Proc. of the 14th NIST-NCSC National Computer Security Conference, Washington D.C., October 1991, pages 267-282

HONEST DATABASES THAT CAN KEEP SECRETS

*Ravi S. Sandhu and Sushil Jajodia**

Center for Secure Information Systems
and
Department of Information and Software Systems Engineering
George Mason University, Fairfax, VA 22030-4444

ABSTRACT Polyinstantiation has generated a great deal of controversy lately. Some have argued that polyinstantiation and integrity are fundamentally incompatible, and have proposed alternatives to polyinstantiation. Others have argued about the correct definition of polyinstantiation and its operational semantics. In this paper we provide a fresh analysis of the basic problem that we are trying to solve, i.e., how can a honest database keep secrets? Our analysis leads us to the concept of restricted polyinstantiation wherein we show how to solve this problem without compromising on any of the following requirements: secrecy, integrity, availability-of-service, element-level labeling and high assurance. This is the first solution to meet all these requirements simultaneously.

1 INTRODUCTION

What distinguishes a multilevel database from ordinary single level ones? In a multilevel world as we raise a user's clearance new facts emerge; conversely as we lower a user's clearance some facts get hidden. Therefore users with different clearances see different versions of reality. Moreover, these different versions must be kept coherent and consistent—both individually and relative to each other—without introducing any downward signaling channels.[†]

The caveat of "no downward signaling channels" poses a major new problem in building multilevel secure database management systems (DBMSs) as compared to ordinary single-level DBMSs. This caveat is inescapable and absolute. We must reject outright "solutions" which tolerate downward signaling channels. Solutions with such channels, e.g., as proposed in [1, 9], may well be acceptable as an engineering compromise in particular situations. But they are clearly not acceptable as general-purpose solutions. This point needs to be emphasized because security is usually the one to take the first hit in engineering trade-offs. It behooves us as security researchers to present solutions which avoid taking this hit while at the same time providing

- no downward signaling channels,
- consistency and integrity of the database both within and across levels,
- flexibility for application semantics,

*The work of both authors was partially supported by the U.S. Air Force, Rome Air Development Center through subcontract #C/UB-49;D.O.No.0042 of prime contract #F-30602-88-D-0026, Task B-O-3610 with CALSPAN-UB Research Center.

[†]We deliberately use the term downward signaling channel rather than covert channel. A downward signaling channel is a means of downward information flow which is inherent in the data model and will therefore occur in *every* implementation of the model. A covert channel on the other hand is a property of a specific implementation and not a property of the data model. In other words, even if the data model is free of downward signaling channels, a specific implementation may well contain covert channels due to implementation quirks.

© Ravi S. Sandhu and Sushil Jajodia, 1991

- fine-grained classification of data (i.e. element-level labeling), and
- high assurance with minimal trusted code.

The central point of this paper is to demonstrate how these diverse goals can be met in a multilevel relational DBMS without compromising security as part of the bargain. Our solution is simple in concept and almost obvious in retrospect. For the most part it uses standard concepts from the database arena. A key new idea is to introduce a special value called "restricted" distinct from the normal data values of an attribute (or column) as well as distinct from "null." The value "restricted" denotes that the particular field cannot be updated at the specified level. So long as the value of a field is not "restricted" our multilevel relations behave much as ordinary single-level relations do. Particular attention is required when a field is changed from unrestricted to restricted and vice versa. A notable property of our solution is that it can be implemented entirely by untrusted subjects, i.e., subjects which are not exempted from the simple security or $\star$-properties.[‡]

The rest of this paper is organized as follows. Section 2 reviews the concept of polyinstantiation from an intuitive point of view, with the objective of identifying the sources of polyinstantiation and alternatives to it. Section 3 informally introduces our solution of restricted polyinstantiation and illustrates it by examples. Section 4 formalizes and precisely defines our solution. It also provides additional examples. Section 5 discusses how our solution can provide the highest degree of assurance. Section 6 concludes the paper.

2 POLYINSTANTIATION

The concept of polyinstantiation was explicitly introduced by Denning et al [3] in connection with the SeaView project. Since then much has been written about this topic [1, 3, 4, 5, 6, 7, 9, for instance]. In this paper we will set aside all this previous theory, formalism and debate. Instead we go back to first principles and consider by means of examples how polyinstantiation arises and therefore how it might be controlled. We assume the reader is familiar with basic relational notions and terminology.

