An Algebra for Belief Persistence in Multilevel Security Databases

Tsz Shing Cheng
Iowa State University

Shashi K. Gadia
Iowa State University, gadia@iastate.edu

Follow this and additional works at: http://lib.dr.iastate.edu/cs_techreports
Part of the Information Security Commons, Systems Architecture Commons, and the Theory and Algorithms Commons

Recommended Citation
http://lib.dr.iastate.edu/cs_techreports/39

This Article is brought to you for free and open access by the Computer Science at Iowa State University Digital Repository. It has been accepted for inclusion in Computer Science Technical Reports by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
An Algebra for Belief Persistence in Multilevel Security Databases

Abstract
In a multilevel security environment, the security levels form a hierarchy which is generally assumed to be a lattice. A user can see not only its own information, but also information belonging to lower users. In a multilevel security database, different users have different beliefs (versions of information) about the same real world object. In this paper we present a relational model SecDB for multilevel security data. We also present an SQL-like language SecSQL for querying and updating security information. For a given level, a tuple consists of all the differing beliefs about the same real world object. Therefore, the model provides a built-in coherence to different beliefs of the same real world object. For an operator to be well defined, its application should preserve beliefs and coherence. This persistence of belief and coherence is achieved through the concept of an anchor borrowed from an earlier work. On one hand (in addition to the usual database queries) SecSQL yields itself naturally to formulation of security related queries, yet on the other hand the algebraic operators yield natural identities which hold a good promise of algebraic optimization.

Disciplines
Information Security | Systems Architecture | Theory and Algorithms
An algebra for belief persistence in multilevel security databases

Tsz Shing Cheng and Shashi K. Gadia
Department of Computer Science
Iowa State University
Ames, IA 50011
{tcheng,gadia}@cs.iastate.edu; (515)294-4377
September 1995

Abstract. In a multilevel security environment, the security levels form a hierarchy which is generally assumed to be a lattice. A user can see not only its own information, but also information belonging to lower users. In a multilevel security database, different users have different beliefs (versions of information) about the same real world object. In this paper we present a relational model SecDB for multilevel security data. We also present an SQL-like language SecSQL for querying and updating security information. For a given level, a tuple consists of all the differing beliefs about the same real world object. Therefore, the model provides a built-in coherence to different beliefs of the same real world object. For an operator to be well defined, its application should preserve beliefs and coherence. This persistence of belief and coherence is achieved through the concept of an anchor borrowed from an earlier work. On one hand (in addition to the usual database queries) SecSQL yields itself naturally to formulation of security related queries, yet on the other hand the algebraic operators yield natural identities which hold a good promise of algebraic optimization.

Index terms. Databases, security, database security, multilevel security, relational databases, beliefs, belief data, dimensional databases.

1. Introduction

In a multilevel security environment, the user levels form a hierarchy which is generally assumed to be a lattice. Every user has its own object space. We assume that an object is uniquely identified by its key values. Information available to a user consists of the following:

• The object space which is a belief about which objects exist in the real world.
• Property values of objects in the user’s object space.
• Knowledge of which object in user’s object space is known to a lower user, possibly with different identity (key) and property values.
• All information available to lower users.

1. We use the terms user, security level, security class and user level interchangeably. For the sake of simplicity one may assume that there is only one user at a given level.
As an example suppose object space of \( u_1 \) is \{John, Mary\}, and object space of \( u_2 \) is \{Inga, John, Tom\}, as shown in Figure 1. Then \( u_1 \) believes that there are only two objects, whereas \( u_2 \) believes there are three objects in the real world. In addition \( u_1 \) believes that John is known as Tom to \( u_2 \), Mary is unknown to \( u_2 \), and the persons known as Inga and John to \( u_2 \) do not exist in the real world. The user \( u_2 \) is not aware of the object space of \( u_1 \).

![Object spaces of users](image)

Figure 1. Object spaces of users

It is clear that an upper user has read access to lower users. This provides a lower user a write access to upper users. To maintain confidentiality of information, the upper user should not have a write access to a lower user, or equivalently, a lower user should not have a read access to the upper user. In a model for secure databases it is important that such communication can not be created covertly by users through the facilities available in the model. In other words the model should avoid covert channels. For example, consider a model where a lower user requesting a tuple insertion gets a system rejection because a classified tuple with the same key already exists in the relation. If this is permitted, a user with high security clearance can misuse it to send one bit of information to a user with low security clearance. This would create a covert channel. Polyinstantiation, i.e. multiple instantiations of keys seems to be a necessary condition to avoid covert channel.

Several papers have given models for multilevel security database. Some of the papers [JS90, JS91, DLS87, HOT91] have proposed integrity rules to avoid or interpret polyinstantiation. [SW92] explains some issues in multilevel security and proposes a belief based semantics for multilevel security database. Although problems like covert channel can be controlled or resolved in various ways in the existing models, but among other issues true multilevel identity is not supported. The key of an object classified secret has to be the same at an unclassified level, even though at the unclassified level the key may be different in the real world. Thus the fact that both identities belonging to the same object is lost. [CS95] seems to draw ideas from numerous sources including [DLS88, LDS90, HOT91, JS90, JS91, SW92] and proposes a model and incorporates several constraints prevalent in the security literature. An additional interesting feature of this model is that a higher user can associate an attribute value of an object with attribute value of an object known to a lower level user. We give a more detailed account of [JS91], [SW92] and [CS95] in Section 5.

In this paper we present a relational model SecDB for multilevel security data. We also present an SQL-like language SecSQL for querying and updating security information. For a given level,
a tuple in our model consists of all the differing beliefs about the same real world object. Therefore, the model provides a built-in coherence to different beliefs of the same real world object. For an operator to be well defined, its application should preserve beliefs and coherence. This persistence of belief and coherence is achieved through the concept of anchor borrowed from an earlier work [BG90]. On one hand (in addition to the usual database queries) SecSQL yields itself naturally to formulation of security related queries, yet on the other hand the algebraic operators yield natural identities which hold a good promise of algebraic optimization.

Our solution to the multilevel security has been provided within the framework of dimensional data. The term dimensional databases applies to databases where there is an underlying dimensional space and property values are valid at points in the dimensional space. Examples of dimensional databases are temporal and spatial databases. Considerable research has already been done in this area, and much of it extends to security data.

