified cover story for a secret mission). It is important to note that the relation itself does not indicate which of these two interpretations is true.

Figure 2.2 shows a relation with element polyinstantiation. A relation with element polyinstantiation contains two or more tuples with identical values both for the apparent primary key and the associated classification level element, but these tuples have different values for one or more other elements, as illustrated in Figure 2.2. Both tuples in this relation refer to the same starship Enterprise; however, the objectives and destinations of these tuples appear different to users at the Secret level.

| Starship | | Objective | | Destination | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Talos | U | U |
| Enterprise | S | Spying | S | Rigel | S | S |

**Figure 2.2: Multilevel Relation With Element Polyinstantiation**

As mentioned before, even when multilevel relations are labeled by tuple instead of element, explicit polyinstantiation is still possible. Consider the same relation as in Figure 2.1 with the tuple-level labeling that is offered by all currently evaluated commercial MLS DBMS products. The S-user will see the multilevel relation shown in Figure 2.3. In this case the TC column is not bolded. Because there are no security labels associated with individual elements, the classification of the tuple must be stored explicitly (or implicitly) in the DBMS.

| **Starship** | **Objective** | **Destination** | **TC** |
|---|---|---|---|
| Enterprise | Exploration | Talos | U |
| Enterprise | Spying | Rigel | S |

**Figure 2.3: Polyinstantiated Multilevel Relation With Tuple-Level Labeling**

Notice that when tuple-level labeling is used, there is no differentiation between tuple polyinstantiation and element polyinstantiation. In the example above, both tuples may possibly pertain to a single starship. In this case, the U-tuple may be a cover story which was purposely inserted to appease low user curiosity, or it may represent data about the starship which was entered by a low user (whether erroneously or from another Unclassified source such as a news feed) or it may have been left unchanged when an update was performed by a high user. On the other hand, it is also possible that the tuples refer to two different starships that happen to have the same name, although this situation may have arisen by error.

## 2.3 POLYHIGH VERSUS POLYLOW POLYINSTANTIATION

Tuple and element polyinstantiation can be triggered in two different ways: these are called *polyhigh* and *polylow* for mnemonic convenience.

1. Polyhigh occurs when a subject at a high level attempts to insert a tuple with the same apparent primary key as a low tuple or attempts to modify a low tuple. Although the high subject could be notified of this collision, the low-level data cannot be modified. Modifying the data and permitting access to it at the lower level would result in a direct write down in violation of the MAC policy. Modifying the data and raising its level to match the higher level of the subject would create a downward signaling channel, as the data would disappear at the low level. Polyhigh polyinstantiation leaves the old tuple unmodified, and instead inserts a new tuple which contains the new higher level data.

2. Polylow occurs when a subject at a low level attempts to insert a tuple with the same apparent primary key as a high tuple or modify an attribute of a tuple which already contains high data. In this case the low subject cannot be notified of the collision: if the update were rejected, there would be a downward signaling channel. Therefore, a new tuple is added with the lower level data.

The following examples show polyhigh and polylow using element polyinstantiation; tuple polyinstantiation is similar.

To illustrate polyhigh, suppose that a subject attempts to update the following relation:

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | null | U | U |

Suppose that the following sequence of updates occurs.

1. A U-user updates the destination of the Enterprise to be Talos. The result of this update is:

| **Starship** | **Objective** | **Destination** | **TC** |
|---|---|---|---|
| Enterprise U | Exploration U | Talos U | U |

2. Next, an S-user wishes to update the destination of the Enterprise to be Rigel. The system cannot reject this update without denying legitimate privileges to the user, but there would be a downward signaling channel if the U-level data were replaced by S-level data because U-users would see data disappearing, as a result of the data being upgraded. Therefore, a new tuple is added, making the relation polyinstantiated. U-users would see the following unchanged relation:

| **Starship** | **Objective** | **Destination** | **TC** |
|---|---|---|---|
| Enterprise U | Exploration U | Talos U | U |

S-users, however, could see the following modified relation:

| **Starship** | **Objective** | **Destination** | **TC** |
|---|---|---|---|
| Enterprise U | Exploration U | Talos U | U |
| Enterprise U | Spying U | Rigel S | S |

S-users may interpret this polyinstantiated relation in two ways: (1) the tuples represent a real-world truth, namely that the U-Level destination is a cover story for the S-level destination, or (2) that there is some error or inconsistency in the database that must be repaired.

To illustrate polylow, suppose that the two updates above occur in the opposite order. Starting from the same initial relation as above, suppose the updates occur in the following sequence.

1. An S-user updates the destination of the Enterprise to be Rigel. U-users see the following unchanged relation:

| **Starship** | **Objective** | **Destination** | **TC** |
|---|---|---|---|
| Enterprise U | Exploration U | null U | U |

S-users, however, see the following modified relation:

| **Starship** | **Objective** | **Destination** | **TC** |
|---|---|---|---|
| Enterprise U | Exploration U | Rigel S | S |

2. A U-user wishes to update the destination of the Enterprise to be Talos. The system cannot reject this update or even notify the U-user without causing a downward signaling channel, so the U-tuple is modified. U-users see the following modified relation:

| **Starship** | **Objective** | **Destination** | **TC** |
|---|---|---|---|
| Enterprise U | Exploration U | Talos U | U |

S-users see the following polyinstantiated relation, identical to the one at the end of the polyhigh example:

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Talos | U | U |
| Enterprise | U | Spying | U | Rigel | S | S |

In some applications, permitting the denial of service (in the case of polyhigh) or the overwriting of high data with low data (in the case of polylow), may be tolerable. Generally, however, polyinstantiation is required to enforce a multilevel security policy which prevents signaling channels as well as denial of service. Polyinstantiation may introduce complexity and even ambiguity into a database. These and other issues associated with polyinstantiation are examined in the following section.

# SECTION 3

# POLYINSTANTIATION CONSIDERATIONS

This section summarizes issues associated with polyinstantiation. Section 3.1 examines particular problems that polyinstantiation introduces. Section 3.2 presents a straightforward but flawed alternative to polyinstantiation. Section 3.3 examines the issue of enforcing automatic polyinstantiation versus user requested polyinstantiation. Limiting polyinstantiation to occur only when an authorized user explicitly requests it (i.e., to enforce cover stories) simplifies enforcement, eliminating many of the problems associated with polyinstantiation. Section 3.4 examines the architectural implications of supporting polyinstantiation.

## 3.1 PROBLEMS CAUSED BY POLYINSTANTIATION

This section discusses problems that enforcement of polyinstantiation presents to the database designer and maintainer, as well as the database application designer and database user. A core problem associated with polyinstantiation is the fundamental conflict between maintaining the secrecy of the information within a database and the integrity of the database. A database designer has added responsibility to see to the continued consistency of the database despite the need to present multiple views to its users. Enforcing secrecy through polyinstantiation results in the loss of real-world entity integrity and increases the complexity of the data relationships within the database. Due to this added complexity, as well as the unintentional ways in which polyinstantiation may be mistakenly invoked, increased administrative burden is also a problem.

### 3.1.1 Loss of Real-World Entity Integrity

Intuitively, a standard relational database represents a single non-contradictory view of the real world. Adding multilevel security and polyinstantiation support means that the relational database must now represent multiple views, potentially one or more for each classification level represented in the database. Whereas in a non-MLS database each tuple in a relation would normally represent a unique instance of a real-world entity, in an MLS DBMS this one-for-one relationship does not necessarily hold true and ambiguity becomes a problem.

To address secrecy-threatening signaling issues, low level users must be prevented from learning any information about higher level views. Polyinstantiation provides a means of hiding high views from low users. However, with polyinstantiation, realworld entity integrity may now only be ensured within a level and not across multiple levels. Further, a user of the database whose authorization spans more than one classification level will now be in the position of having to choose from multiple views. If the database presents views which contain contradictory data without associated explanation, the user may be faced with an inconsistent situation [Cholvy 94].

As described in Section 2.0, multiple tuples with the same apparent primary key could represent either (1) separate real-world entities modeled with the same key in the database, or (2) a single real-world entity modeled differently at different security levels. In non-MLS DBMSs, the first case violates the property that primary keys be unique. A non-MLS DBMS would simply prohibit the addition of a second tuple with the same key as an existing tuple. However, in an MLS environment, this prohibition may conflict with confidentiality requirements. If the existing tuple is at a higher security level than the proposed second tuple, prohibiting the addition of the second tuple will signal the existence of the higher level tuple. This consequence is at the core of the tuple polyinstantiation problem. Since the user inserting the new tuple cannot be informed of the existence of the higher level tuple, one cannot determine whether the user is entering data about the same real-world entity.

The relational theory of normalization has been studied as a way to address polyinstantiation and the relationship of integrity to secrecy [Qian 92, 94]. The notion of integrity is problematic for MLS DBMSs. Qian argues that if integrity constraints serve as a filter on how much low-level data can flow high then the case of a low datum contradicting a high datum would neither be apparent at a particular level (violating integrity) nor would either datum need to be disallowed (causing loss of information and introducing a signaling channel). Likewise, the notion of secrecy is problematic in the presence of integrity constraints. Again, it is argued that if update semantics are adopted that permit high-level users to gain access to low-level data only as long as integrity is preserved, then signaling channels are eliminated: Low updates are not denied due to high data, and high updates will not cause low data to change. Qian concludes that the detection of deductive inference channels is a special case of integrity protection: integrity implies the absence of deductive inference channels. Thus, according to Qian, the relationship between integrity and secrecy are not necessarily in fundamental conflict.

### 3.1.2 Increased Database Complexity to the User

Polyinstantiation can result from a number of scenarios:

1. A user intentionally introducing a cover story against data stored at a higher level.
2. Automatic polylow or polyhigh polyinstantiation correctly acting to counter the signaling threat which would result from announcing the existence of higher data when a user (or malicious code acting on a user's behalf) modifies or inserts data already stored at another level.
3. A user who, by failing to check whether the data being added is referenced already at a lower level, mistakenly modifies a relation at a level higher than necessary. (This is especially a concern associated with executing at the wrong level an automated program possessing only limited checks.)
4. A data regrade in progress in which a high user adds a tuple at the new level (low or high), but has not yet deleted the old tuple.

Given the wide range of reasons which can be the cause of a database demonstrating the effects of polyinstantiation, the increased complexity to interpret what is actually signified by what is stored in the database is obvious.

For example, consider the following scenario using a tuple level labeling DBMS. Given the following relation:

| **Starship** | **Objective** | **Destination** | **TC** |
|---|---|---|---|
| Defiant | Repair | Talos | S |
| Enterprise | Exploration | Talos | U |

An operator is provided the Secret information that the Enterprise should prepare for a spying mission. By mistake, the operator updates this information as a TS-user, creating the following relation:

| **Starship** | **Objective** | **Destination** | **TC** |
|---|---|---|---|
| Enterprise | Spy | Talos | TS |
| Defiant | Repair | Talos | S |
| Enterprise | Exploration | Talos | U |

Another operator is relayed the information that the destination of the Secret Enterprise mission is Rigel. Logging in as an S-user and making this straightforward addition creates the following table:

| **Starship** | **Objective** | **Destination** | **TC** |
|---|---|---|---|
| Enterprise | Spy | Talos | TS |
| Enterprise | Exploration | Rigel | S |
| Defiant | Repair | Talos | S |
| Enterprise | Exploration | Talos | U |

On a far outpost, a new supply ship is christened with the name Defiant and a U-user adds this information as the ship sets out for Talos, unaware this name is already taken by a secret military ship.

| **Starship** | **Objective** | **Destination** | **TC** |
|---|---|---|---|
| Enterprise | Spy | Talos | TS |
| Enterprise | Exploration | Rigel | S |
| Defiant | Repair | Talos | S |
| Defiant | null | Talos | U |
| Enterprise | Exploration | Talos | U |

Now suppose the Romulans attack Talos and in this crisis situation you are given the task of querying the database to determine which resources are near the front. How difficult will it be to determine what is reality vs. cover story vs. database entry error?