2.1 The Source of Polyinstantiation

Polyinstantiation can occur in basically two different ways which we call *polyhigh* and *polylow* respectively for mnemonic convenience.

1. Polyhigh occurs when a high user[§] attempts to insert data in a field which already contains low data. Overwriting the low data in place will result in a downward signaling channel. Therefore the high data can be inserted only by creating a new instance of the field to store the high data. We also have the option of rejecting the update altogether with the attendant possibility of denial-of-service to the high user.
2. Polylow occurs in the opposite situation where a low user attempts to insert data in a field which already contains high data. In this case rejecting the update is not a viable option because it establishes a downward signaling channel. That leaves us two alternatives. We can overwrite the high data in place which violates the integrity of the high data. Or we can create a new instance of the field to store the low data.

[‡]The protocols of section 4 can be simplified if trusted subjects which are exempted from these properties are allowed in selected situations.

[§]Strictly speaking we should be saying subject rather than user. For the most part we will loosely use these terms interchangeably. Where the distinction is important we will be appropriately precise.

In both cases note that we have identified "secure" alternatives to polyinstantiation. These alternatives are secure in the sense of secrecy and information flow. Unfortunately the alternatives have denial-of-service and integrity problems reiterated below.

1. The alternative to polyhigh entails denial-of-service to high users by low users (i.e., once a low value has been entered in a field a high value cannot be entered until the low value has been nullified by a low subject[¶]).
2. The alternative to polylow entails destruction of high data by low users which presents a serious integrity problem (i.e., the high data is overwritten in place by low data.)

A naive implementation of these alternatives will create more real security problems than it solves. Our main contribution in this paper is to show how these alternatives to polyhigh and polylow can be employed in a careful, disciplined manner to achieve secrecy, availability-of-service and integrity with high assurance.

It should be noted that there is an important difference between polyhigh and polylow. Polyhigh can be completely prevented by reactive mechanisms at the cost of denial-of-service to entry of high data. This is likely to be a tolerable cost in many applications. On the other hand polylow cannot be completely prevented by reactive mechanisms. At the moment of enforcement a reactive mechanism has only the alternative of overwriting high data by low data. This is likely to be intolerable in most applications. Therefore polylow must—for all practical purposes—be prevented by a proactive mechanism, i.e., steps must be taken in advance of the problem's occurrence to ensure that it cannot occur.

2.2 Polyhigh Example

Let us now consider a concrete example to make polyhigh and polylow clearer. Consider the following relation SOD where Starship is the apparent primary key.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	null	U	U

Here, as in all our examples, each attribute in a tuple not only has a value but also a classification. In addition there is a tuple-class or TC attribute. This attribute is computed to be the least upper bound of the classifications of the individual data elements in the tuple.

Now consider the following scenario.

1. A U user updates the destination of the Enterprise to be Talos. The relation is therefore modified as follows.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U

2. Next a S user attempts to modify the destination of the Enterprise to be Rigel. We cannot overwrite the destination in place because that would create a downward signaling channel. We can reject the update at the risk of denying entry of legitimate secret data. Or we can polyinstantiate and modify the relation to appear as follows, respectively for U and S users. Note that U users see no change.

[¶] This protocol—of nullifying low data prior to entry of high data—does not guarantee protection against denial-of-service. If a low value is nullified to enable entry of a high value there remains the risk that a low Trojan Horse can enter another low data value before the high subject has the opportunity to enter its high value. The solution described in this paper (see Section 3) eliminates this vulnerability.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U
Enterprise	U	Exploration	U	Rigel	S	S

What are we to make of this last relation given above. There are at least two reasonable interpretations.

- *Cover Story.* The destination of Talos may be a cover story for the real destination of Rigel. In this case the database is accurately mimicking the duplicity of the real world. There are, however, other ways of incorporating cover stories besides polyinstantiation. For example we may have two attributes, one for cover-story destination and one for the real destination. Debate on the relative merits and demerits of these techniques is outside the scope of this paper. *For purpose of this paper we assume that polyinstantiation is not to be used for cover stories. We therefore reject this alternative as a valid interpretation.*

- *Temporary Inconsistency.* We have a temporary inconsistency in the database which needs to be resolved. For instance the inconsistency may be resolved as follows: the S user who inserted the Rigel destination latter logs in at the U level and nullifies the Talos value, so thereafter the relation appears respectively as follows to U and S users.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	null	U	U

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Rigel	S	S

  It is most important to understand that this scheme does not create a downward signaling channel from one subject to another. The nullification of the destination at the U level is being done by a U subject. One might argue that there is a downward signaling channel with a human in the loop. The human is however trusted not to let the channel be exercised without good cause. Finally note that the U user who executed step 1 of the scenario may again try to enter Talos as the destination, which brings us within the scope of polylow.