The rest of the paper is organized as follows. In Section 2 we present our model, SecDB, supporting multilevel identity. In Section 3 we present a query language for this model. In Section 4 we present the update mechanism. We give a more detailed account of [JS91, SW92] in Section 5. In Section 6 we discuss several interesting features of our model in the context of dimensional databases. In Section 7 we give our conclusions.

2. SecDB, a model for multilevel security database

In this section we present our model, SecDB, for multilevel security data. We develop the concepts of security space, object space, security elements, attribute values, tuples and relations. We will introduce the concept of belief consistency for our relations. Most important of all we will introduce the concept of anchor. The anchors become necessary to let the beliefs persist beyond application of algebraic operators.

2.1. Lattice of users

We assume that we are given an arbitrary finite lattice $\Lambda$ of users, called the security space. As our running example we consider the lattice $\{\alpha\beta, \alpha, \beta, \lambda\}$, as shown in Figure 2, together with the partial order $\leq$. (It may help the reader to think of $\alpha\beta, \alpha, \beta, \lambda$ as $\{\alpha, \beta\}, \{\alpha\}, \{\beta\}, \emptyset$, respectively, and $\leq$ as $\subseteq$.) Suppose $u_1 \leq u_2$, then we say $u_2$ dominates $u_1$, $u_2$ is above $u_1$, $u_1$ is below $u_2$ or $u_1$ is dominated by $u_2$; in addition in the context of $u_1$ we call $u_2$ a higher user, and similarly, in the context of $u_2$, we call $u_1$ a lower user. Thus the user $\alpha\beta$ dominates all users $\alpha\beta, \alpha, \beta$ and $\lambda$. For a user $u$, the set of all users dominated by $u$ is denoted as $u^-$. E.g. $\alpha^- = \{\alpha, \lambda\}$.

![Figure 2. A lattice $\Lambda$ of security levels](image-url)
2.2. Security element

A security element \( \mu \) is a subset of the security space \( \Lambda \). A security element allows us to represent a user’s knowledge about a real world object. Clearly, the set of all security elements is closed under union, intersection and complementation. These closure properties lay a foundation for uniform handling of “or”, “and” and “not” of natural languages in a query language [GN93]. A \( \mu \)-element is a subset of \( \mu^- \). An example of an \( \alpha\beta \)-element is \( \{\alpha\beta, \alpha, \lambda\} \). The purpose of this security element is to allow the user \( \alpha\beta \), to assemble a belief about a real world object known to \( \alpha\beta \) (ownself), \( \alpha \) and \( \lambda \). The \( \mu \)-elements are closed under union, intersection. The \( \mu \)-elements are also closed under complementation when the complementation is computed with respect to \( \mu^- \).

2.3. Attribute values

A \( \mu \)-assignment to an attribute \( A \) is a function \( \xi \) from a \( \mu \)-element into \( \text{dom}(A) \), the domain of \( A \). For example, \( \langle \{\alpha\beta, \alpha\} \text{ Jack}, \{\beta\} \text{ John}, \{\lambda\} \text{ Tom} \rangle \) is an \( \alpha\beta \)-assignment to the attribute NAME. Whereas, in the conventional 1nf approach, an attribute value in existing multilevel security models capture information pertaining to only one level of a belief, in our model we assemble the entire information about a belief in a (single) attribute value. The non-Inf nature of our tuples makes a substantial simplification in user queries [GN93]. The domain of \( \xi \) is denoted as \( \llbracket \xi \rrbracket \). For example, \( \llbracket \langle \{\alpha\beta, \alpha\} \text{ Jack}, \{\beta\} \text{ John}, \{\lambda\} \text{ Tom} \rangle \rrbracket = \{\alpha\beta, \alpha, \beta, \lambda\} \).

2.4. Anchors: incorporating belief persistence in attribute values

The attribute value \( \langle \{\alpha\beta, \alpha\} \text{ Jack}, \{\beta\} \text{ John}, \{\lambda\} \text{ Tom} \rangle \) is a belief belonging to the user \( \alpha\beta \). A graphical representation of this object is shown in Figure 3(a), and a tabular representation in Figure 3(b). It may seem as if an attribute value always belongs to the highest user mentioned in the attribute value. However this is not a desirable characterization. For example, if a database query selects \( \langle \{\beta\} \text{ John}, \{\lambda\} \text{ Tom} \rangle \) from the above object, the fact that this information belongs to \( \alpha\beta \) is lost. Thus to make an attribute value algebraically persistent, we need to enhance it. For this purpose we add an anchor to an attribute value. The anchor for the above attribute value is \( \langle\alpha\beta \text{ John}\rangle \). Attribute values are used to build tuples, and tuples are used to build relations. In the context of a relation it helps to know not only whose belief we have (the belief is \( \alpha\beta \)’s in this case), but also which belief it is (the belief is the object “John” in this case). The concept of an anchor has been employed in [BG90]; our solution for multilevel security in some ways similar to, and in

![Figure 3. Objects and anchoring](image-url)
other ways different from that work.

**Example 1.** The attribute value \(\langle \{\alpha\beta, \alpha\} \text{ Jack, } \{\beta\} \text{ John, } \{\lambda\} \text{ Tom}\rangle\) after anchoring becomes \(\langle \alpha\beta \text{ Jack: } \{\alpha\beta\}, \alpha \text{ Jack, } \{\beta\} \text{ John, } \{\lambda\} \text{ Tom}\rangle\). Figure 3(c) shows a tabular representation of this anchored attribute value.

### 2.5. Use of anchors in achieving persistence of algebraic identities

The information in belief data can be fragile when algebraic operators are applied. Our goal is to design a query language which is as natural for a user as possible. In order to achieve this we favor passing the complexities of belief data on to the system. Anchors help us in meeting this non-trivial objective.