An operational polyinstantiating database may be subject to a number of inputs over its lifetime which can create similar ambiguous and/or spurious tuples. Mechanisms and enforcement of administrative procedures will be needed to periodically cleanup undesired polyinstantiation effects. The additional burden to database administration is discussed in the following section.

### 3.1.3 Increased Database Administration

Real-world use of databases enforcing polyinstantiation will demand additional duties of the database administrator, as well as greater effort on the part of the users. Users must be careful not to cause unwanted polyinstantiation. Inadvertent and unwanted polyinstantiation may result due to clumsy user error, incorrect use of functionality, or *-property anomalies which result from operations being performed from the wrong level by a user or automated function. Undesired polyinstantiation will create additional clean-up issues for the database administrator. All of these increased database administration issues are experienced with tuple-level labeling databases, and become ever more problematic for element-level labeling databases.

Depending upon the specific decomposition and materialization algorithms with which polyinstantiation is enforced, a database may also become cluttered with what may be considered spurious tuples. In the original element-level label SeaView model [Lunt 90], a single update operation could create a number of tuples equal to the product of the number of tuples in each of the affected domains.

For example, in the original SeaView, an S-user issuing an update to the following relation:

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Talos | U | U |

to state that the Enterprise's mission was actually spying at Rigel, would generate the following tuples:

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Talos | U | U |
| Enterprise | U | Exploration | U | Rigel | S | S |
| Enterprise | U | Spying | S | Talos | U | S |
| Enterprise | U | Spying | S | Rigel | S | S |

The polyinstantiation integrity property enforced by SeaView requires that the key and its classification determine each attribute and its classification via multivalued dependency (see Section 4.2). If the user updating the relation believes the second and third tuples are spurious, clean-up tasks are required to remove them. Further discussion of spurious tuples and techniques to minimize them is presented in Section 4.2's detailed discussion of decomposition and materialization algorithms.

The enforcement of polyinstantiation may also cause the database administrator to be subject to complaints about database performance and increased requests for database tuning to improve it. Depending upon the implementation and the number of instances across levels, the overhead associated with polyinstantiation may significantly slow query performance.

A final administrative issue is interoperation with resident and subsequently acquired database applications. Untrusted applications written for standard versions of DBMS products may be incompatible with polyinstantiation enforcement, and may behave unpredictably when polyinstantiation is encountered. A database administrator will have additional duties to ensure that introduced tools or utilities will not "break" the database's integrity, and that the tools will not break or produce erroneous results.

## 3.2 A SIMPLE, BUT UNACCEPTABLE, ALTERNATIVE TO POLYINSTANTIATION

A simple solution may be theorized that enforces "secure" alternatives to both visible and invisible polyinstantiations. These alternatives are secure in the sense of secrecy and information flow, and preserve primary key requirements in multilevel relations; but unfortunately, they suffer from denial of service and integrity problems. The following two rules describe this solution:

1. Whenever a high user makes an update which violates the uniqueness requirement, we simply refuse that update.

2. Whenever a low user makes a change that conflicts with the uniqueness requirement, the conflicting high data is overwritten in place by the low data.

It is not difficult to see that this simple alternative to polyinstantiation preserves the uniqueness requirement in multilevel relations. This solution is also secure in the sense of secrecy and information flow. Although this solution may be acceptable in some specific situations, it is clearly unacceptable as a general solution because it introduces serious denial-of-service and integrity problems.

## 3.3 AUTOMATIC VERSUS INTENTIONAL POLYINSTANTIATION

Two approaches to implementing polyinstantiation may be considered: instruct a system to automatically polyinstantiate whenever an action by a high or low user opens a potential inference channel, or polyinstantiate only when manually directed to by a high user and handle any remaining inference channel concerns through some other means. The more general approach of automatic polyinstantiation is the prevalent method which has been investigated, but restricting polyinstantiation to occur only upon a user's request eliminates many of the automatic polyinstantiation approach's shortcomings.

Intentional polyinstantiation refers to user-directed polyinstantiation, generally in support of creating cover stories that lead to alternative explanations and the forestalling of an inferential chain resulting in unauthorized disclosure of information [Garvey 91]. Cover stories must be designed to purposely mislead the lower user with a plausible explanation and prevent the inference of the classified value. To be effective, a cover story usually requires consistency.

Some researchers assert that cover stories are the only valid reason for the use of polyinstantiation [Burns 90, 91]. The complexity and confusion caused by automatic polyinstantiation, especially when likely blunders by unsophisticated users are acknowledged, supports this argument for implementing only intentional polyinstantiation. The ability for complexity arising from polyinstantiation to impact a modeled mission was illustrated in Section 3.1.2. These researchers would advocate alternatives to polyinstantiation where cover stories are not explicitly needed. It may be argued that polyinstantiation is not a fundamental property of multilevel databases. Rather, polyinstantiation simply provides a powerful technique for supporting cover stories. Other research has sought to demonstrate that polyinstantiation is not even essential for supporting cover stories, and may be considered a poor technique since it is difficult to prevent spurious cover stories from occurring [Wiseman 90].

The problem of maintaining global consistency remains in databases that restrict polyinstantiation to intentional polyinstantiation. High users who query the database and are returned high level data and low level cover stories without explanation are still faced with an inconsistent situation.

One approach to this complexity is to assume that the higher the level view of a database the more reliable it is, and that the other views should be considered cover stories [Cholvy 94]. In the case of databases with partial ordering, Cholvy and Cuppens suggest that *topics* be associated with the data. Topics allow representation of semantic links between data, and may be used to parameterize the order of the security levels and to merge data with this finer grain of preference. The database administrator would define an order of preference for merging information related to a specific topic.

For example, consider a database which includes Unclassified and Secret hierarchical classifications and two compartments: Destination and Cargo. The topics *dest* and *freight* are defined against the database. The order of preference specified for merging information related to these two topics may vary. For instance, the total order of preference defined for merging information related to the topic *dest* might be:

$$(S, \{Destination, Cargo\}) >_{dest} (S, Destination) >_{dest} (S, Cargo) >_{dest} U$$

while the total order of preference for the topic *freight* might be:

$$(S, (Destination, Cargo)) >_{freight} (Si\ Cargo) >_{freight} (Si\ Destination) >_{freight} U$$

The order of preference between these two topics differ because, according to the specific need to know of users at level (S, Cargo), information related to the topic *freight* is more reliable at level (S, Cargo) than at level (S, Destination) and the opposite is true for information related to the topic *dest.*

Another approach is to create two different attributes named Cargo and Freight which are classified at different global levels. These could be used as a means of controlling the two different views without ambiguity [Hinke 75].

## 3.4 ARCHITECTURAL CONSIDERATIONS IN SUPPORTING POLYINSTANTIATION

An MLS DBMS may be designed and built using a number of different architectures. The adopted architecture is closely tied to the range of polyinstantiation strategies available for a given system. Two primary architectures may be defined based on whether the database is trusted with respect to MAC [NAP 83]. In this section, we briefly discuss these two architectures in order to identify their influence on polyinstantiation strategies.

Figures 3.1 and 3.2 illustrate the two approaches. Figure 3.1 illustrates the No MAC Privileges (NMP) architecture[6]. Separate databases store data at each classification level. Each DBMS process can access all databases with data at or below its level.

A variation to the NMP approach is for each DBMS to contain data at a given level and replicated data from all lower databases. The SINTRA database system prototype has adopted this replicated distributed approach with MAC enforcement by a trusted frontend, physical separation as a protection measure, and no modification to untrusted backend DBMSs [Kang 94]. The SINTRA architecture boasts greater data retrieval performance because a user's view is materialized from a single database, rather than across multiple single level database fragments.

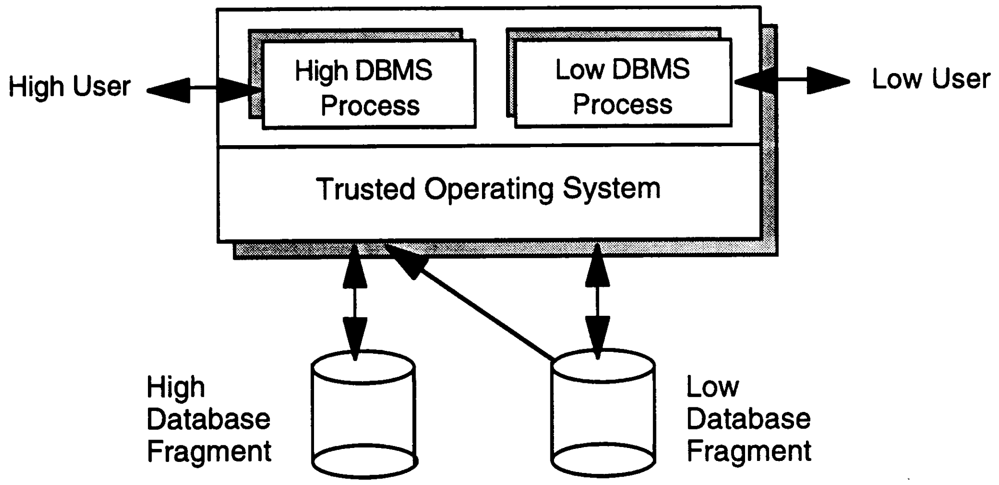

**Figure 3.1: No MAC Privileges Architecture**

Figure 3.2 illustrates the Trusted Subject architecture. A single database is used to store data at multiple levels, and the DBMS is trusted (i.e., has MAC privilege) to guard against illegal information flows.

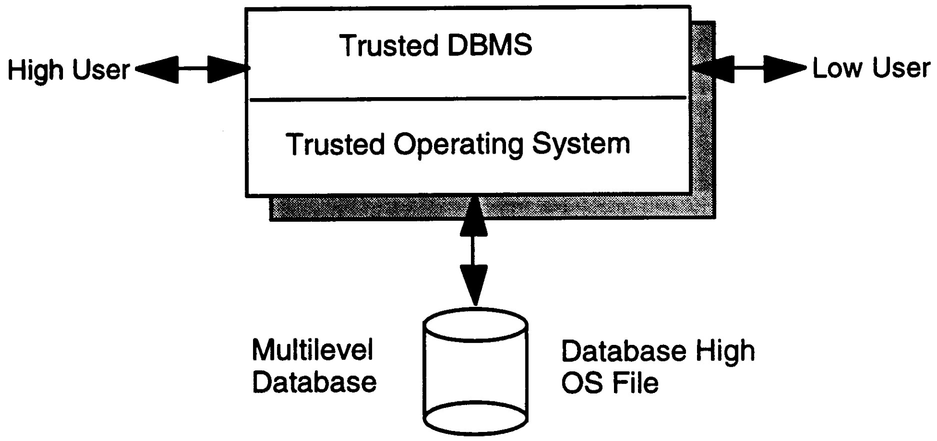

**Figure 3.2: Trusted Subject Architecture**

6. We use the term 'No MAC Privileges' to describe this architecture, which has been variously referred to as TCB Subsets [TDI 91], Kernelized Architecture [Air Force 83], and OS MAC Mode [Oracle 94]. These terms are all equivalent excepting that TCB Subsets is even more general, permitting the layering of any policies, rather than just DAC on top of MAC.