2.3 Polylow Example

Our example for polylow is similar to the polyhigh example with the difference that the two update operations occur in the opposite order. So again consider the following relation SOD where Starship is the apparent primary key.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	null	U	U

This time consider the following scenario.

1. A S user modifies the destination of the Enterprise to be Rigel. The relation is modified to appear respectively as follows to U and S users. Note that U users see no change in the relation.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	null	U	U

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Rigel	S	S

2. A U user updates the destination of the Enterprise to be Talos. We cannot reject this update on the grounds that a secret destination for the Enterprise already exists, because that amounts to establishing a downward signaling channel. We can overwrite the destination field in place at the cost of destroying secret data. This would give us the following relation for both U and S users.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U

For obvious reasons this alternative has not been seriously considered by most researchers. That leaves us the option of polyinstantiation which will modify the relation at the end of step 1 to the following for U and S users respectively.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U
Enterprise	U	Exploration	U	Rigel	S	S

This is exactly the same relation as obtained at the end of step 2 in our polyhigh example. The possible interpretations are therefore similar, i.e., we either have a temporary inconsistency or a cover story (the latter alternative has already been rejected for our database). The temporary inconsistency can be corrected by having a U subject (possibly created by a S user logged in at the U level) nullify the Talos destination. But the inconsistency may recur again and again.

3 RESTRICTED POLYINSTANTIATION

In the previous section we have examined the source of polyinstantiation and identified polyhigh and polylow as the two different ways in which polyinstantiation arises. In this section we consider applications which have the following requirements.

1. Downward signaling channels cannot be tolerated.
2. The simple security and $\star$-properties must be enforced for all subjects, i.e., no trusted code can be used.
3. Temporary inconsistencies cannot be tolerated.
4. Denial of data entry service to high users cannot be tolerated.

Moreover each of these requirements has equal importance and one cannot be sacrificed for another. The scenarios of the polyhigh and polylow examples of the previous section show that polyinstantiation by itself cannot meet these requirements simultaneously. One requirement or the other must give in some way.

In this section we show how all four requirements identified above can be simultaneously met. We describe our solution as *restricted polyinstantiation.* 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." We define the semantics of "restricted" in such a way that we are able to eliminate both polyhigh and polylow. "Null" has exactly the same semantics as any other data value and needs no special treatment.

Let us now play out the polyhigh and polylow scenarios of the previous section to intuitively motivate our solution. A formal description of the update protocols is given in the next section.

3.1 Polyhigh Example Revisited

Consider again the following relation SOD where Starship is the apparent primary key.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	null	U	U

Now consider the following scenario.

1. A U user updates the destination of the Enterprise to be Talos. The relation is therefore modified as follows.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U

2. Next a S user attempts to modify the destination of the Enterprise to be Rigel. We cannot polyinstantiate even temporarily, so we must reject this update. Do we have denial-of-service to the S user? No, because the S user can obtain service as follows.

   Step 2a. The S user first logs in as a U-subject and marks the destination of the Enterprise as restricted giving us the following relation.[‖]

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	restricted	U	U

   The meaning of restricted is that this field can no longer be updated by a U user. U users can therefore infer that the true value of Enterprise's destination is classified at some level not dominated by U.

   Step 2b. The S user then logs in as a S-subject and enters the destination of the Enterprise as Rigel giving us the following relations at the U and S levels respectively.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	restricted	U	U

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Rigel	S	S

[‖]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, by say direct personal communication or a secure telephone call.

How does this differ from the scenario of section 2.2 (where the end result after cleaning up the temporary inconsistency was as above except that we have null instead of restricted)? The main difference is that, after step 2a, U users are no longer able to update the destination of the Enterprise. In particular, attempts by U users to reenter Talos as the destination of Enterprise will be rejected on the grounds that the field is restricted. Therefore the relation is guaranteed to be consistent till such time as the restricted value is eliminated. Consideration of who should be allowed to enter and remove the restricted value is deferred for now.