We illustrate the use of anchors through an example. We informally consider a selection of the form \(\sigma(r, \mu)\). The selection retrieves the functional restriction of relation \(r\) to the security element \(\mu\). Next consider the following algebraic identity.

\[\sigma(r, \mu) \cup \sigma(r, \nu) = \sigma(r, \mu \cup \nu)\]

Intuitively a user expects this algebraic identity to hold. First we will show that naive computations will fail to yield the algebraic identity. Then we will apply anchors and show how the algebraic identity is obtained. We will use the following values for \(r, \mu\) and \(\nu\):

- For \(r\) we will use the single object of Figure 3. (In other words \(r\) is a single attribute single tuple relation.)
- \(\mu = \{\alpha, \beta\}\)
- \(\nu = \{\beta, \lambda\}\)

The naive computations are shown in Figure 4(a). In this case \(r\) is the un-anchored object of Figure 3(b). Starting with \(r\), we compute \(\sigma(r, \mu), \sigma(r, \nu), \sigma(r, \mu) \cup \sigma(r, \nu)\) and \(\sigma(r, \mu \cup \nu)\). From parts (iii) and (iv) of Figure 4(a), it is clear that \(\sigma(r, \mu) \cup \sigma(r, \nu)\) and \(\sigma(r, \mu \cup \nu)\) are not equal; they don't even have the same number of tuples!

Figure 4(b) shows computations with anchors. In this case \(r\) is the anchored object of Figure 3(c). With anchoring, the results of \(\sigma(r, \mu), \sigma(r, \nu), \sigma(r, \mu) \cup \sigma(r, \nu)\) and \(\sigma(r, \mu \cup \nu)\) are also shown. Note that when the system computes union (in this case \(\sigma(r, \mu) \cup \sigma(r, \nu)\)), the collapse of two tuples is with respect to their anchors. This permits the restoring of an object (belief) in a single tuple after it is decomposed into several tuples. It is clear that Figure 4(b)(iii) shows the result of \(\sigma(r, \mu) \cup \sigma(r, \nu)\) as well as \(\sigma(r, \mu \cup \nu)\). We note that the anchoring is done automatically by the system before querying. The user makes no effort in the underlying process.

Before we proceed further, we formalize the concept of anchor. The formal treatment appears below between two double lines. Reader may skip it in the first reading and come back to it when in doubt. These annotations may be removed from a published version.

---

### 2.6. Formal treatment of anchored domains

We assume that we have an attribute \(A\) in mind, and \(\text{dom}(A)\) denotes domain of \(A\). An **anchored u-element** is an ordered pair \((\{u\}, \mu)\), where \(\mu\) is a subset of \(u^-\). For ease of readability, we denote a u-element \((\{u\}, \mu)\) as \(u: \mu\).
Example 2. \( \alpha \beta : \{ \alpha \beta, \alpha, \lambda \} \) is an anchored \( \alpha \beta \)-element; its anchor is \( \alpha \beta \) and its domain is \( \{ \alpha \beta, \alpha, \lambda \} \). Another example of an anchored element is \( \beta : \{ \lambda \} \), but \( \beta : \{ \alpha \} \) is not an anchored element, because \( \alpha \) is not a below of \( \beta \).

Now, we define union, intersection and complementation of anchored \( u \)-elements for a fixed value of \( u \). Basically, for a fixed anchor \( u \), these operations are performed on the domains, with the \( u \) as the universe. For example,

\[
\begin{align*}
\alpha \beta : \{ \alpha, \lambda \} \cup \alpha \beta : \{ \beta, \lambda \} &= \alpha \beta : \{ \alpha, \beta, \lambda \} \\
\alpha \beta : \{ \alpha, \lambda \} \cap \alpha \beta : \{ \beta, \lambda \} &= \alpha \beta : \{ \lambda \} \\
\alpha \beta : \{ \alpha, \lambda \} - \alpha \beta : \{ \beta, \lambda \} &= \alpha \beta : \{ \alpha \} \\
- \alpha : \{ \alpha, \lambda \} &= \alpha \beta : \{ \alpha \beta, \lambda \} \\
- \alpha : \{ \alpha, \lambda \} &= \alpha : \emptyset
\end{align*}
\]

It is clear that the above definitions provide each user \( u \), a space of anchored elements which are closed under union, intersection and complementation. However, this definition is not an end in itself. It is not our intention to have different relations for different users. A relation \( r \) is shared by all users and we want to be able to compute its domain.

An anchored domain \( \mu \) is a (finite) set of anchored \( u \)-elements, such that for a given user \( u \), there is at most one \( u \)-element in \( \mu \). This definition forces different \( u \)-elements for the same value of \( u \) to be coalesced together. The union, intersection and complementation among anchored domains is defined naturally.
Example 3. The following is an example show how the user $\alpha\beta$ may perform union, intersection and complementation among anchored domains.

$$
\{\alpha\beta; (\alpha, \lambda), \beta; (\lambda)\} \cup \{\alpha\beta; (\beta, \lambda), \alpha; (\alpha, \lambda)\} = \{\alpha\beta; (\alpha, \beta, \lambda), \alpha; (\alpha, \lambda), \beta; (\lambda)\}
$$

$$
\{\alpha\beta; (\alpha, \lambda), \beta; (\lambda)\} \cap \{\alpha\beta; (\beta, \lambda), \alpha; (\alpha, \lambda)\} = \{\alpha\beta; (\lambda), \alpha; \emptyset, \beta; \emptyset\} = \{\alpha\beta; (\lambda)\}
$$

$$
\{\alpha\beta; (\alpha, \lambda), \beta; (\lambda)\} - \{\alpha\beta; (\beta, \lambda), \alpha; (\alpha, \lambda)\} = \{\alpha\beta; (\alpha, \beta, \lambda)\}
$$

$$
- \{\alpha\beta; (\alpha, \lambda), \beta; (\lambda)\} = \{\alpha\beta; (\alpha\beta, \beta, \lambda), \alpha; (\alpha, \lambda), \beta; (\beta), \lambda; (\lambda)\}
$$

$$
- \{\beta; (\lambda)\} = \{\beta; (\beta), \lambda; (\lambda)\}
$$

Example 4. The user $\beta$ is unaware of $\alpha\beta$ and $\alpha$. Therefore, the user $\beta$ will see the following part of the last identity in the previous example.

$$
\{\beta; (\lambda)\} = \{\beta; (\beta), \lambda; (\lambda)\}
$$

It is easily verified that union, intersection and complementation satisfy properties of a boolean algebra at every step of the definition. As all our definitions in dimensional databases, these definitions are designed to arrive at a query language which is easy to use.