Polyinstantiation is a natural consequence of the NMP architecture. Since a lower-level DBMS has no knowledge of higher-level data, there is no way to prevent lower-level subjects from making updates that conflict with higher-level data. Requiring all keys to be classified at the lowest level protects against tuple polyinstantiation, but element polyinstantiation could still occur. Furthermore, this requirement may not be suitable for all applications. A specific approach to avoid this problem is discussed in Section 4.4.2. Element polyinstantiation may be allowed by defining logical relations that span multiple levels. The underlying databases would store single-level fragments of the relations. Restrictions on fragmentation are the first method to control the types of polyinstantiation semantics allowed within a system. If relations are fragmented, the fragments must be integrated to provide a coherent response to a query. Sections 4 and 5 discuss a number of approaches to integrating data.

The controls to prevent illegal information flow, which are built into the OS TCB in the NMP architecture, must be implemented through trusted software in the DBMS part of the TCB in the Trusted Subject architecture. For some applications it may be desirable to permit certain channels and avoid polyinstantiation. In general, this is easier to do in a Trusted Subject architecture where the DBMS portion of the TCB has access to all data levels, than in the NMP architecture where each DBMS instance can only observe data at levels it dominates.

## SECTION 4

## POLYINSTANTIATION APPROACHES

A number of different approaches can be used to implement polyinstantiation in a database management system reflecting divergent perspectives on the meaning and uses of polyinstantiation within an MLS environment. Each perspective has its strengths and its weaknesses, and the correct choice of approach depends on the requirements of specific applications. Different applications will be built to model different understandings of and requirements for multilevel data. Some approaches are only appropriate for databases enforcing element-level labeling, and as such may not be applicable to current commercial DBMS efforts which enforce tuple-level labeling.

This section introduces an example which will be used to examine various polyinstantiation approaches. Our discussion starts with approaches to MLS DBMS design that rely upon polyinstantiation and propagation of tuples to reflect different meaningful combinations of attribute values. Next, the section presents strategies that answer users' queries by using the security levels of retrieved tuples to derive new tuples. The last group of approaches places explicit restrictions on users' views of data, and includes approaches which permit complete avoidance of polyinstantiation. While this section presents a wide range of options, no claim is made that its exploration of polyinstantiation approaches is exhaustive. Likewise, alternative approaches toward the express goal of avoidance of polyinstantiation may be theorized, including careful auditing of channel usage, and constraints on the levels allowed in the DBMS.

### 4.1 AN EXAMPLE FOR COMPARING POLYINSTANTIATION APPROACHES

This section provides a detailed example that is used by the rest of Section 4 and Section 5 to compare and contrast approaches to polyinstantiation. Some of these approaches use polyinstantiation, while others add restrictions to eliminate the need for polyinstantiation. We use the same relation as previously, SOD, with the attributes Starship, Objective, and Destination. We assume for simplicity that the subjects and objects in our database may be described by simple hierarchical levels -- for example U and S[7]. Furthermore, we assume here that the Starship attribute is always Unclassified. Therefore, the classification range[8] of the Starship attribute has lower and upper bounds of U. Suppose, however, that both the Objective and Destination attributes have a classification range with a lower bound of U and an upper bound of S. Figure 4.1 shows the schema of this new relation, $SOD_C$.

The Tuple Class security label, abbreviated as TC, gives us the classification of the entire tuple. TC is a redundant (computed) security label whose value is the least upper bound of the classifications associated with the individual attributes in a tuple. The range of TC is derived in an obvious way from the classification ranges of the individual attributes.

7. This generalizes naturally to more complex lattices with the exception that in the case of incompatible levels the determination of which data to return to the user can be more complex. For example, the data returned to a user cleared to S and compartments A and B querying a database that has conflicting data at (S, A) and (S, B) would need to be reconciled. One approach to this issue is described in [Cholvy 94].
8. In some applications, an enumeration of applicable classification levels may be used instead of a continuous range. The discussion here remains the same.

Starship is the apparent primary key of SOD. Intuitively, this means that if $SOD_c$ contained only single-level data, then Starship would be the actual primary key of the relation. If SODc contains Unclassified and Secret data, however, the actual primary key of SODc would be Starship along with the classifications associated with the attributes. This primary key concept is central to the polyinstantiation problem and is formally stated in the next section.

| **Attribute** | **Classification Range** |
|---|---|
| Starship | [U, U] |
| Objective | [U, S] |
| Destination | [U, S] |
| Tuple Classification (TC) | [U, S] |

**Figure 4.1: Multilevel Scheme for the Relation $SOD_c$**

An instance of $SOD_c$ is likely to contain tuples at different levels. Therefore, it is important to distinguish between the U-instance of $SOD_c$, visible to Unclassified users, and the S-instance, visible to Secret users. Increasing a user’s clearance level should keep all previously visible information intact and perhaps add some new facts visible only at the higher level. For example, consider the U-instance of $SOD_c$ given in Figure 4.2. It contains exactly one tuple, meaning that, as far as Unclassified users are concerned, the starship Enterprise has set out to explore Talos.

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Talos | U | U |

**Figure 4.2: U-Instance of $SOD_c$**

Figure 4.3 enumerates eight different S-instances of $SOD_c$, all of which are consistent with the U-instance of Figure 4.2. Their common property is that the single tuple of the U-instance appears in all eight S-instances. Each tuple in an instance of $SOD_c$ defines a mission for the starship in question. A U-instance of $SOD_c$ allows only one mission per starship. S-instances, on the other hand, allow up to four missions (pairs of Objective and Destination) per starship, three of which are Secret and one Unclassified.

To gain further intuition into the polyinstantiation problem, consider instance 8 of Figure 4.3. This instance contains four tuples for the starship Enterprise. The classification associated with the Objective and Destination attributes makes each tuple distinct.

| No. | Starship | | Objective | | Destination | | TC |
|---|---|---|---|---|---|---|---|
| 1 | Enterprise | U | Exploration | U | Talos | U | U |
| 2 a | Enterprise | U | Exploration | U | Talos | U | U |
| b | Enterprise | U | Spying | S | Talos | U | S |
| 3 a | Enterprise | U | Exploration | U | Talos | U | U |
| b | Enterprise | U | Exploration | U | Rigel | S | S |
| 4 a | Enterprise | U | Exploration | U | Talos | U | U |
| b | Enterprise | U | Spying | S | Rigel | S | S |
| 5 a | Enterprise | U | Exploration | U | Talos | U | U |
| b | Enterprise | U | Exploration | U | Rigel | S | S |
| c | Enterprise | U | Spying | S | Rigel | S | S |
| 6 a | Enterprise | U | Exploration | U | Talos | U | U |
| b | Enterprise | U | Spying | S | Talos | U | S |
| c | Enterprise | U | Spying | S | Rigel | S | S |
| 7 a | Enterprise | U | Exploration | U | Talos | U | U |
| b | Enterprise | U | Spying | S | Talos | U | S |
| c | Enterprise | U | Exploration | U | Rigel | S | S |
| 8 a | Enterprise | U | Exploration | U | Talos | U | U |
| b | Enterprise | U | Spying | S | Talos | U | S |
| c | Enterprise | U | Exploration | U | Rigel | S | S |
| d | Enterprise | U | Spying | S | Rigel | S | S |

**Figure 4.3: Eight S-Instances of SOD$_c$**

The eight S-instances of SOD$_c$ can be partitioned into three classes as follows:

1. Instance 1 has no polyinstantiation.

2. Instances 2, 3, and 4 have a single U-tuple (a) and a single S-tuple (b) for the Enterprise. The U-tuple could be interpreted as a cover story for the correct information in the S-tuple. Instances 2, 3, and 4 show a cover story applied against different aspects of higher classified information. For example, instance 2 has a cover story for the objective but not the destination, while instance 3 has a cover story for the destination but not the objective.

3. Instances 5, 6, 7, and 8 are, however, confusing to interpret if it is assumed that higher level

data correctly represent the real world. Each of these cases has more than one S-tuple for the Enterprise, but only one U-tuple. Nonetheless, a meaningful and consistent interpretation and update semantics for such relations may be developed [Jajodia 90a, 90c].

## 4.2 PROPAGATION APPROACH

One way to implement polyinstantiation is termed the *propagation approach* The perspective that polyinstantiation is inherent in an MLS DBMS reflects the idea that, in the real world, people with different security clearances may see different information about the same entity. Similarly, MLS DBMS users at different levels may see different attribute values for the same real-world entity (e.g., an Unclassified cover story for a starship's destination), and the users' updates will reflect these different views. New tuples are added to reflect the different attribute values. The number of polyinstantiated tuples may be quite large under this approach to polyinstantiation.

The propagation approach must meet two requirements:

1. Ensuring that keys still function to uniquely identify tuples in the database, and
2. Controlling the propagation of tuples to include only meaningful combinations of attribute values.

The first requirement can be met by augmenting the apparent key with a security level and enforcing the standard key uniqueness property over this augmented key. The second requirement is more difficult to meet and researchers are still debating which types of combinations are meaningful. In general, *multivalued dependencies* (see [Date 83] for a more detailed explanation) are used to define the particular combinations allowed by a specific solution. While many variants are possible, the SeaView project [Denning 87, 88a, 88b; Lunt 89, 90, 91] and the proposed modifications of Jajodia and Sandhu [Jajodia 90b] provide the basis of this approach. First, we present the original SeaView approach, then Jajodia's and Sandhu's proposed modification, and finally some new techniques subsequently proposed by the SeaView project.

The goal of the SeaView project was to design an MLS relational database management system that satisfies the TCSEC for Class A1 [DoD 85]. It is claimed that the SeaView design and architecture can satisfy the A1 requirements simply by hosting it on an A1 operating system to which Trusted Oracle has been ported. This has not been done because no A1 OS is available [Hsieh 93, Lunt 94].

SeaView solves the problem of polyinstantiation of key attributes themselves by defining an *entity integrity* property. This property requires all attributes in a key to be *uniformly classified*. That is, for any instance $R_c$ of a multilevel relation schema $R_c$, for any tuple $t \in R_c$, and for any attributes $A_i$ and $A_j$ in the apparent primary key $K_R$ of $R$, $t[C_i] = t[C_j]$. Notice that this means it is possible simply to define a single security label $C_K$ to represent the classification level of all attributes in the apparent primary key. Further, no tuples may have null values for key attributes. This restriction ensures that keys can be meaningfully specified and checked for uniqueness. In addition, all non-key attribute classifications must dominate $C_K$. This restriction guarantees that if a user can see any part of a tuple, then he can see the key.

To meet the first challenge, that of using keys to determine when tuples model distinct real-world entities, SeaView defines a *polyinstantiation integrity* property, described below. The formulation of polyinstantiation integrity in SeaView consists of two distinct parts. The first part consists of a functional dependency[9] component whose effect is to prohibit polyinstantiation within the same access class. The second part consists of a multivalued dependency[10] requirement.