Does step 2a introduce a signaling channel? Yes, but this signaling channel is very similar to the one resulting from the nullification of Talos at the U-level in the example of section 2.2. Both involve a trusted S user in the loop who presumably will ensure that the channel is not exercised wantonly, but rather that this inference is permitted only when the real world situation is actually so. Such a channel with trusted humans in the loop can be exercised only by Trojan Horses that are capable of manipulating the real world. This entails the manipulation of real trusted people making real decisions and not merely the manipulation of bits in a database.

3.2 Polylow Example Revisited

Now consider the two update operations in the opposite order. So again we begin with the following relation SOD where Starship is the apparent primary key.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	null	U	U

This time consider the following scenario.

1. A S user modifies the destination of the Enterprise to be Rigel. This update is rejected! Instead the S user is asked to go through steps 2a and 2b of section 3.1 giving us the following relations at the U and S levels respectively.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	restricted	U	U

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Rigel	S	S

2. A U user updates the destination of the Enterprise to be Talos. The update is rejected on the grounds that the field is restricted.

Note that there is no denial-of-service to the S user. What is happening is a denial of improper service, i.e., there is a protocol for entering high data which all S users are required to follow. Failure to follow the protocol results in denial-of-service but this can hardly be considered a security breach. The denial-of-service to the U user is, of course, only appropriate in this situation.

There is a crucial difference between this protocol and the one discussed in section 2.1. In both cases entry of high data is enabled by an action of a low subject. Our protocol requires the low subject to enter the "restricted" value in the data element. In section 2.1 the suggestion was for the low subject to enter a "null" value. The key difference in the two cases is that a null value can be made non-null by a low Trojan Horse, whereas the restricted value cannot be made unrestricted by a low Trojan Horse. The latter operation requires a special privilege whose distribution is carefully controlled by non-discretionary means. This privilege is available only to selected low subjects who are trusted to exercise its use properly.

4 THE PREVENT PROTOCOLS

In this section we precisely define the collection of update protocols illustrated by example in the previous section. We collectively call this collection the *prevent protocols* because they prevent polyinstantiation due to either polyhigh or polylow from occurring. These protocols can be implemented entirely by untrusted subjects, i.e., subjects which are not exempted from the simple security or $\star$-properties.

4.1 Multilevel Relations

We begin by reviewing some basic concepts and notation for multilevel relations. Let A_1, C_1, A_2, C_2, ..., A_n, C_n denote the attributes (columns) of a multilevel relation R with element level labeling. Each A_i is a *data attribute* and each C_i is the *classification attribute* for A_i. A data attribute can take on values from its natural domain D_i extended with two special values, "null" and "restricted," whose meaning will be defined shortly. We assume that each C_i can take on any value c in the security lattice.** We require that C_i cannot be null. Finally R has a collection of *relation instances* R_c one for each access class c in the given lattice.

Assume there is a user-specified primary key AK consisting of a subset of the data attributes A_i. We call AK the *apparent primary key* of the multilevel relation scheme. In general AK will consist of multiple attributes. We have the following requirement in analogy to entity integrity in the standard relation model. (The notation $t[A_i]$ denotes the value of the A_i attribute in tuple t, and similarly for $t[C_i]$.)

Property 1 [Entity Integrity] Instance R_c of R satisfies entity integrity iff for all $t \in R_c$: (i) AK is uniformly classified in each tuple, i.e., $A_i, A_j \in AK \Rightarrow t[C_i] = t[C_j]$, and (ii) the classification of each non-key data attribute dominates the classification of the apparent key, i.e., $A_i \notin AK \Rightarrow t[C_i] \geq t[C_{AK}]$ where C_{AK} is the classification of AK. □

The notions introduced thus far are standard ones first introduced in the SeaView model [7]. Our next requirement severely limits polyinstantiation and distinguishes the approach of this paper from previous work on element-level labeling (such as [3, 4, 5, 6, 7]).