The last definition will allow us to compute $[[r]]$, the domain of a relation. The operator $[[r]]$ is of fundamental importance in dimensional databases. It allows us to capture the concept of “when” in a temporal database, “where” in a spatial database, and “when and where” in a spatio-temporal database. In those databases, the definition of $[[r]]$ is very simple. $[[r]]$ plays a fundamental role in the syntax of our SQL-like algebraic query language. On one hand it makes the syntax highly recursive and natural, and on the other hand it allows users to express “selections” from natural languages as selections instead of joins in our query language. In multilevel security, the definition of $[[r]]$ is complex compared to other forms of dimensional data. It allows us to capture “who” in a secure manner. We remark that the said complexity in the definition of $[[r]]$ is not passed on to the user, but rather it is absorbed by the system.

2.7. Formal treatment of anchored attribute values

A u-anchor over A, or simply an anchor is a function from a singleton $\{u\}$ to dom(A). An anchored u-assignment with the anchored u-element $u; \mu$ as its domain, is a pair of functions $(\xi_u; \xi_v)$, denoted $\xi_u; \xi_v$, where $\xi_u$ is a u-anchor, and $\xi_v$ is a function from $u^-$ into dom(A) such that if $\xi_v$ is defined at $u$, it has to agree with $\xi_u$. The domain of $\xi_u; \xi_v$, denoted $[[\xi_u; \xi_v]]$ is defined as the anchored domain $[[\xi_u]]; [[\xi_v]]$. We complete our formal definitions of anchors by revisiting Example 1.

Example 5. As mentioned in Example 1, the attribute value $\langle \alpha\beta, \alpha \rangle$ Jack, $\langle \beta \rangle$ John, $\langle \lambda \rangle$ Tom belonging to user $\alpha\beta$ after anchoring becomes $\langle \alpha\beta, \alpha \rangle$ Jack, $\langle \alpha\beta, \alpha \rangle$ Jack, $\langle \beta \rangle$ John, $\langle \lambda \rangle$ Tom. Figure 3(c) shows a tabular representation of this anchored attribute value. In addition the domain of the anchored value is $\alpha\beta; (\alpha\beta, \alpha, \beta, \lambda)$.

2.8. Associative navigation

In databases the construct A\theta B is of fundamental importance. In the conventional 1nf approach, the attribute values A and B are atomic, therefore, A\theta B evaluates to TRUE or FALSE. In our dimensional models, A and B are functions, and A(p)\theta B(p) is TRUE at some points p and FALSE at others. Therefore, the appropriate counterpart of A\theta B in our dimensional models is
[[A\theta B]], which computes the set of points \( p \) where \( A(p) \theta B(p) \) holds. In other words, in dimensional data, \([A\theta B]\) is a domain expression and not a boolean expression.

Though in our existing dimensional models the above definition of \([A\theta B]\) is satisfactory, in security data we need to reexamine this in the context of anchors. We could have the following two options:

- Define \([A\theta B]\) as above, i.e. \{p: A(p) \theta B(p) holds\}
- Define \([A\theta B]\) more conservatively as \{p: beliefs A and B belong to the same user and \( A(p) \theta B(p) \) holds\}

We favor a conservative approach, and adopt the second alternative as a default choice. Perhaps the best justification for such a choice can be given in terms of natural join. We encourage the reader to see Example 6 where the natural join (equi-join to be exact) \([\text{emp.DEPT} = \text{management.DEPT}]\) is used with the conservative choice. This choice allows the computation spaces of different users to be kept separate from each other.

2.9. Tuples

A u-tuple, or simply a tuple is a concatenation of u-assignments whose security domains are the same. The assumption that all u-assignments in a tuple have the same domain is called the homogeneity assumption [GN93]. Note that in all the existing models, security level is used as an attribute, and thus tuples in these models are a priori homogeneous. Thus, the homogeneity assumption does not limit our modeling power when compared to the existing approaches. Figure 5(a) shows example of an employee tuple over NAME SALARY DEPT, with NAME as its key. This tuple belongs to the user \( \alpha\beta \). The tuple represents a single real world object, known as Jack to the users \( \alpha\beta \) and \( \alpha \), as John to \( \beta \), and as Tom to \( \lambda \).

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha\beta ) Jack {( \alpha\beta, \alpha )}</td>
<td>( \alpha\beta ) 100K {( \alpha\beta )}</td>
<td>( \alpha\beta ) Toys {( \alpha\beta, \beta )}</td>
</tr>
<tr>
<td>{( \beta )} John</td>
<td>50K {( \alpha )}</td>
<td>( \alpha ) Shoes {( \alpha )}</td>
</tr>
<tr>
<td>{( \lambda )} Tom</td>
<td>80K {( \beta )}</td>
<td>( \lambda ) PCs {( \lambda )}</td>
</tr>
</tbody>
</table>

(a) A tuple

| \( \beta \) John {\( \beta \)} | \( \beta \) 80K {\( \beta \)} | \( \beta \) Shoes {\( \beta \)} |
| John                          | 50K {\( \lambda \)} | \( \lambda \) Shoes {\( \lambda \)} |
| \( \lambda \) Harry          |

(b) A tuple that is belief consistent with the tuple in (a)

| \( \alpha \) Jack {\( \alpha \)} | \( \alpha \) 200K {\( \alpha \)} | \( \alpha \) Shoes {\( \alpha \)} |
| Jack                           | 40K {\( \lambda \)} | \( \lambda \) PCs {\( \lambda \)} |
| \( \lambda \) Tom             |

(c) A tuple that is not belief consistent with the tuple in (a)

Figure 5. Belief consistency among tuples
2.10. Belief consistency among tuples

Suppose \( \tau_1 \) is a tuple belonging to a user \( u_1 \) and \( \tau_2 \) is a tuple belonging to a lower user \( u_2 \), such that the two tuples agree on key attributes at level \( u_2 \). Then the two tuples are said to be belief consistent if they agree on all attributes at level \( u_2 \). Figure 5 shows belief consistent and belief inconsistent tuples. A set of tuples is said to be belief consistent if all its tuples are pairwise belief consistent.