**SeaView Polyinstantiation Integrity Property:** A multilevel relation $R_c$ satisfies *polyinstantiation integrity* (PI) if and only if for every $R_c$ there are for all $A_i \not\in K_R$:

1. $K_R, C_K, C_i \rightarrow A_i$
2. $K_R, C_K \rightarrow\rightarrow A_i, C_i$

(The above single arrow is simple functional dependency; the double arrow represents multivalued functional dependency.) The PI property can be regarded as implicitly defining what is meant by the primary key in a multilevel relation. The primary key of a multilevel relation is $K_R \cup C_K$ u $C_R$ (where $C_R$ is the set of classification security labels for data attributes not in $_K$) since from PI it follows that the functional dependency $K_R \cup C_K \cup C_R \rightarrow A_R$ holds (where $A_R$ consists of all attributes that are not in $K_R$). For example, considering a U instance for the schema displayed in Figure 4.1, <the apparent primary key> (Starship) ∪ <classification level of all attributes in the apparent primary key> ‘U’ ∪ <set of classification security labels for data attributes not in the primary key> ‘S’ → (Objective, Destination).

The inclusion of the multivalued dependency in the definition of polyinstantiation integrity means that one update may result in a number of tuples being added to the relation. To illustrate, consider the situation in which an S-user attempts to go from S-instance 1 to S-instance 4 in Figure 4.3 by inserting the Secret tuple specifying the secret mission of spying on Rigel. SeaView will interpret this as a request to go from S-instance 1 to S-instance 8, thereby manufacturing two additional missions for the Enterprise. Unfortunately, this increases the potential for such additional information, that may not reflect true data, to be retrieved from the database by users with higher clearances.

In fact, of the eight instances defined in Figure 4.3, SeaView’s definition of polyinstantiation integrity allows *only two* combinations of these eight instances within a single relation scheme. Specifically, a SeaView relation can accommodate either instances 1, 2, 3, and 8 or instances 1 and 4 within a single scheme in the absence of the uniform classification constraint. SeaView admits only instances 1 and 4 if the Objective and Destination attributes are uniformly classified (i.e., either both are classified U or both S).

It is easy to see that, in the worst case, the number of manufactured tuples grows at the rate of $|$ security-latticel $|^k$, where k is the number of non-key attributes in the relation. For instance, Figure

9. A functional dependency between a set of attributes $X_1,...,X_n$ and an attribute Y (all of the same relation) means that each value of Y at any one time has associated with it precisely one combination of values for $X_1,...,X_n$. This dependency is denoted Xl,...,Xn → Y. See [Date 81] for a more detailed explanation.
10. A multivalued dependency between sets of attributes X1,...,Xn and Y1,...,Yn (all of the same relation) means that the set of values for Y1,...,Yn depends only on the values for attributes X1,...,Xn. See (Date 81] for a more detailed explanation.

4.4 shows a TS-instance of a relation similar to SOD, except that it has a range of four security levels for the Objective and Destination attributes. The particular TS-instance shown there describes four missions for the Enterprise, one each at the Unclassified, Confidential, Secret, and Top Secret levels.

| Starship | | Objective | | Destination | | TC |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Talos | U | U |
| Enterprise | U | Mining | U | Sirius | C | C |
| Enterprise | U | Spying | S | Rigel | S | S |
| Enterprise | U | Coup | TS | Orion | TS | TS |

**Figure 4.4: TS-Instance of SOD With Four Missions**

The definition of polyinstantiation integrity in SeaView requires that this information be represented by the sixteen missions shown in Figure 4.5. Users with clearances of U, C, S, and TS will respectively see 1, 4, 9, and 16 missions with the SeaView approach.

Jajodia et al., propose dropping the multivalued dependency from the polyinstantiation integrity property defined in the SeaView model [Jajodia 90a]. The authors argue that the multivalued dependency prohibits the existence of relation instances that are desirable in practice. Specifically, they argue that it is possible to accommodate all eight instances of Figure 4.3. Jajodia has developed formal operational semantics for these update operations on multilevel relations [Jajodia 91a, 91b, 91c].

| Starship | | Objective | | Destination | | TC |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Talos | U | U |
| Enterprise | U | Exploration | U | Sirius | C | C |
| Enterprise | U | Mining | C | Talos | U | C |
| Enterprise | U | Mining | C | Sirius | C | C |
| Enterprise | U | Exploration | U | Rigel | S | S |
| Enterprise | U | Mining | C | Rigel | S | S |
| Enterprise | U | Spying | S | Talos | U | S |
| Enterprise | U | Spying | S | Sirius | C | S |
| Enterprise | U | Spying | S | Rigel | S | S |
| Enterprise | U | Exploration | U | Orion | TS | TS |
| Enterprise | U | Mining | C | Orion | TS | TS |
| Enterprise | U | Spying | S | Orion | TS | TS |
| Enterprise | U | Coup | TS | Talos | U | TS |
| Enterprise | U | Coup | TS | Sirius | C | TS |
| Enterprise | U | Coup | TS | Rigel | S | TS |
| Enterprise | U | Coup | TS | Orion | TS | TS |

**Figure 4.5: Propagated Tuples of SOD$_c$**

Lunt and Hsieh of the SeaView team developed a semantics for the basic database manipulation operations (insert, update, and delete) [Lunt 91]. Based on these semantics, they propose a different definition for polyinstantiation integrity consisting of two separate pieces: (1) a state property containing the same functional dependency component, and (2) a transition property concerning a new dynamic multivalued dependency component. The latter property can be illustrated informally by an example from [Lunt 91].

Consider the multilevel relation scheme *SOD* (Starship, $C_{Starship}$, Objective, $C_{Objective}$, Destination, $C_{Destination}$, TC), composed of three attributes with associated element-level security labeling, and the computed tuple security label TC. The Starship attribute is the apparent primary key of *SOD*. An instance $SOD_c$ at a classification level $c$ is assumed to satisfy the two constraints of the PI property.

Now, consider the following relation instance $SOD_U$:

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Talos | U | U |

Suppose a Confidential user changes the value of Objective to "Mining," as shown here:

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Talos | U | U |
| Enterprise | U | Mining | C | Talos | U | C |

Under the update semantics of [Lunt 91], whenever an update involves some, but not all, of the non-key attributes, certain dynamic multivalued dependencies are enforced in the multilevel relations. In the example, the dynamic multivalued dependencies are:

$$\text{Starship } C_{Starship} \rightarrow\!\!\rightarrow (\text{Objective}, C_{Objective}) \mid (\text{Destination}, C_{Destination})$$

where the notation $X \rightarrow\!\!\rightarrow Y \mid Z$ denotes the multivalued dependencies $X \rightarrow\!\!\rightarrow Y$ and $X \rightarrow\!\!\rightarrow Z$.

Next, suppose a TS user updates the value of Destination to equal "Rigel." As before, since this update involves some (but not all) of the nonkey attributes, the dynamic multivalued dependency property causes two more tuples to be added to the relation:

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Talos | U | U |
| Enterprise | U | Mining | C | Talos | U | C |
| Enterprise | U | Exploration | U | Rigel | TS | TS |
| Enterprise | U | Mining | C | Rigel | TS | TS |

At this point, suppose a Secret user changes the value of the Objective to "Spying." The following relation instance will result:

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Talos | U | U |
| Enterprise | U | Mining | C | Talos | U | C |
| Enterprise | U | Exploration | U | Rigel | TS | TS |
| Enterprise | U | Mining | C | Rigel | TS | TS |
| Enterprise | U | Spying | S | Talos | U | S |
| Enterprise | U | Spying | S | Rigel | TS | TS |

As stated in [Lunt 91], the way in which an update occurs determines whether or not the multivalued dependency should be enforced. Essentially, if two or more attributes were updated in a single update statement, the multivalued dependency would not be enforced. However, if the two attributes were updated in two independent operations,[11] the multivalued dependency would be enforced. This dynamic approach has not been formalized, nor is it being incorporated into the SeaView prototype.

## 4.3 DERIVED VALUES APPROACH

A second perspective on polyinstantiation is that although a multilevel relation may have several tuples for the same real-world entity, there should be only one such tuple per classification level. Instead of a classification level $C_i$ associated with each attribute $A_i$, the schema $R_c$ includes a single classification level for each tuple, TC. When a user wants to update only certain attributes at a particular level, the values of the other attributes will be derived from values at lower security levels.

Consider the following relation SOD where Starship is the key:

| **Starship** | **Objective** | **Destination** | **TC** |
|---|---|---|---|
| Enterprise | Exploration | Talos | U |

Now suppose an S-user wishes to modify the destination of the Enterprise to be Rigel. He can simply do so by inserting a new Secret tuple to SOD as follows:

(Enterprise, û, Rigel, S)

The symbol û is to be interpreted as follows: For this S-tuple the value of the Objective field is identical to the corresponding U-tuple in SOD. As a consequence, when an S-user asks for the SOD relation to be materialized, he sees the following:

11. This case refers to low attributes being updated by high update operations.

| **Starship** | **Objective** | **Destination** | **TC** |
|---|---|---|---|
| Enterprise | Exploration | Talos | U |
| Enterprise | Exploration | Rigel | S |

The relation will appear unchanged to the U-user.

The Lock Data Views (LDV) project [Haigh 91] follows this *derived data approach.*

The derived data approach has been implemented for the United States Transportation Command Air Mobility Command MLS Global Decision Support System (GDSS) [Nelson 91]. The *MLS GDSS* implementation limits polyinstantiation in a multilevel relation to at most one tuple per security class. Information is labeled at one of two levels, U or S. The design is based on the organization's assumption that when S and U data are integrated into a single S response, the S data takes precedence over the U data. This design can be extended to environments with more than two strictly ordered security levels. Organizations for which this strict hierarchical rule does not apply, such as many compartmented environments, would need to incorporate substantial changes into this design in order to use it.

In the MLS GDSS application, trusted application software functionally extends the commercial off-the-shelf (COTS) MLS DBMS to manage tuple-level polyinstantiation. Before inserting an S-tuple, the trusted software ensures that a U-tuple exists with the same key. If it does not exist, the insertion of an S tuple is not permitted. If a U-tuple with the same apparent primary key does exist, the trusted application software examines each S-tuple attribute value, except the apparent key value, and determines if it replicates the attribute’s value in the U-tuple. If so, the value is not replicated in the S-tuple but instead is set to null, minimizing data replication. The U-tuple thus serves as the foundation upon which the S-tuple is built. The MLS GDSS solution is best explained with several examples. Consider the following relation:

| **Starship** | **Objective** | **Destination** | **TC** |
|---|---|---|---|
| Enterprise | Exploration | Talos | U |

Now suppose an S-user wishes to modify the destination of the Enterprise to Rigel. The S-user directs the system, through the trusted software, to insert an S-tuple into the SOD as follows:

S-USER:

```
Insert into SOD

        (Starship, Objective, Destination)

Values (‘Enterprise’, ‘Exploration’, ‘Rigel’);
```

The U and S tuples are now *stored* in the relation as follows:

| Starship | Objective | Destination | TC |
|---|---|---|---|
| Enterprise | Exploration | Talos | U |
| Enterprise | null | Rigel | S |

By reducing the replication of data across polyinstantiated tuples, the probability of maintaining the integrity of the database improves. Additionally, except for the key value, the sensitivity level of all attribute values contained within the *stored* tuple are equivalent to the TC value. Given this equivalence to the TC value, trusted application software derives attribute value labels from the TC value. Users operating at the U-level are presented with a display showing the derived attribute value labels as follows:

| Starship | Objective | Destination |
|---|---|---|
| Enterprise U | Exploration U | Talos U |

Users operating at the S-level are presented with a single composite display of a materialized tuple. This materialized tuple comprises S and U data as follows:

| Starship | Objective | Destination |
|---|---|---|
| Enterprise U | Exploration U | Rigel U |

One of the major impacts of this derived values polyinstantiation approach, as implemented in the MLS GDSS, involves the DBMS join operator at the S-level. Figure 4.6 illustrates the simplest form of this problem which needed to be addressed by GDSS. A typical join operation between two tables matches and retrieves rows based on the primary key Starship. In order to retrieve data residing at the same security level, and thus permit proper collapsing of the rows into a materialized tuple, the join is further qualified by the row's security label attribute TC:

S-USER:

```
Select * from Table 1, Table 2

        where Table 1.Starship = Table 2.Starship and Table 1.TC = Table 2.TC
```

An important functional requirement in MLS GDSS is that S-users expect to see S-data as the end product of a retrieval, if S-data exists; otherwise, U-data is returned. Case 1 in Figure 4.6 shows a join between two tables that produces the correct materialized tuple for an S user. Case 2 illustrates the anomaly associated with the join. In this case, Table 2 contains only U-data. Since the query requires that the tuple labels must match, the query does not return the S-row of Table 1 to be joined with the U-row of Table 2. Thus, if data does not exist at the same security levels in each table, then information may be lost during the join operation.