Property 2 [Key Integrity] R satisfies key integrity iff for every R_c we have for all $i : AK, C_{AK} \rightarrow A_i, C_i$. □

This property stipulates that the user-specified apparent key AK, in conjunction with key-classification C_{AK}, functionally determines all other attributes. In other words R_c cannot have more than one tuple for a given combination of values for AK and C_{AK}. That is, the real primary key of the relation is AK, C_{AK}. The effect of key integrity is to rule out instances such as the following.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U
Enterprise	U	Exploration	U	Rigel	S	S

The reason for rejecting this instance is its inconsistency in specifying two different destinations—one secret and one unclassified—for the Enterprise. Recall our assumption that cover stories are not to be incorporated by polyinstantiation, so interpretations such as discussed in [5] do not apply in this situation. Key integrity does allow instances such as the following where there is polyinstantiation of the key.

**In practice of course it is desirable to place appropriate upper and lower bounds on each C_i. This will only require minor changes to the following discussion.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U
Enterprise	S	Spying	S	Rigel	S	S

In this case we interpret the two tuples as describing two distinct Starships which happen to have the same name.

The next property is concerned with consistency between relation instances at different access classes. Here again we depart from the analogous property defined in [5, 6, 7].[††]

Property 3 [Inter-Instance Integrity] R satisfies inter-instance integrity iff for all $c' \leq c$ we have $R_{c'} = \sigma(R_c, c')$ where the *filter function* σ produces the c'-instance $R_{c'}$ from R_c as follows:

1. For every tuple $t \in R_c$ such that $t[C_{AK}] \leq c'$ there is a tuple $t' \in R_{c'}$ with $t'[AK, C_{AK}] = t[AK, C_{AK}]$ and for $A_i \notin AK$

$$t'[A_i, C_i] = \begin{cases} t[A_i, C_i] & \text{if } t[C_i] \leq c' \\ <\text{restricted}, c'> & \text{otherwise} \end{cases}$$

2. There are no tuples in $R_{c'}$ other than those derived by the above rule. □

The filter function maps a multilevel relation to different instances, one for each descending access class in the security lattice. Filtering limits each user to that portion of the multilevel relation for which he or she is cleared. For instance filtering the following S-instance of SOD

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Rigel	S	S

gives us the following U-instance

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	restricted	U	U

4.2 Update Protocols

In section 4.1 we have identified integrity properties for multilevel relations considered at some instant in time as static objects. We now consider the dynamic behavior of these relations by considering their update semantics. We emphasize that our protocols do not require any exception from the simple security or $\star$-properties.[‡‡] There are three subcases to consider as follows.

4.2.1 Data Value Update

By the term *data value* we mean any value other than "restricted." Our first protocol addresses the case where the value of attribute $t[A_i]$ is changed from its previous data value to a new data value, i.e., neither the previous value nor the new one can be "restricted." "Null" does not need any special treatment in our protocols and is viewed as just another data value. We have the following update protocol.

[††]The definition of the filter function given in [5, 6, 7] differs from the one given here in that <restricted,c'> is replaced by <null,$t[C_{AK}]$>.

[‡‡]Note that the protocols can be simplified if trusted subjects which are exempted from these properties are allowed in selected situations. In particular the protocol to change a restricted value to unrestricted (see section 4.2.3) would be considerably simplified by using a trusted subject which is exempted from the $\star$-property.

Protocol 1 $t[A_i]$ can be changed from its previous data value to a new data value by a c-user only if $t[C_i] = c$.

The effect of this update operation is defined as follows.

1. The value of $t[A_i]$ is changed to its new value in all relation instances $R_{c'}$, $c' \geq c$. The value of $t[C_i]$ remains unchanged as c in all $R_{c'}$, $c' \geq c$.
2. All other instances of R remain unchanged. □

Note that the precondition for this protocol is stated as a necessary condition ("only if"). It is thus a mandatory requirement. In addition to this mandatory pre-condition we may as usual impose further mandatory and/or discretionary controls.

To illustrate the protocol consider the following U and S instances of SOD respectively.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	restricted	U	U

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Rigel	S	S

An update by a U user to change the Objective from "Exploration" to "Mining" has the following effect.

Starship		Objective		Destination		TC
Enterprise	U	Mining	U	restricted	U	U

Starship		Objective		Destination		TC
Enterprise	U	Mining	U	Rigel	S	S

That is the update takes effect at both the U and S levels. An attempt by a S user to change the Objective attribute would be rejected. So would an attempt by a U user to change the Destination attribute. A S user may change the Destination attribute to say "Talos" giving us the following U and S instances of SOD respectively.