2.11. Relations

A relation \( r \) over a scheme \( R \), with \( K \subseteq R \) as its key, is a finite set of belief consistent non-empty homogeneous tuples such that for a given level, no two tuples can have the same key.

Figure 6 shows an example of a database called PersonnelDB. The database consists of two relations: emp (NAME SALARY DEPT) and management (DEPT MANAGER). We assume that NAME is the key of emp, and DEPT is the key of management. We also assume that DEPT and MANAGER functionally determine each other. Note that in our model we do not need any additional attributes beyond those in a classical relation, yet our modeling power extends the existing models in multilevel security.

Not all data is visible to every user. A user at level \( u \) can query all tuples at level \( u \) and below. For example Figure 6 shows all the data visible to \( \alpha\beta \). Figures 7 and 8 show emp relation as seen by user \( \alpha \) and user \( \lambda \), respectively. Note that the user \( \lambda \) sees a classical database. We also note that the data (in the form it is presented) in Figure 7 is also visible to user \( \alpha\beta \), and data presented in Figure 8 is also visible to users \( \alpha, \beta \) and \( \alpha\beta \). This is because a user has a capability to set itself to a lower user for read only capability. It is obvious that polyinstantiation, a seemingly necessary condition to avoid covert channels, is available in our model.

3. The query language SecSQL

In this section we give a brief introduction to SecSQL, the query language for our model. Expressions in SecSQL can be divided into three mutually exclusive groups: domain expressions which evaluate to security elements, boolean expressions which evaluate to TRUE and FALSE, and relational expressions which evaluate to security relations. The most interesting operator in SecSQL is its SQL-like select statement. The select statement draws its simplicity and power through associative navigation made possible by domain expressions and boolean expressions. In this section we primarily concentrate upon the SQL-like select statement of SecSQL and giving several examples.

• \([[A \theta B]]\) is a domain expression which extracts points in the security space where \( A \) and \( B \) are in \( \theta \)-relationship. For example, if \( A \) is the security assignment \( \langle \alpha \beta \ a_1: \{ \alpha \beta \} a_1, \{ \alpha \} a_2, \{ \lambda \} a_3 \rangle \), and \( B \) is \( \langle \{ \alpha \beta \} a_1, \{ \alpha \} a_2, \{ \beta \} a_3 \rangle \), then \([[A=B]]\) evaluates to \( \alpha\beta: \{ \alpha\beta, \alpha \} \).

• If \( e \) is a relational expression, then \([[e]]\) is a domain expression, whose value is the union of domains of tuples in the relation computed by \( e \). For example, \([[\text{emp}]]\) is \( \{ \alpha\beta: \{ \alpha\beta, \alpha, \beta, \lambda \}, \alpha: \{ \alpha, \lambda \}, \beta: \{ \beta \}, \lambda: \{ \lambda \} \} \). This construct is a source of powerful nesting among SecSQL expressions. It can also be used by itself as a query.

• \([[A \theta B]]\) and \([[e]]\) are atomic domain expressions, they evaluate to security elements. If \( \mu \) and \( \nu \) are domain expressions, then so are \( \mu \cup \nu, \mu \cap \nu, \mu \neg \nu \) and \( \neg \mu \).

• If \( \mu \) and \( \nu \) are domain expressions, then \( \mu \subseteq \nu \) is a boolean expression.
### (a) Relation emp (NAME, SALARY, DEPT)

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>αβ Jack</td>
<td>αβ 100K</td>
<td>αβ Toys</td>
</tr>
<tr>
<td>β John</td>
<td>β 80K</td>
<td>β PCs</td>
</tr>
<tr>
<td>λ Tom</td>
<td>λ 40K</td>
<td>λ PCs</td>
</tr>
</tbody>
</table>

### (b) Relation management (DEPT, MANAGER)

<table>
<thead>
<tr>
<th>DEPT</th>
<th>MANAGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>αβ Toys</td>
<td>αβ Toys</td>
</tr>
<tr>
<td>λ Toys</td>
<td>λ Toys</td>
</tr>
</tbody>
</table>

Figure 6. The personnelDB

### (c) Relation emp (NAME, SALARY, DEPT)

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>αβ Simon</td>
<td>αβ 60K</td>
<td>αβ Toys</td>
</tr>
<tr>
<td>α Peter</td>
<td>α 40K</td>
<td>α Toys</td>
</tr>
<tr>
<td>λ Peter</td>
<td>λ 40K</td>
<td>λ Toys</td>
</tr>
<tr>
<td>λ Mary</td>
<td>λ 55K</td>
<td>λ Auto</td>
</tr>
<tr>
<td>λ Tom</td>
<td>λ 40K</td>
<td>λ PCs</td>
</tr>
</tbody>
</table>

Figure 7. The emp relation as seen by user α

### (d) Relation emp (NAME, SALARY, DEPT)

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>40K</td>
<td>Toys</td>
</tr>
<tr>
<td>Mary</td>
<td>55K</td>
<td>Auto</td>
</tr>
<tr>
<td>Tom</td>
<td>40K</td>
<td>PCs</td>
</tr>
</tbody>
</table>

Figure 8. The emp relation as seen by user λ
• We define $A \theta B$ to be an abbreviation of the boolean expression $[[A \theta B]] \neq \emptyset$.
• If $f$ and $g$ are boolean expressions then so are $f \lor g$, $f \land g$ and $\neg f$.

The select statement

The select statement of SQL is the most interesting one. It is similar to the select statement in SQL. The main difference is the addition of the “restricted to $\mu$” clause which allows the retrieval of a tuple to be restricted to the domain returned by the domain expression $\mu$. In SecSQL it has the following form.

```
select X: K
restricted_to $\mu$
from $r_1, r_2, \ldots, r_n$
where $f$
```

In the next few examples, we use $\text{OWN}$ to represent the level of the user. Therefore, $\text{OWN}$ will be substituted by the user (level) who submits the query. Also recall that $u^-$ denotes the set of all users below $u$ (including $u$).