In this simplified example, one might argue that removing the qualification that the tables must be joined by tuple labels would permit joins. Doing this would return two rows in Case 2, one containing only U information, and the other containing S and U information. If this approach were taken, the tuple materialization process would become more complex and would need to extract multiple tuple labels and assign them to the appropriate columns in the row that was returned. Also, the join example shown in Case 1 would result in four rows of data returned from the server, instead of just two. The complexity of the problem and the work required of the DBMS server would increase significantly as more tables were joined. Database server performance would decrease accordingly, perhaps to unacceptable levels.

**Case 1:**

| **Starship** | **Objective** | **Destination** | **TC** |
|---|---|---|---|
| Enterprise | Exploration | Talos | U |
| Enterprise | null | Rigel | S |

Table 1

| **Starship** | **Type** | **Propulsion** | **TC** |
|---|---|---|---|
| Enterprise | Starship | Photon | U |
| Enterprise | Battlestar | Queller Drive | S |

Table 2

| **Starship** | **Objective** | **Destination** | **Type** | **Propulsion** |
|---|---|---|---|---|
| Enterprise U | Exploration U | Rigel S | Battlestar S | Queller Drive S |

Result of Join at S Level

**Case 2:**

| **Starship** | **Objective** | **Destination** | **TC** |
|---|---|---|---|
| Enterprise | Exploration | Talos | U |
| Enterprise | null | Rigel | S |

Table 1

| **Starship** | **Type** | **Propulsion** | **TC** |
|---|---|---|---|
| Enterprise | Starship | Photon | U |

Table 2

| **Starship** | **Objective** | **Destination** | **Type** | **Propulsion** |
|---|---|---|---|---|
| Enterprise U | Exploration U | Talos U | Starship U | Photon U |

Result of Join at S Level

**Figure 4.6: Joins in GDSS**

In order to ensure the correct materialization of a logical joined tuple, MLS GDSS does not currently use the join capabilities of the MLS COTS DBMS. Instead, tuples are selected from individual tables and then joined outside the DBMS by GDSS application software. While this operation results in some processing overhead, it ensures that data are not accidentally excluded from the S-user without unsupported modification to the COTS DBMS itself.

## 4.4 VISIBLE RESTRICTIONS APPROACH

The third perspective on polyinstantiation is that users are made aware that data are restricted to certain levels. In practice, this perspective means that users are cognizant of the levels of data that they can see and update. The goal is to provide a more “honest” database without compromising security. This perspective can lead to many different approaches; this section presents five different possibilities, including techniques to eliminate the need for polyinstantiation.

### 4.4.1 Belief Approach

One visible restrictions approach to polyinstantiation is motivated by the idea that data at each level should reflect the “beliefs” of users at that level about the real world [Smith 92]. For simplicity, we call this work the *belief approach*. In this approach, users *see* all the data that they could read per the Bell-LaPadula model, but *believe* the highest level data dominated by their operating level.

In this approach, updates reflect beliefs about the real world and they are regulated by the following property:

> **Update Access Property:** Data at a particular level can only be inserted, modified, or deleted by users at that level.

Thus, data at each level reflects the beliefs of the users who maintain it. Users may see the data that they believe as well as data believed by users at lower levels.

At the heart of this property is a model that takes a stand between tuple- and element-level polyinstantiation. Keys may be classified at a different level than other attributes within the same tuple, but all non-key attributes within a single tuple share a classification level.

Given a relation schema *R*, the multilevel relation $R_c$ used in the belief model includes two additional classification attributes: a key classification level ($K_c$) and a tuple classification level ($T_c$). The model imposes two restrictions:

1. In any tuple, $T_c$ must dominate $K_c$.
2. For the set of key attributes K and for all non-key attributes .$A_i,...,A_n$ in $R_c$,

$$K, K_c, T_c \rightarrow A_i,...,A_n$$

Intuitively then, tuples with the same values for key attributes but different key classification levels refer to different real-world entities. Tuples that are identical in key attributes and key classification levels but differ in tuple classification levels represent different beliefs about the same real-world entities. To maintain this distinction, users at a particular level are not allowed to reuse key attribute values for new entities.

Given the relation SOD in Figure 4.7, in the belief model, U-users believe the first and second tuples. C-users believe the third tuple, and S-users believe the fourth and fifth tuples. The second and third tuples refer to the same real-world starship, but U- and C-users have different beliefs about its objective and destination. The first and fifth tuples refer to different starships.

| **Starship** | $\mathbf{K_c}$ | **Objective** | **Destination** | $\mathbf{T_c}$ |
|---|---|---|---|---|
| Voyager | **U** | Shipping | Mars | U |
| Enterprise | **U** | Exploration | Vulcan | U |
| Enterprise | **U** | Diplomat | Romulus | C |
| Zardor | **S** | Warfare | Romulus | S |
| Voyager | **S** | Spying | Rigel | S |

**Figure 4.7: Example of SOD in the Belief Model**

U-users can see only the first two tuples in Figure 4.7, C-users can see the first three tuples, and S-users can see all five tuples.

Although users are allowed to see all tuples at levels dominated by their belief levels, the query language includes the optional keyword BELIEVED BY to allow users to restrict queries further. Thus, S-users can ask to see all allowable tuples, or only those believed by C- and S-users, or others.

So, the query "Display the destination of all starships named Enterprise" is expressed as:

```
SELECT          Destination
FROM            SOD
WHERE           Starship = "Enterprise"
BELIEVED BY     ANYONE
```

The result of this query, when issued against the relation in Figure 4.7, is:

| **Destination** | $\mathbf{T_c}$ |
|---|---|
| Vulcan | U |

for a U-user, and

| **Destination** | $\mathbf{T_c}$ |
|---|---|
| Vulcan | U |
| Romulus | C |

for all users at levels C or higher.

The query “Display the beliefs of U-users as to the destination of all starships named Enterprise” is expressed as:

```
SELECT          Destination
FROM            SOD
WHERE           Starship = “Enterprise”
BELIEVED BY     U
```

The result of this query, when issued against the relation in Figure 4.7, is:

| **Destination** | $\mathbf{T_c}$ |
|---|---|
| Vulcan | **U** |

for all users.

The query “Display the classification level and destination of all starships named Voyager” is expressed as:

```
SELECT          K_c, Destination
FROM            SOD
WHERE           Starship = “Voyager”
BELIEVED BY     ANYONE
```

The result of this query, when issued against the relation in Figure 4.7, is:

| $\mathbf{K_c}$ | **Destination** | $\mathbf{T_c}$ |
|---|---|---|
| U | Mars | U |

for U- and C-users, and

| $\mathbf{K_c}$ | **Destination** | $\mathbf{T_c}$ |
|---|---|---|
| U | Mars | U |
| S | Rigel | S |

for all users at levels S or higher.

### 4.4.2 Insert-Low Approach

Another variation of explicit restriction, the *insert-low approach,* has been adopted by the SWORD project at the Royal Signals and Radar Establishment in England [Wood 92]. In this approach, each relation is assigned a *table usage classification,* abbreviated as table class, at the time of its creation. Each attribute is assigned a *column classification* that must dominate the table class.

The purpose of the table class is two fold: First, any insertion or deletion of tuples in a relation can be made by users operating at the level of the table class of the relation. Second, the table class controls exactly how the updates involving an access class that dominates the table class can be made to the relation. This concept will be explained in greater detail below.

Consider once again the relation schema SOD. Say the table classification of SOD is U.

A typical instance of SOD is given as follows:

| **Starship** | **Objective** | **Destination** |
|---|---|---|
| Enterprise U | Exploration U | Talos U |
| Voyager U | Spying S | Rigel TS |

In this case, SWORD will show the entire relation to TS-users, while for those users at lower levels, SWORD will substitute <not cleared> whenever a user has insufficient clearance to view a value. Thus, for example, a C-user will see the following instance:

| **Starship** | **Objective** | **Destination** |
|---|---|---|
| Enterprise U | Exploration U | Talos U |
| Voyager U | <not cleared> | <not cleared> |

To see how SWORD avoids tuple polyinstantiation, consider once again the relation SOD with U as its table class. Suppose the initial database state is as follows:

| **Starship** | **Objective** | **Destination** |
|---|---|---|
| Enterprise U | Exploration U | Talos U |

Suppose some U-user inserts the tuple (Voyager, S, Spying, U, Talos, U) into SOD. SWORD allows lower level users to insert values at higher levels as long as the attribute value classifications are dominated by the appropriate column classification. In this example, the column classification for Starship would have to be S or higher. Furthermore, since the table classification of SOD is U, this constitutes a legal insertion, and as a result, U-users and S-users will see the following states, respectively:

| **Starship** | **Objective** | **Destination** |
|---|---|---|
| Enterprise U | Exploration U | Talos U |
| <not cleared> | Spying U | Talos U |

U-user

| **Starship** | **Objective** | **Destination** |
|---|---|---|
| Enterprise U | Exploration U | Talos U |
| Voyager S | Spying U | Talos U |

S-user

At this point, suppose a U-user wants to make an insertion (Freedom, U, Mining, U, Mars, U) to

SOD. Since the Starship attribute of tuples in SOD are not all visible to the U-user, there is always a possibility that the Starship value of the tuple to be inserted equals that of the existing high tuple, leading to element polyinstantiation (or tuple polyinstantiation in the case of attributes constituting the primary key). SWORD avoids this by prohibiting U-users from inserting or modifying values in this attribute. In the case of key attributes, like Starship, this means that all further insertions by U-users will be forbidden. However, since the table classification is U, only U-users can insert tuples into SOD. As a consequence, *no further insertions can be made to SOD at all.* In SWORD applications, then, the column classifications for all attributes constituting the primary key must equal the table class or users may be able to prohibit future insertions.

We next illustrate in more detail how element polyinstantiation is avoided in SWORD. Consider the SOD instance:

| **Starship** | | **Objective** | | **Destination** | |
|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Talos | U |

Next, suppose a TS-user wishes to modify the destination of the Enterprise to be Rigel. This is accomplished in two steps. First, the TS-user must log in as a U-user and change the classification of Talos from U to TS. Having done so, the TS-user can log in at his level and then make the desired update. As a result, the U instance and TS instance will become as follows:

| **Starship** | | **Objective** | | **Destination** | |
|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | <not cleared> | |

U-user

| **Starship** | | **Objective** | | **Destination** | |
|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Rigel | TS |

TS-user

Given the database state shown immediately above, suppose an S-user wants to insert a Secret destination for the Enterprise. He may do so by first logging in as a U-user, changing the classification of the attribute Destination from TS to S. As a result of this change, all users, *including* the TS-user, will see the following relation:

| **Starship** | | **Objective** | | **Destination** | |
|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | <not cleared> | S |

Now, the S-user can log in at classification level S and make the appropriate change.