Starship		Objective		Destination		TC
Enterprise	U	Mining	U	restricted	U	U

Starship		Objective		Destination		TC
Enterprise	U	Mining	U	Talos	S	S

To appreciate how "null" is treated just like any other data value consider what happens if a S user nullifies the Destination attribute. We get the following U and S instances of SOD respectively.

Starship		Objective		Destination		TC
Enterprise	U	Mining	U	restricted	U	U

Starship		Objective		Destination		TC
Enterprise	U	Mining	U	null	S	S

The Destination attribute remains restricted for U users and the null value is shown only to S users. The classification of the null at S signifies that data in this field can only be entered by S users. If the Destination attribute has a null value at the U level then both U and S instances of SOD must be as follows.

Starship		Objective		Destination		TC
Enterprise	U	Mining	U	null	U	U

In this case U users are allowed to enter data for the Destination attribute whereas S users are not permitted to do so. In order to enable S users to change the Destination of the Enterprise we must first restrict this field at the U level. This brings us to our next protocol.

4.2.2 Update from Unrestricted to Restricted

Let us first consider the case where the security lattice is totally ordered (i.e., there are no compartments). An update of attribute A_i in tuple t from some existing data value to "restricted" is performed as follows.

Protocol 2 $t[A_i]$ can be changed from its previous data value to "restricted" by a c-user only if $t[C_i] = c$.
The effect of this update operation is defined as follows.

1. The value of $t[A_i, C_i]$ is changed to $<$restricted, $c>$ in the instance R_c.
2. Let $\pi(c)$ be the immediate predecessor of c (i.e., $\pi(c) > c$ and there is no c' such that $\pi(c) > c' > c$). The value of $t[A_i, C_i]$ is changed to $<$null, $\pi(c)>$ in all instances $R_{c'}, c' > c$.
3. All other instances of R remain unchanged. □

It suffices to have the pre-condition $t[C_i] = c$ for this operation because, in conjunction with the inter-instance integrity property, $t[C_i] = c$ implies

$$(\forall c' : t[C_{AK}] \leq c' < c) \; t[A_i, C_i] = <\text{restricted}, c'> \text{ in } R_{c'}$$

In other words a data element can be made restricted at level c only if its data value is currently classified at level c, which in turn implies that the data element is restricted at all relevant levels below c.

To illustrate the effect of such updates consider the following U instance of SOD (which is identical to the S instance).

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Rigel	U	U

A U user can change the destination of the Enterprise to be "restricted" giving us the following U and S instances.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	restricted	U	U

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	null	S	S

Now let us consider the general case of a partially ordered security lattice. The problem with partially ordered labels lies in step 2 in defining the effect of protocol 2. In a partial ordering there may be multiple immediate predecessors of c so $\pi(c)$ is no longer uniquely defined. As part of the update operation we have to designate one of c's immediate predecessors as the distinguished one which will remain unrestricted. All other immediate predecessors become restricted. Let $\pi(c)$ denote the distinguished immediate predecessor. Step 2 of protocol 2 needs to be restated as follows.

2$'$. The value of $t[A_i, C_i]$ is changed as follows for all instances $R_{c'}, c' > c$.

$$t[A_i, C_i] = \begin{cases} <\text{null}, \pi(c)> & \text{if } c' \geq \pi(c) \\ <\text{restricted}, c'> & \text{if } c' \not\geq \pi(c) \end{cases}$$

As an example consider a lattice with four labels, S, U, M_1 and M_2; where M_1 and M_2 are both dominated by S and both dominate U, but M_1 and M_2 are themselves incomparable. Suppose we have the following instance of SOD at all four levels.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Rigel	U	U

Let a U user make the Destination field of the Enterprise "restricted" while designating M_1 to be π(U) for this update. The U, M_1, M_2 and S instances of SOD will respectively become as follows.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	restricted	U	U

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	null	M_1	M_1

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	restricted	M_2	M_2

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	null	M_1	M_1

4.2.3 Update from Restricted to Unrestricted

Again for simplicity let us first consider the case where the lattice is totally ordered. We have the following protocol for making a field unrestricted.

Protocol 3 $t[A_i]$ can be changed from its current value of "restricted" to a data value dv only by a c-user.

The effect of this update operation is defined as follows.