Example 6. (For user $\alpha\beta$) Give managers of all employees in all object spaces.

```
select NAME, emp.DEPT, management.MANAGER : NAME
from emp, management
restricted_to $[[\text{emp.DEPT} = \text{management.DEPT}]]$
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>DEPT</th>
<th>MANAGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha\beta$ Jack</td>
<td>${\alpha\beta}$ Toys</td>
<td>$\alpha\beta$ Jack</td>
</tr>
<tr>
<td>$\alpha$ Simon</td>
<td>${\alpha\beta}$ Toys</td>
<td>$\alpha\beta$ Jack</td>
</tr>
<tr>
<td>$\alpha$ Peter</td>
<td>${\alpha, \lambda}$ Toys</td>
<td>$\alpha$ Peter</td>
</tr>
<tr>
<td>$\alpha$ Jack</td>
<td>${\alpha}$ Toys</td>
<td>$\alpha$ Peter</td>
</tr>
<tr>
<td>$\beta$ John</td>
<td>${\beta}$ PCs</td>
<td>$\beta$ John</td>
</tr>
<tr>
<td>$\lambda$ Peter</td>
<td>${\lambda}$ Toys</td>
<td>$\lambda$ Peter</td>
</tr>
<tr>
<td>$\lambda$ Tom</td>
<td>${\lambda}$ PCs</td>
<td>$\lambda$ Tom</td>
</tr>
</tbody>
</table>

Figure 9. Result of a natural join followed by a projection (See Example 6)

This is basically a natural join of emp and management, except that the attribute SALARY is not projected. The query retrieves what one would expect it to retrieve, and its user complexity is comparable to a classical query. The result of the query is shown in Figure 9. Note that the conservative choice for $[[A \theta B]], [[\text{emp.DEPT} = \text{management.DEPT}]]$ in this case (see 2nd bullet in Section 6) manages to keep the security spaces of all users partitioned yielding a satisfactory result.

Example 7. (For any user) Give managers of those employees known only to me.

```
select NAME, management.MANAGER
from emp, management
restricted_to $[[\text{emp.DEPT} = \text{management.DEPT}]]$
```
where \[
[[\text{NAME}]] = \text{OWN} : \{\text{OWN} \}
\]

**Example 8.** (For user \( \geq \alpha \)) *Give only the identities believed by \( \alpha \) of all objects in user \( \alpha \)'s object space.*

```sql
select *
restricted_to \( \alpha : \{ \alpha \} \)
from emp
```

**Example 9.** (For user \( \geq \alpha \)) *Give the identities of all objects in user \( \alpha \)'s object space.*

```sql
select *
restricted_to \( \alpha : \alpha ^{-} \)
from emp
```

**Example 10.** (For any user) *Give the identities of object “John” in my object space.*

```sql
select \text{NAME}
from emp
where \text{OWN} : \text{OWN}^{-} \subseteq [[\text{NAME}=\text{John}]]
```

**Example 11.** (For any user) *Give all objects in my object space which have the identity “John” at any level.*

```sql
select *
from emp
where \text{OWN} : \{ \text{OWN} \} \subseteq [[\text{NAME}]] \text{ and } [[\text{NAME}=\text{John}]] \neq \emptyset
```

**Example 12.** (For any user) *Give the security points at which the identity “John” exists in some object space.*

\[
[[\text{select} * \text{restricted}_to [[\text{name}=\text{John}]] \text{ from emp}]]
\]

**Example 13.** (For any user) *Give the identities of objects in the whole object spaces, restricting to the security points at which the identity “John” exists in some object space.* Note that the query of Example 12 is nested here.

```sql
select *
restricted_to [[select * restricted_to [[Name=John]] from emp]]
from emp
```

**Example 14.** (For any user) *Give the identities of the employees known only to my level.*

```sql
select *
from emp
where [[\text{NAME}]] = \text{OWN} : \{ \text{OWN} \}
```

**Example 15.** (For any user) *Give the identities of the employees known to every level.*