### 4.4.3 Prevention Approach

A third approach to explicit restriction relies on preventative techniques to eliminate tuple polyinstantiation completely. Three basic techniques may be envisioned [Sandhu 91]:

1. **Make all the keys visible.** In this technique, the apparent primary key is required to be labeled at the lowest level at which a relation is visible. For example, suppose that the designer requires that all keys must be unclassified. Consequently, the following relation would be forbidden:

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Talos | U | U |
| Enterprise | S | Spying | S | Rigel | S | S |

Instead, note that the following two relations, called USOD and SSOD, represent the same information:

| **UStarship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Talos | U | U |

USOD

| **SStarship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | S | Spying | S | Rigel | S | S |

SSOD

In other words, USOD and SSOD horizontally partition the original SOD relation, with all the U-Starships in USOD and all the S-Starships in SSOD.

2. **Partition the domain of the primary key.** Another way to eliminate tuple polyinstantiation is to partition the domain of the primary key among the various access classes possible for the primary key. For our example, suppose that the application requires that starships whose names begin with A-E be Unclassified, starships whose names begin with F-T be Secret, and so on. Whenever a new tuple is inserted, the system enforces this requirement as an integrity constraint. In this case, the Secret Enterprise must be renamed, perhaps as follows:

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Talos | U | U |
| Voyager | S | Spying | S | Rigel | S | S |

The DBMS can now reject any attempt by a U-user to insert a starship whose name begins with F-Z, without causing any information leakage or integrity violation.

3. **Limit insertions to be done by trusted subjects.** A third way to eliminate tuple polyinstantiation is to require that all insertions must be done by a system-high user, with a write-down occurring as part of the insert operation. Strictly speaking, it is only necessary to have a relation-high user (i.e., a user to whom all tuples are visible). In the context of the example, this means that a U-user who wishes to insert the tuple (Enterprise,

Exploration, Talos) must request an S-user to do the insertion. The S-user will do so by invoking a trusted subject that can check for key conflict, and if there is none, insert a U-tuple by writing down. If there is a conflict the S-user informs the U-user about it, so the U-user can, for example, change the name of the starship.

The first technique is available in any DBMS that allows a range of access classes for individual attributes (or attribute groups) by simply limiting the classification range of the apparent key to be a singleton set. The second technique is available to any DBMS that can enforce domain constraints with adequate generality and trustworthiness. The third technique is always available but requires the use of trusted code, and tolerates some leakage of information (although with a human in the loop). The best technique will depend upon the characteristics of the DBMS and the application, particularly concerning the frequency and source of insertions.

In addition to the above, Sandhu's prevention approach also proposes techniques to prevent element polyinstantiation without compromising confidentiality, integrity, or denial-of-service requirements. The basic idea is to introduce a special symbol denoted by "restricted" as the possible value of a data element. The value "restricted" is distinct from any other value for that element and is also different from "null." In other words, the domain of a data element is its natural domain extended with "restricted" and "null." Sandhu then defines the semantics of "restricted" to be able to eliminate both polyhigh and polylow polyinstantiation [Sandhu 91].

Consider again the polyhigh scenario of Section 2.3. Begin with the following relation:

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Talos | U | U |

Next, suppose an S-user attempts to modify the destination of the Enterprise to be Rigel. This update does not cause any security violation, but if the new destination is classified Secret, additional steps are required to prevent even temporary polyinstantiation. The prevention approach requires an S-cleared user first to log in as a U-user[12] and to mark the destination of the Enterprise as "restricted," giving the following relation:

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | *restricted* | U | U |

The meaning of <restricted, U> is that this field can no longer be updated by an ordinary U-user.[13] U-users can therefore infer that the true value of the Enterprise's destination is classified at some level not dominated by U. The S-user then logs in as an S-subject and enters the destination of the Enterprise as Rigel, giving us the following relations at the U- and S-levels, respectively:

12. Alternately, the S-user logs in at the U-level and requests some properly authorized U-user to carry out this step. Communication of this request from the S-user to the U-user may also occur outside of the computer system, say by direct personal communication or a secure telephone call.

13. Only U-users selectively entrusted via non-discretionary means with the "unrestrict" privilege for this field can update it. Trojan horse software running at U is not able to modify a restricted value.

| Starship | | Objective | | Destination | | TC |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | *restricted* | U | U |

| Starship | | Objective | | Destination | | TC |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | *restricted* | U | U |
| Enterprise | U | Exploration | U | Rigel | S | S |

Note that this protocol does not introduce a signaling channel from an S-subject to a U-subject. There is an information flow, but from an S-user (logged in as a U-subject) to a U-subject. This is an important distinction. This type of information flow, which includes humans in the process, cannot be completely eliminated without cover stories. However, this protocol does prevent malicious software from signaling information without the knowledge of users.

Next, consider how the polylow scenario of Section 2.3 works with the restricted requirement. In this case, the Enterprise can have a Secret destination only if the destination has been marked as being “restricted” at the Unclassified level. Thus, if the S- and U-users, respectively, see the following instances of SOD:

| Starship | | Objective | | Destination | | TC |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | *restricted* | U | U |
| Enterprise | U | Exploration | U | Rigel | S | S |

| Starship | | Objective | | Destination | | TC |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | *restricted* | U | U |

then an attempt by a U-user to update the destination of the Enterprise to Talos will be

| Starship | | Objective | | Destination | | TC |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | null | U | U |

rejected. Alternatively, if both S- and U-users see the following instance:

then the U-user update will be allowed (without causing polyinstantiation).

The concept of the “restricted” mark is straightforward, as long as the classification lattice is totally ordered. In the general case of a partially ordered lattice, some subtleties arise. How to completely eliminate polyinstantiation using “restricted” is discussed at length in [Sandhu 91]. In general, updating the value of an element to “restricted” cannot cause polyinstantiation. On the other hand, updating the value of an element to a data value, say, at the C-level, can be the cause of polyinstantiation. If polyinstantiation is to be completely prohibited, this update must require that the data element be restricted at all levels that do not dominate C. The fact that the data element is restricted at all levels below C can be verified by the usual integrity-checking mechanisms in a DBMS [Sandhu 90b]. However, it is difficult to guarantee this at levels incomparable with C. In preparing to enter a data value at the C level, the system would need to start a system-low (actually, a data-element-low) process that can then write up. A protocol for this purpose is described in [Sandhu 91].[14]

### 4.4.4 Explicit Alternatives Approach

A fourth approach to explicit restriction allows the application developer to choose among explicit alternatives for polyinstantiation. Sandhu and Jajodia brought together a number of their previously published ideas, along with some new ones, to define a particular semantics for polyinstantiation called *polyinstantiation for cover stories* (PCS) [Sandhu 92]. PCS allows the developer to specify whether an attribute (or attribute group) of a multilevel tuple will support: (1) no polyinstantiation or (2) deliberate polyhigh polyinstantiation at the explicit request of a user to whom the polyinstantiation is visible. PCS strictly limits the extent of polyinstantiation by requiring that each real-world entity be modeled in a multilevel relation by at most one tuple per security class. The goal of PCS is to provide a natural, intuitive, and useful technique for implementing cover stories, with run-time flexibility regarding the use of cover stories. A particular attribute may be used for cover stories for some tuples and not for others. Even for the same real-world entity, a particular attribute may be polyinstantiated at some time and not at other times.

PCS combines the “one tuple per tuple class” concept with the “restricted” concept of Section 4.4.3. The basic motivation for PCS can be appreciated by considering the following instance of SOD:

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | *restricted* | U | Talos | U | U |
| Enterprise | U | Spying | S | Rigel | S | S |

In this case, the Destination attribute of the Enterprise is polyinstantiated, so that <Talos, U> is a cover story for the real S destination of Rigel. The Objective is not polyinstantiated.

Consider the occurrence of polyinstantiation due to polylow, as discussed by the example in Section 2.3. This example begins with S- and U-users, respectively, having the following views of SOD:

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Rigel | S | S |

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | null | U | U |

14. It should be noted that this protocol works for any arbitrary lattice and does not require any trusted subjects. The use of trusted subjects will allow simpler protocols for this purpose.

So far there is no polyinstantiation. Polyinstantiation occurs in the example when a U-user updates the destination of the Enterprise to be Talos.

PCS takes a slightly different approach to this example. According to the PCS approach, polyinstantiation *does* exist in the S-instance of SOD given above. PCS shows this instance as:

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | null | U | U |
| Enterprise | U | Exploration | U | Rigel | S | S |

In this approach, polyinstantiation already exists prior to the U-user updating the destination of the Enterprise to be Talos. This update merely modifies an already polyinstantiated relation instance to be:

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Talos | U | U |
| Enterprise | U | Exploration | U | Rigel | S | S |

With this approach, *element polyinstantiation can occur only due to polyhigh*. Polylow simply cannot be the cause of element polyinstantiation. Consequently, polyinstantiation will occur only by the deliberate action of a user to whom the polyinstantiation is immediately available. In other words, element polyinstantiation does not occur as a surprise.

The PCS approach treats null values like any other data value (except in the apparent key fields where null should not occur). Previous work on the semantics of null in polyinstantiated databases has taken the view that nulls are subsumed by non-null values independent of the access class [Jajodia 90b, Sandhu 90a]. In this case, the first tuple in the following relation available to S-users:

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | null | U | U |
| Enterprise | U | Exploration | U | Rigel | S | S |

is subsumed by the second tuple, resulting in the following relation for S-users used in the polylow example of Section 2.3:

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Rigel | S | S |

Under the explicit alternatives approach, the former relation is completely acceptable. The latter can be acceptable, but only if the lower limit on the classification of the destination attribute is S.

To further illustrate the semantics of null in PCS, consider the following relation:

| Starship | | Objective | | Destination | | TC |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | null | U | U |
| Enterprise | U | Exploration | U | null | U | S |

PCS considers this to be a polyinstantiated relation. The fact that there are nulls rather than data values in the polyinstantiated field has no bearing on the treatment of this relation. In contrast, the semantics of null in [Jajodia 90b] and [Sandhu 90a] require all null values to be classified at the level of the apparent key (U in this case), thereby deeming the second tuple illegal.

The PCS approach leaves many of the choices of whether or not to polyinstantiate to the discretion of application designer. It differentiates between updates that can and cannot cause polyinstantiation by using two different keywords (UPDATE and PUPDATE) to make the distinction explicit. The PCS approach also relies on the distinguished data value “restricted.” The meaning of this data value is that users at a classification level which returns the value “restricted” for an attribute, may not modify the value of that attribute. As in the prevention approach (Section 4.4.3), PCS includes special privileges for imposing and lifting such restrictions.

### 4.4.5 Multilevel Relational Data Model Approach

A fifth approach described here is an extension of the work just described in Section 4.4.4. The main benefits of this model, the Multilevel Relational (MLR) data model [Chen 95], over previous models is that it retains the capability for upward information flow while eliminating ambiguity of references by foreign keys.