1. The value of $t[A_i, C_i]$ is changed to $<dv, c>$ in all instances $R_{c'}$, $c' \geq c$.
2. All other instances of R remain unchanged. □

The pre-condition for this update, that $t[A_i, C_i] = <\text{restricted}, c>$ in R_c, is sufficient to ensure that $t[A_i, C_i] = <\text{restricted}, c'>$ in all R'_c, $c' \leq c$ (due to inter-instance integrity).

The protocol will overwrite any existing data value for $t[A_i]$ in instances R'_c, $c' > c$. This operation therefore has the potential for creating integrity problems by overwriting existing higher level data. We have rejected this approach as a general solution in section 2. Here we are proposing to employ it for the specific purpose of converting a field from restricted to unrestricted. We require that this be a specially privileged operation so that we can be sure it is executed only when the real world conditions warrant it. We will return to this point in the next section.

To illustrate this operation consider the following U and S instances of SOD.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	restricted	U	U

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	null	S	S

A suitably privileged U user can change the value of the Destination attribute in this tuple to be say "Talos" giving us the following (identical) U and S instances of SOD.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U

Next let us consider the case of a partially ordered security lattice. The pre-condition of protocol 3 is no longer sufficient. Before a c user is allowed to change a restricted field to non-restricted we must ensure that field is restricted at all levels which do not dominate c. This includes levels which are dominated by c as well as levels incomparable with c. The latter requirement cannot be checked by a c user without violating simple-security. We circumvent this problem by requiring the update of protocol 3 to occur in two phases as follows.

1. *Preparatory Phase.* Login at level $t[C_{AK}]$ and set

$$t[A_i, C_i] = <\text{restricted}, c'> \quad \text{in all instances } R'_c,\ c' \geq t[C_{AK}]$$

   i.e., set $t[A_i]$ to "restricted" at all levels where tuple t is visible.

2. *Update Phase.* Login at level c and set $t[A_i, C_i] = <dv, c>$.

The net effect of this modified protocol is to set

$$t[A_i, C_i] = \begin{cases} <dv, c> & \text{in all instances } R_{c'},\ c' \geq c \\ <\text{restricted}, c'> & \text{in all instances } R_{c'},\ c' \not\geq c \end{cases}$$

For example consider the following U, M_1, M_2 and S instances of SOD respectively taken from the end of section 4.2.2.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	restricted	U	U

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	null	M_1	M_1

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	restricted	M_2	M_2

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	null	M_1	M_1

The preparatory phase will give us the following U, M_1, M_2 and S instances of SOD respectively.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	restricted	U	U

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	restricted	M_1	M_1

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	restricted	M_2	M_2

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	restricted	M_2	M_2

In other words the preparatory phase restricts the Destination attribute of this tuple at all levels above U (which is the key class of the tuple). Subsequently, the update phase results in (say) the following U, M_1, M_2 and S instances of SOD respectively.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	restricted	U	U

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	restricted	M_1	M_1

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Rigel	M_2	M_2

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Rigel	M_2	M_2

5 ASSURANCE

In this section we briefly consider how the prevent protocols can be enforced.

Our first observation is that all our protocols adhere to both simple security and the $\star$-property. They can therefore be enforced by a DBMS trusted computing base (TCB) to the highest assurance standards without the use of subjects which are exempt from simple-security or the $\star$-property.

Secondly, our protocols are designed to achieve integrity and availability-of-service in addition to secrecy. The secrecy objective can be enforced to A1 standards by strict enforcement of simple security and the $\star$-properties. In order to achieve the integrity and availability of service requirements we need controls beyond the traditional simple security and $\star$-property. Let us consider each of the following three cases in turn.

5.1 Data Value Update

This is the simplest case where our multilevel relations behave much as conventional single-level relations do. It is obvious that in a high integrity system updates must be carefully controlled even within a single security level. Conventional databases use mechanisms such as well-formed transactions and least privilege for this purpose [2, 8]. The DBMS TCB must provide high assurance support for such mechanisms. We do not need any additional mechanisms for multilevel DBMSs. The required mechanisms should anyway be available in high-quality single-level DBMSs as discussed in [8].

5.2 Update from Unrestricted to Restricted

Assigning a restricted value to a field with classification c requires a check that this field is already restricted at levels below c. This is feasible within the scope of simple security. In high assurance systems this application-independent pre-condition should be checked by the DBMS TCB. At lower levels of assurance the pre-condition may be tested by individual transactions rather than the DBMS.