```sql
select *
from emp
where [[\text{NAME}]] = \text{OWN} : \text{OWN}^{-}
```

**Example 16.** (For any user \( \geq \alpha \)) *Give all objects, in my object space, of which the names
believed by level $\alpha$ and $\lambda$ are the same. Note that $|$NAME((OWN,$\alpha$))$|$ retrieves the name after stripping the security point (OWN,$\alpha$); $\cdot$ is a “read only operation” being used in a comparison here. In our model a value cannot be written after stripping its security point, except in a terminal situation such as printing a report for human consumption.

\[
\text{select} \quad * \\
\text{from} \quad \text{emp} \\
\text{where} \quad |\text{NAME((OWN,$\alpha$))}| = |\text{NAME((OWN,$\lambda$))}|
\]

4. Update mechanism

In our model, a user can update objects/beliefs only at its own level. An update has to preserve the belief consistency of a relation. After an update operation is performed, the transaction will be broadcast to all the levels above so that higher users may take appropriate actions. In SecSQL, update operations are insert, delete, and change.

In what follows we use the following notation. We assume that $r$ is a relation with $K_1K_2...K_m$ as its key. The key is also abbreviated as $K$ and an assignment of ordinary values to keys is abbreviated as $K=k$. Similarly $X=x$ denotes assignment to an arbitrary set $X$ of attributes of $r$.

4.1. Connect to clause and belief consistency

Some of our update operations include an optional “connect to \{u:(K=k)\}” clause. Its purpose is to allow lower object(s) $u$ to be connected. Note that the syntax allows only the key values of the lower object to be specified. Values of other attributes are determined automatically. This guarantees that the update operations preserve belief consistency. Note that the “connect to” clause is not available to the lowest level user.

4.2. Insert

When a user $u$ inserts an object, the user may also connect it to identities at lower levels. The system will first check if the new object exists at level $u$. If so, the user will get an error signal from the system; otherwise, the transaction will proceed. An insert update is of the form

\[
\text{insert} \quad (X=x) \quad \text{in} \quad r \\
\text{[connect to \{u:(K=k)\}]}
\]

Example 17. (For user $\alpha\beta$) Suppose that the user $\alpha\beta$ inserts the object “Mike” in the emp relation in Figure 6(a), associating the identity to “Peter” at level $\alpha$. The user can execute the following SecSQL statement and its effect is shown in Figure 10(a).

\[
\text{insert} \quad (\text{NAME}=\text{Mike}; \text{SALARY}=200K; \text{DEPT}=\text{Toys}) \quad \text{in} \quad \text{emp} \\
\text{connect to} \quad \alpha: \quad (\text{NAME}=\text{Peter})
\]

4.3. Delete

When a user $u$ deletes an object, the $u$-tuple for that object will be removed from the relation, and the identity of the object will be removed from each tuple in the relation. If a tuple becomes empty, it will be removed from the relation. The delete update is of the form

\[
\text{delete} \quad (K=k) \quad \text{from} \quad r
\]
<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>αβ Mike</td>
<td>{αβ}</td>
<td>Mike</td>
</tr>
<tr>
<td></td>
<td>{α}</td>
<td>Peter</td>
</tr>
<tr>
<td>αβ 200K</td>
<td>{αβ}</td>
<td>200K</td>
</tr>
<tr>
<td></td>
<td>{α}</td>
<td>40K</td>
</tr>
<tr>
<td>αβ Toys</td>
<td>{αβ}</td>
<td>Toys</td>
</tr>
<tr>
<td></td>
<td>{α}</td>
<td>Toys</td>
</tr>
</tbody>
</table>

Example 17: (User is αβ) insert (Mike; 200K; Toys) in emp connect to α:(NAME: Peter)
(a) one tuple is added

| αβ Jack | {αβ,α} | Jack |
| | {β} | John |
| | {λ} | Tom |
| αβ 100K | {αβ} | 100K |
| | {α} | 50K |
| | {β} | 80K |
| | {λ} | 40K |
| αβ Toys | {αβ,β} | Toys |
| | {α} | Shoes |
| | {β} | PCs |

Example 18: (User is β) delete (NAME: John) from emp
(b) First tuple is deleted, and the second tuple is modified to as in third tuple

| αβ Jack | {αβ,α} | Jack |
| | {β} | John |
| | {λ} | Tom |
| αβ 100K | {αβ} | 100K |
| | {α} | 50K |
| | {β} | 80K |
| | {λ} | 40K |
| αβ Toys | {αβ} | Toys |
| | {α} | Shoes |
| | {β} | PCs |

Example 19: (User is α) change (Jack) in emp to (John; 60K) connect to λ:(Tom)
(c) First tuple modified to as in second, and third tuple is modified to as in fourth

Figure 10. Update operations on the emp relation of Figure 6
Example 18. (For user β) Suppose that a user β deletes the object “John” from the emp relation in Figure 6(a). After the following delete statement is executed, and its effect is shown in Figure 10(b).

\[
\text{delete (NAME=John) from emp}
\]

4.4. Change

If a user u modifies the attribute values of an object, the system will check if the new key, if any, of the object already exists at level u. If so, the user will get an error signal from the system; otherwise, each tuple involving the identity will be updated. A change update is of the form

\[
\text{change (K=k) in r [to X=x])}
\]

\[
\text{[connect to \{u: (K'=k')\}]]}
\]

Example 19. (For user α) Suppose that the user α changes the NAME from “Jack” to “John” and the SALARY from 50K to 60K, and connects it to “Tom” at level λ, in the emp relation in Figure 6(a). The following SecSQL statement is executed and its effect is shown in Figure 10(c).

\[
\text{change (NAME=Jack) in emp to (NAME=John; SALARY=60K)}
\]

\[
\text{connect to \lambda: (NAME=Tom)}
\]

5. Comparisons with other works

In this section we compare our model with other models closely related to our relational model. First we compare our model SecDB with the relational model of [JS91]. Then we compare SecDB with the relational model of [SW92]. This comparison shows some pronounced differences between the two models, particularly due to the support of multilevel identity in our model.

The Jajodia-Sandhu [JS91] model

In [JS91] model, multilevel identity is not supported. In other words, the same object across different security levels must have the same apparent key. Although the problem of covert channel can be controlled by assigning a security classification to each field, it cannot model the objects with multilevel identity in the multilevel security environment. Secondly, they don't have as a clear belief-based semantics as in our model. For example, when a user u dominates the security levels of some fields in a tuple but not the others, then the system will replace the fields that the user has not clearance by nulls, using a filter function. However, the nulls may have two meanings. The user can think of them as the information that they are not allowed to read, or the information that is not available for some reasons. In [SW92], an improved belief-based multi-level security model is proposed and it is compared in the next section.

The MLS [SW92] model

Although logical and physical issues have been addressed, we only highlight the logical modeling here. In [SW92] model, a belief-based semantics is adopted: a user believes the contents of the database at its own level. An object is represented by several tuples, each at a different level. The schema for every relation R in the interpretation includes a 'Key classification' attribute KC, representing the security level of the information in K: R(K, KC, A₁,...,Aₘ). The domain of KC in a database at level u is the set of lattice levels dominated by u. Furthermore, K concatenated with KC functionally determines all the attributes of R within the database at level u. An addi-
tional attribute TC (tuple class) is also used to identity each tuple in R. To see how the same key can exist at different levels in their model, an example from [SW92] is shown in Figure 11(a), assuming the linearly ordered security level domain is (Secret (S), Classified (C), Unclassified (U)).

In Figure 11(a), the entity Enterprise U (the Enterprise believed in by level U) believed by level S is represented by two tuples. Level S agrees that Enterprise U (the Enterprise believed in by level U) exists, and agrees with level U regarding the starship's objective, but has a different belief about its destination. Note that the beliefs of U about the Enterprise U are seen by level C, but C does not believe that Enterprise U exists, since no tuple (with TC = C) belongs to Enterprise U. Thus, multilevel objects are supported only when their apparent keys are the same at all levels.

In our model, however, such information (level S’s belief about Enterprise U) can be captured in one tuple, as shown in Figure 11(c).

The MLR [CS95] model