This model, which supports attribute level labeling, requires that there be at most one tuple in each access class for a given entity. It requires that if a tuple includes attributes which are classified lower than the tuple’s access class, then a tuple at this lower level must exist with the same attribute value. Because of this requirement, and the fact that the minimum classification of an element may be higher than that of the primary key, it allows classification attributes to be NULL for attributes which have NULL values. The model also requires that a tuple exist at a given level in order to recognize that tuple at that level (i.e., there is no automatic recognition of the validity of less sensitive data elements). The main ideas behind the model are as follows:

1. The data accepted by a subject is divided into two parts: all data defined at the subject's level and any data explicitly “borrowed” from lower levels. Allowing data-borrow ensures that the MLR data model retains upward information flow because changes to lower level data can be automatically propagated to higher levels.

2. A subject can see data which is accepted by subjects at its level or at the levels below it.

3. A tuple contains all the data accepted (either owned or borrowed) by subjects at the level of the tuple. If a tuple at a given level does not exist, then subjects at that level do not accept the existence of that tuple.

The MLR data model is more precisely defined by the following five integrity properties as follows:

**Entity Integrity:**

Entity integrity (which comes from SeaView) protects the integrity of the apparent primary key (AK). It basically has three requirements which must be met by each tuple:

1. No attributes in AK have a null value.
2. The attributes in AK are uniformly classified.
3. All non-key attributes must dominate the classification of AK.

**Polyinstantiation Integrity:**

This includes two requirements:

1. There can only be one tuple at a given classification level for each unique AK.
2. For each unique AK there can only be one attribute value at each attribute level (i.e., you cannot have multiple values for the same attribute at the same attribute level for a given AK).

**Data Borrowed Integrity:**

This property, which is new to this model, ensures that borrowed data actually exists and changes to lower level data can be automatically propagated to higher levels. It basically states that:

1. In all tuples, for each non-null attribute that is less sensitive than the tuple classification, there must exist a tuple at the level of the attribute with the same attribute value and attribute classification (i.e., if you borrow less sensitive data, that data must exist).

**Foreign Key Integrity:**

This property, which also came from SeaView, basically states that:

1. For each foreign key, all attributes of that key are either null or non-null.
2. For each foreign key, all attributes of the key are uniformly classified.

**Referential Integrity:**

Referential integrity in the standard database model ensures that each foreign key references an existing primary key [Entity 96]. In this model, referential integrity states that:

1. For each foreign key, there must be a matching AK.
2. The classification of the foreign key must dominate the classification of the AK.
3. The classification of the tuple containing the foreign key must equal the classification of the tuple containing the AK.

This not only ensures that matching primary keys exist and are visible to the referencing tuple, but also requires that the two tuples be classified at the same level. This last requirement means that for any level *c*, *c*-tuples can only reference *c*-tuples. This follows from the restriction that only tuples at a given level are accepted at that level.

The main benefit of this new restriction is that there is no ambiguity in any references by a foreign key. Without this restriction, a tuple's foreign key could match the primary key of multiple polyinstantiated tuples that are dominated by the tuple with the foreign key. This requirement eliminates this potential ambiguity.

The model is illustrated in the following examples:

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Talos | U | U |
| Enterprise | U | Exploration | U | Rigel | S | S |

Using our familiar example, the tuples above would be perfectly legal in the MLR data because each tuple has a different access class and there is a U tuple that has the same Starship and Objective attribute values as the S tuple (i.e., Enterprise and Exploration). The following would not be legal for two reasons: one because it contains two tuples with the same apparent primary key that have the same access class (i.e., S) and two because there is no matching Mining Objective attribute in the U tuple (i.e., two attributes at the same level for the same primary key have conflicting values).

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | Exploration | U | Talos | U | U |
| Enterprise | U | Mining | U | Rigel | S | S |
| Enterprise | U | Spying | S | Rigel | S | S |

The following tuple illustrates the use of the "null" classification value capability. In this case, the objective is null. This feature is provided because attributes may have a minimum classification higher than the minimum classification of the primary key. In this example, the minimum classification for Objective could be Confidential. Without allowing null attribute and label values, there would be no way of creating a tuple at U which contains the Enterprise and Talos attribute values at U.

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | null | null | Talos | U | U |

Given this initial tuple, the following tuples could build upon this base as follows:

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | null | null | Talos | U | U |
| Enterprise | U | Mining | S | Rigel | S | S |
| Enterprise | U | Spying | TS | Rigel | S | TS |

These tuples indicate that the actual purpose of the Enterprise mission is to perform Spying on Rigel while cover stories of going to Talos or doing Mining are provided to less cleared individuals. If the Rigel destination was changed by an S user, then it would automatically propagate up to the TS tuple since it is being borrowed by that tuple. However, in the following tuple, a change to the Rigel destination at the S level would not be propagated, since (although) the destinations appear to be the same, that value was not borrowed from the lower level tuple.

| **Starship** | | **Objective** | | **Destination** | | **TC** |
|---|---|---|---|---|---|---|
| Enterprise | U | null | null | Talos | U | U |
| Enterprise | U | Mining | S | Rigel | S | S |
| Enterprise | U | Spying | TS | Rigel | TS | TS |

This model includes the addition of a new SQL statement which is used to indicate borrowing from lower level tuples, the UPLEVEL statement. This statement is used to indicate which attributes to borrow from a lower level tuple. This statement can be used to modify an existing tuple or create a new one.

In summary, this model unifies a number of other models and then extends them in such a way as to retain upward information flow without permitting downward flows, while eliminating semantic ambiguity of foreign keys due to polyinstantiated elements within a reasonably flexible attribute level labeling scheme.

## SECTION 5

## CURRENT COMMERCIAL APPROACHES TO POLYINSTANTIATION

Previous sections of this document discussed various issues surrounding polyinstantiation in MLS DBMSs. This section summarizes the polyinstantiation approaches provided by three commercially available MLS DBMSs: INFORMIX, Sybase, and Oracle; and the Trusted Rubix DBMS commercial prototype.

Each of these DBMS products provides tuple level labeling, so element level labeling, and the polyinstantiation issues associated with it, are not applicable.

**INFORMIX:**

The key for a tuple in INFORMIX OnLine/Secure automatically includes the tuple security label. Thus, polyinstantiation is always possible and cannot be suppressed by the DBMS. INFORMIX OnLine/Secure places no special requirements on suppressing polyhigh or polylow polyinstantiation. It does not provide any tools for cleaning up polyinstantiation. Any housecleaning of polyinstantiation effects will require a manual procedure or custom software. [INFORMIX 92]

**Sybase:**

The tuple sensitivity in the Sybase Secure SQL Server is automatically part of all keys. Thus polyinstantiation is always possible and cannot be suppressed by the DBMS. Sybase Secure SQL Server does not provide any specific tools for cleaning up polyinstantiation. Any cleanup requires a manual procedure or custom software. [Sybase 93]

**Oracle:**

Trusted Oracle can be configured to run in one of two modes. When run in DBMS MAC mode, a single Trusted Oracle database can store information at multiple labels. In this mode, Trusted Oracle can turn polyinstantiation on and off at the table level by requiring key integrity which does not include the tuple label. When on, the primary key includes the tuple label, which allows poLyinstantiation to occur. When off, the key does not include the tuple label, thus preventing polyinstantiation. The ramifications of having polyinstantiation turned off is that high users cannot insert tuples above lower level tuples that already exist with the same primary key and vice versa, which causes denial of service (to some degree) for high users as well as a covert channel for low users.

Including or not including the tuple label in the primary key is specified at table definition time. The tuple label can be added to the primary key at some future date, which would then permit polyinstantiation. However, once the tuple label is included in the primary key, it cannot be removed. [Oracle 93]

When run in OS MAC mode, Trusted Oracle is capable of storing data at only a single label, and the DBMS is constrained by the underlying OS MAC policy. Without any MAC privilege, the

DBMS cannot suppress polyinstantiation because a low DBMS will not be aware of any tuples with the same primary key at a higher level, and a high DBMS cannot be trusted to modify low data. As such, polyinstantiation cannot be prevented when Trusted Oracle is running in OS MAC mode.

**Trusted Rubix:**

Trusted Rubix provides the most flexible polyinstantiation mechanism of the products surveyed by its support of three polyinstantiation modes: POLYHIGH, POLYLOW, and what is referred to as POLYNONE. The selected mode is defined on a per table basis, when each table is created. Once a table had been created and the polyinstantiation discipline declared, it is not possible to change that discipline for the remainder of the table's lifetime. [Rubix 93]

POLYNONE

When this mode is selected for a table, polyinstantiation is prevented in that table, thus introducing a covert channel for low users and potentially preventing some inserts by high users.

POLYLOW

This mode causes HIGH keys to be replicated at LOW when a LOW subject attempts to insert an item whose key matches that of an extant HIGH item. This is done to avoid the covert channel problem. However, if a HIGH user attempts to insert a tuple which contains a key that already exists at a lower level, that insert is prevented.

POLYHIGH

This mode causes LOW keys to be replicated at HIGH when a HIGH subject attempts to insert an item whose key matches that of an extant LOW item. This is done to allow a high user to insert HIGH tuples when a LOW version already exists, rather than preventing duplication of keys at HIGH. This mode includes the characteristics offered by the POLYLOW mode since, in the Trusted Rubix implementation, it is not possible to support POLYHIGH behavior without supporting POLYLOW. This mode basically makes polyinstantiation of the Trusted Rubix table behave like a table which has had polyinstantiation turned on under Trusted Oracle DBMS MAC mode.

## SECTION 6

## SUMMARY

The design of an MLS DBMS must take into account the problem of polyinstantiation. When data items exist at multiple classification levels, the potential exists for inconsistent values for the same data item at different levels. Polyinstantiation may occur over tuples or elements, and it may arise through updates at low or high classification levels. Researchers have developed a number of different approaches to polyinstantiation; no one solution is best for all applications. This document has outlined approaches to resolving the polyinstantiation problem:

1. Propagate polyinstantiated tuples to reflect valid combinations of values (Section 4.2). Users at different levels may see different attribute values for the same realworld entity.

2. Identify derived tuples based on underlying polyinstantiated tuples (Section 4.3). Although a multilevel relation may have several tuples for the same real-world entity, there will only be one such tuple per classification level.

3. Indicate restrictions or inconsistencies present in the data so that polyinstantiation can be controlled (Section 4.4). Users are made aware that data are restricted to certain levels, and are cognizant of the levels of data that they can see and update.

There are no explicitly stated TCSEC requirements for automatic or intentional polyinstantiation mechanisms. Evaluators will require that a vendor of an MLS DBMS address in some manner the signaling channels which polyinstantiation prevents. Current commercial MLS DBMSs enforce tuple level polyinstantiation. With appropriate (possibly complex/inconvenient) database design, tuple level labeling can offer functionality comparable to research prototypes supporting element level labeling and polyinstantiation.

## REFERENCES

[Air Force 83] Air Force Studies Board, *Multilevel Data Management Security,* National Research Council, National Academy Press, Washington, DC, 1983.

[Audit 96] National Computer Security Center, *Auditing Issues in Secure Database Management Systems,* NCSC Technical Report-005, Volume 4/5, May 1996.

[Bell 76] Bell, D. E., LaPadula, L. J., *Secure Computer Systems: Unified Exposition and Multics Interpretation,* The MITRE Corporation, March 1976.

[Burns 90] Burns, R. K., "Integrity and Secrecy: Fundamental Conflicts in the Database Environment," *Proceedings of the Third RADC Database Security Workshop,* Castille, NY. 1990.

[Burns 91] Burns, R. K., "Polyinstantiation -- A Position Statement," *Fourth Workshop on the Foundations of Computer Security,* June 1991.