The effect of restricting a field at the c level is dangerous in that it can cause denial-of-service to c users. So when the destinations of all our flights are made restricted, when they should not be, we might end up grounding the entire fleet! Therefore the ability to mark a field as restricted should be a carefully controlled privilege. This privilege should be assigned to a few subjects who need to do this operation. We can ensure that this privilege cannot be acquired except by some very special non-discretionary means such as involving intervention by a security officer.

The general problem of incorrect data essentially exists whether or not we recognize restricted as a special value. For suppose a malicious program running at the U level, and obeying simple security and $\star$-property, sets the destination of all flights to be Dayton, Ohio. Does the entire fleet converge on Wright Patterson Air Force Base? Presumably a high integrity system has corrective measures to detect and recover from such errors. In principle, incorrectly restricted fields present a similar problem except that recovery may be slightly more cumbersome.

5.3 Update from Restricted to Unrestricted

An update from restricted to unrestricted is different from the previous two cases because we cannot test the pre-conditions for this action within the confines of simple security. If we wish to prevent overwriting of high data by this operation we have to check that no high data exists (i.e., no non-null high data exists). In view of simple security this is not feasible. Therefore we define the operation as potentially overwriting high data. It follows that we must strictly control the ability to make a restricted value unrestricted. The control in this case should be even stricter than in the case of update from unrestricted to restricted. Alternately, we can use a trusted subject for this operation.

6 CONCLUSION

In this paper we have shown how both the polyhigh and polylow variations of polyinstantiation can be eliminated by our solution of restricted polyinstantiation. This allows us to avoid downward signaling channels, inconsistencies, denial of data entry to high users and the overwriting of high data by low subjects while providing element-level labeling. This is the first solution to meet all these requirements simultaneously.

In conclusion we wish to note that restricted polyinstantiation makes a particular trade-off among conflicting objectives. It may be eminently suitable to most applications. Yet we would advise against having this as the only option. Databases are long lived and develop a great deal of inertia over their life. Moreover different applications may call for different trade-offs. For example temporary inconsistencies may be preferred to inconvenience in data entry. General-purpose multilevel secure DBMSs must cater to such applications too. Therefore our recommendation is that restricted polyinstantiation be available as one of several options that a multilevel secure DBMS supports.

Acknowledgment

We are indebted to John Campbell, Joe Giordano, and Howard Stainer for their support and encouragement, making this work possible. The opinions expressed in this paper are of course our own and should not be taken to represent the views of these individuals.

References

[1] Burns, R.K. "Referential Secrecy." *IEEE Symposium on Security and Privacy*, Oakland, California, May 1990, 133-142.

[2] Clark, D.D. and Wilson, D.R. "A Comparison of Commercial and Military Computer Security Policies." *IEEE Symposium on Security and Privacy*, 184-194 (1987).

[3] Denning, D.E., Lunt, T.F., Schell, R.R., Heckman, M., and Shockley, W.R. "A Multilevel Relational Data Model." *IEEE Symposium on Security and Privacy*, 220-234 (1987).

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

[5] Jajodia, S. and Sandhu, R.S. "Polyinstantiation Integrity in Multilevel Relations." *IEEE Symposium on Security and Privacy*, Oakland, California, May 1990, 104-115.

[6] Jajodia, S., Sandhu, R.S. and Sibley, E. "Update Semantics for Multilevel Relations." *Sixth Annual Computer Security Applications Conference*, Tucson, Arizona, December 1990, pages 103-112.

[7] Lunt, T.F., Denning, D.E., Schell, R.R., Heckman, M. and Shockley, W.R. "The SeaView Security Model." *IEEE Transactions on Software Engineering*, 16(6):593-607 (1990).

[8] Sandhu, R.S. and Jajodia, S. "Integrity Mechanisms in Database Management Systems." *13th NIST-NCSC National Computer Security Conference*, Washington, D.C., October 1990, 526-540.

[9] Wiseman, S.R. "On the Problem of Security in Data Bases." In *Database Security III: Status and Prospects*, (Spooner, D.L. and Landwehr, C.E., editors), North-Holland, pages 143-150 (1990). Also available as Royal Signal and Radar Establishment, U.K., Memo 4263.