[Chen 95] Chen, F., Sandhu, R., "The Semantics and Expressive Power of the MLR Data Model," *1995 IEEE Conference on Security and Privacy,* Oakland, CA, May 1995.

[Cholvy 94] Cholvy, L., Cuppens, F., "Providing Consistent Views in a Polyinstantiated Database" *Eighth Annual Working Conference on Database Security,* August 1994.

[Cuppens 92] Cuppens, F., Yazdanian, K., "A 'Natural' Decomposition of Multi-level Relations," *Proceedings of the IEEE Symposium on Security and Privacy,* May 1992, pp. 273-284.

[DAC 96] National Computer Security Center, *Discretionary Access Control Issues in High Assurance Secure Database Management Systems,* NCSC Technical Report-005, Volume 5/5, May 1996.

[Date 81] Date, C. J., *An Introduction to Database Systems,* Third Edition, Addison-Wesley, Reading, MA, 1981.

[Date 83] Date, C. J., *An Introduction to Database Systems,* Volume II, Addison-Wesley, Reading, MA, 1983.

[Denning 82] Denning, D. E., *Cryptography and Data Security,* Addison-Wesley, Reading, MA, 1982.

[Denning 87] Denning, D. E., Lunt, T. F., Schell, R. R., Heckman, M., Shockley, W. R., "A Multilevel Relational Data Model," *Proceedings of the IEEE Symposium on Security and Privacy,* April 1987, pp. 220-234.

[Denning 88a] Denning, D. E., Lunt, T. F., Schell, R. R., Heckman, M., Shockley, W. R., "The SeaView Security Model," *Proceedings of the IEEE Symposium on Security and Privacy,* April 1988, pp. 218-233.

[Denning 88b] Denning, D. E., “Lessons Learned from Modeling a Secure Multilevel Relational Database System,” *Database Security: Status and Prospects* (C. E. Landwehr, editor), North-Holland, 1988, pp. 35-43.

[DoD 85] Department of Defense, *Department of Defense Trusted Computer System Evaluation Criteria,* DoD 5200.28-STD, Washington, DC. December 1985.

[Entity 96] National Computer Security Center, *Entity and Referential Integrity Issues in Multilevel Secure Database Management Systems,* NCSC Technical Report-005, Volume 2/5, May 1996.

[Fernandez 81] Fernandez, E. B., Summers, R. C., Wood, C., *Database Security and Integrity*, Addison-Wesley, 1981.

[Garvey 91] Garvey T., Lunt, T., “Cover Stories for Database Security,” *Proceedings of the 5th IFIP WG11.3 Workshop on Database Security,* W. Va., November 1991.

[Haigh 91] Haigh, J. T., O’Brien, R. C.,Thomsen, D. J., “The LDV Secure Relational DBMS Model,” *Database Security IV: Status and Prospects* (S. Jajodia and C. E. Landwehr, editors), North-Holland, 1991, pp. 265-279.

[Hinke 75] Hinke, T. H., Schaefer, M., *Secure Data Management System,* Technical Report RADC-TR-75-266, System Development Corporation, 1975.

[Hsieh 93] Hsieh, D., Lunt, T., Boucher, P., *The Sea View Prototype.* RL-TR-93-216 Final Technical Report, SRI International. November 1993.

[Inference 96] National Computer Security Center, *Inference and Aggregation Issues in Secure Database Management Systems,* NCSC Technical Report-005, Volume 1/5, May 1996.

[INFORMIX 92] *INFORMIK OnLine/Secure Security Features User’s Guide, Database Server Version 4.1,* INFORMIX Software, Inc., Menlo Park, CA, January 1992.

[Jajodia 90a] Jajodia, S., Kogan, B., “Transaction Processing in Multilevel-Secure Databases Using Replicated Architecture,” *Proceedings of the IEEE Symposium on Security and Privacy,* Oakland, California, May 1990, pp. 360-368.

[Jajodia 90b] Jajodia, S., Sandhu, R. S., “Polyinstantiation Integrity in Multilevel relations,” *Proceedings of the IEEE Symposium on Security and Privacy,* Oakland, California, May 1990, pp. 104-115.

[Jajodia 90c] Jajodia, S., Sandhu, R. S., Sibley, E., “Update Semantics of Multilevel Relations,” *Proceedings of the 6th Annual Computer Security Applications Conference*, 1990, pp. 103-112.

[Jajodia 91a] Jajodia, S., Sandhu, R. S., “Polyinstantiation Integrity in Multilevel Relations Revisited,” *Database Security IV: Status and Prospects* (S. Jajodia and C. E. Landwehr, editors), North-Holland, 1991, pp. 297-307.

[Jajodia 91b] Jajodia, S., Sandhu, R. S., “A Novel Decomposition of Multilevel Relations Into Single-Level Relations,” *Proceedings of the IEEE Symposium on Security and Privacy,* Oakland, California, May 1991, pp. 300-313.

[Jajodia 91c] Jajodia, S., Sandhu, R. S., “Toward a Multilevel Secure Relational Data Model", *Proceedings of the ACM SIGMOD International Conference on Management of Data,* Denver, Colorado, 29-31 May 1991, pp. 50-59.

[Jajodia 94] Jajodia, S., Sandhu, R. S., Blaustein, B., “Solutions to the Polyinstantiation Problem,” *Information Security: An Integrated Collection of Essays,* M. Abrams et al., eds., IEEE Computer Society Press, 1994.

[Kang 94] Kang, M. H., Froscher, J. N., McDermott, J., Costich, O., Peyton, R., “Achieving Database Security Through Data Replication: The SINTRA Prototype,” *Proceedings of the 17th National Computer Security Conference,* Baltimore, MD, October 1994.

[Lunt 89] Lunt, T. F. et al., *Secure Distributed Data Views,* Volume 1-4, SRI Project 1143, SRI International, 1988-89.

[Lunt 90] Lunt, T. F., Denning, D. E., Schell, R. R., Heckman, M., Shockley, W. R., “The SeaView Security Model,” *IEEE Transactions on Software Engineering,* Vol. 16, No. 6, June 1990, pp. 593-607.

[Lunt 91] Lunt, T. F., Hsieh, D., “Update Semantics for a Multilevel Relational Database,” *Database Security IV: Status and Prospects* (S. Jajodia and C. E. Landwehr, editors), North-Holland, 1991, pp. 281-296.

[Lunt 94] Lunt, T. F., Boucher, P. K., “The SeaView Prototype: Project Summary,” *Proceedings of the 17th National Computer Security Conference,* Baltimore, MD, October 1994.

[Meadows 88] Meadows, C., Jajodia, S., “Integrity Versus Security in Multi-Level Secure Databases,” *Database Security: Status and Prospects,* ed. C. Landwehr, North Holland, 1988.

[NAP 83] National Academy Press, *Multilevel Data Management Security,* National Academy Press, Washington, DC, 1983, (FOR OFFICIAL USE ONLY).

[Nelson 91] Nelson, D., Paradise, A., “Using Polyinstantiation to Develop an MLS Application,” *Proceedings of the Seventh Annual Computer Security Applications Conference*,, 1991, pp. 12-22.

[Oracle 93] *Trusted Oracle7 Server Administration Guide,* Version 7.0, January 1993, Oracle Corporation, Redwood City, CA.

[Oracle 94] *Oracle7 and Trusted Oracle7 Final Evaluation Report,* CSC-EPL-94/004, C-Evaluation Report No. 07-95, Library No. S242,198, National Computer Security Center, 5 April 1994.

[Qian 92] Qian, X., “Integrity, Secrecy, and Inference Channels,” *Proceedings of the 5th Rome Lab Database Symposium,* 1992.

[Qian 94] Qian, X., “Inference Channel-Free Integrity Constraints in Multilevel Relational Databases,” *Proceedings of the IEEE Symposium on Security and Privacy,* Oakland, California, 1994, pp. 158-167.

[Rubix 93] *Trusted Rubix Trusted Facility Manual,* Version 1.8, August 1993, Infosystems Technology, Inc., Greenbelt, MD.

[Sandhu 90a] Sandhu, R. S., Jajodia, S., Lunt, T. F., “A New Polyinstantiation Integrity Constraint for Multilevel Relations,” *Proceedings of the IEEE Workshop on Computer Security Foundations,* Franconia, New Hampshire, June 1990, pp. 159-165.

[Sandhu 90b] Sandhu, R. S., Jajodia, S., “Integrity Mechanisms in Database Management Systems,” *Proceedings of the 13th NIST-NCSC National Computer Security Conference,* Washington, DC, October 1990, pp. 526-540.

[Sandhu 91] Sandhu, R. S.,Jajodia, S., “Honest Databases That Can Keep Secrets,” *Proceedings of the 14th NIST-NCSC National Computer Security Conference,* Washington, D.C., October 1991, pp. 267-282.

[Sandhu 92] Sandhu, R. S., Jajodia, S., “Polyinstantiation for Cover Stories,” *Proceedings of the European Symposium on Research in Computer Security,* Toulouse, France, November 1992.

[Smith 92] Smith, K. P., Winslett, M. S., “Entity Modeling in the MLS Relational Model,” *Proceedings of the Very Large Data Base Conference,* Vancouver, Canada, 1992.

[Sybase 93] *Building Applications for Secure SQL Server,* Sybase Secure SQL Server Release 10.0., Sybase, Inc., Emeryville, CA, 3 September 1993.

[TDI 91] *Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria,* NCSC-TG-021, National Computer Security Center, April 1991.

[Wiseman 90] Wiseman, S. R., “On the Problem of Security in Data Bases,” *Database Security III: Status and Prospects* (D. L. Spooner and C. E. Landwehr, editors), North-Holland, 1990, pp. 143-150.

[Wood 92] Wood, A. W., Lewis, S. R., Wiseman, S. R., *The SWORD Multilevel Secure DBMS*, Technical Report No. 92005, Defense Research Agency, Malvern, Worcestershire, England, 1992.

| REPORT DOCUMENTATION PAGE | *Form Approved OMB No. 0704-0188* |
|---|---|

| | | |
|---|---|---|
| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE May 1996 | 3. REPORT TYPE AND DATES COVERED Final |

| | |
|---|---|
| 4. TITLE AND SUBTITLE Polyinstantiation Issues in Multilevel Secure Database Management Systems | 5. FUNDING NUMBERS |
| 6. AUTHOR(S) | |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) National Security Agency Attn: V21, Partnerships and Processes Fort George G. Meade, MD 20755-6000 | 8. PERFORMING ORGANIZATION REPORT NUMBER NCSC Technical Report - 005 Volume 3/5 Library No. S-243,039 |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSORING/MONITORING AGENCY REPORT NUMBER |

11. SUPPLEMENTARY NOTES

| | |
|---|---|
| 12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for Public Release Distribution Unlimited | 12b. DISTRIBUTION CODE |

13. ABSTRACT *(Maximum 200 words)*

This report is the third of five companion documents to the *Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria*. The companion documents address topics that are important to the design and development of secure database management systems, and are written for database vendors, system designers, evaluators, and researchers. This report addresses polyinstantiation issues in multilevel secure database management systems.

| | |
|---|---|
| 14. SUBJECT TERMS Polyinstantiation; Multilevel Secure Database Management Systems | 15. NUMBER OF PAGES 61 |
| | 16. PRICE CODE |

| | | | |
|---|---|---|---|
| 17. SECURITY CLASSIFCATION OF REPORT Unclassified | 18. SECURITY CLASSIIFICATION OF THIS PAGE Unclassified | 19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified | 20. LIMITATION OF ABSTRACT |