In [CS95], Chen and Sandhu draw ideas from several works (including their own [JS91]) and give a multilevel relational data model (MLR). This model catalogues and imposes most constraints known in the security literature. These constraints become necessary in existing models (including [CS95]) because of the scheme and object fragmentation. In our model most of these constraints become redundant, or reduce to the classical model.

In our model an upper user can draw information from lower user. This has to be done manu-
ally by the upper user. [CS95] automates this process to a great extent by allowing an upper user to specify that a certain attribute value of a given object is to be drawn from the corresponding attribute value of some lower user. When a lower user updates such an attribute value, the corresponding value is automatically updated for the upper user. We illustrate how this feature may be incorporated in our model.

Figure 12 shows $\alpha\beta$’s belief about an object “Jack”. Note that salary of Jack for $\alpha\beta$ is entered as a “pointer” $\beta$. This pointer is highly expressive in the context of our model. First the system knows that the object known as Jack to $\alpha\beta$ is known as John to $\beta$. Therefore it substitutes John’s salary (80K) to Jack’s salary at level $\alpha\beta$. If $\beta$ updates Jack’s salary, John’s salary can be made to automatically update.

Note that the association between John and Jack as stated above is not possible in any existing model for multilevel security, including [CS95]. In existing models the key values “John” and “Jack” have to be the same to establish any connection between the beliefs at different levels. Thus the existing models do not support true polyinstantiation.

6. Features of our model and future work

We have presented a model and an SQL like language to query and update belief information for a multilevel security database. The model has been developed within the framework of our on going research in dimensional data [NG92], [CGN93], [GPN92], [CG94], [BG93], [Ga88], [GY88], [GV85], [BG90], [Ch95], [GN93]. Under the banner of dimensional data we have unified ordinary, temporal, spatial and belief data. Considerable research has already been done in dimensional databases. Following is a list of some of the features of dimensional databases which need to be studied to see how well these features continue to hold when security is added as a dimension under our approach.

- **Dimensional seamlessness and query reuse.** Our SQL like query languages have so far remained essentially independent of the choice of the dimensional data. The seamless integration of the different forms of dimensional data is achieved through the concept of dimension alignment, which allows a query on a form of dimensional data (e.g., a temporal query) to be used literally without any change in higher forms of dimensional data (e.g., spatio-temporal) [CGN93]. We expect this feature to extend to security data; thus e.g. one should be able to use a security query for temporal data in a security-spatio-temporal database without any changes.

- **Boolean seamlessness.** The constructs “or,” “and” and “not” of natural languages are incor-
porated symmetrically in our SQL. Such a seamlessness is not possible in existing models of dimensional data because in these models the dimensional domain such as security level is used as a column in a relation [GN93]. To extend this feature to security data, one needs to examine how dimensional elements, attribute values, tuples and relations in dimensional database when the security features are added as a dimension.

- **Fewer joins.** In our model the number of joins one needs are comparable to those in classical databases. In existing models of temporal, spatial and security databases the database representation of a real world object is often fragmented into a potentially unbounded number of fragments. This makes those query languages more difficult for users; before a query can be formulated, unnecessary joins have to be performed in order to paste different fragments together. In this context, the construct \([\mathcal{r}]\) of our model is very useful. It allows a relational expression to be turned into a domain expression, which can be nested in a relational or boolean expression. This construct is natural as well as powerful. It allows natural language selection to be realized as selections in our algebra [GN93]. The construct is also very efficient to implement.

- **Algebraic query optimization.** Because of fewer joins the queries in our algebra tend to be more efficient. But in addition, dimensional data give rise to interesting algebraic identities which provide a foundation for algebraic optimization [NG92]. This framework needs to be extended with the addition of security as a dimension.

- **Integrity of security information.** If the security level can be replaced by a different security level (e.g. replacing \(u_1\) John by \(u_2\) John), we would compromise the integrity of security information. This cannot happen in our model because a value is permanently glued to a security level, and the two cannot be separated. However, it can happen in existing models where security level is used as a column in a relation.

- **Object ids, type hierarchy and belief consistency.** In recent work we have extended our relation framework for dimensional data to incorporate type hierarchy and oids without sacrificing other features [CGN93,Ch95]. In case of security model, this has additional potential for us. It seems that with a careful use of oids, the concept of belief consistency can be achieved more naturally in the model.

- **Pattern matching languages for application development.** Our SQL like languages are interesting and fairly powerful, but they do not eliminate the need for embedding SQL in a lower level language such as C++. We have given powerful pattern matching languages for temporal and spatial databases [CG94]. The pattern matching languages are tailor designed to deal with linguistic peculiarities of different forms of dimensional data and substantially reduce the need for embedding SQL in C++. A pattern matching language can be viewed as a generalization of \([\mathcal{A}\theta\mathcal{B}]\) and be seamlessly integrated with SQL for dimensional data. Such a language needs to be developed for security data from scratch.

- **Incomplete information.** We have given a model for incomplete information for temporal databases [BG92]. We expect the model to extend to security all forms of dimensional data, including security data.

- **Compatibility with classical databases.** For classical user of a classical database, the dimensional space degenerates into a single point. In case of security data, this point is \(\lambda\). When there is only one point in a dimensional space, the construct \([\mathcal{A}\theta\mathcal{B}]\) reduces to \(\mathcal{A}\theta\mathcal{B}\) and the distinction between the restricted_to and where clauses disappears. Therefore, for such a user,
the restricted_to clauses can be suppressed, and a dimensional algebra can be made to look like classical algebra [BG93]. In this way we achieve the full compatibility with the classical databases.

7. Conclusion

The main motivation for imposing constraints in the existing security database models seem to be necessary because of schema and object fragmentation in these models. Our approach differs from the existing approaches in two important ways. First, our relational schemes are same as those in classical relational databases. In other words we do not add additional security related attributes such as “TC, tuple classification” to our relations. Second, the entire belief of a user is captured in a single tuple. The values are functions of time; functions by definition take a single value for a given user. Because of these reasons, most of the constraints appearing in the security literature are automatically satisfied in our model and yet others reduce to constraints on the classical relational model.

The major strength of our approach is the ease of query for a user, and in addition the fact that the user never has to worry about cooking up incorrect information, e.g. inadvertently splicing a belief of a user to a different user. We believe our model supports true polyinstantiation at the object level avoiding covert channel.

Our approach has lead to seamless integration of ordinary, spatial, temporal and multilevel security data. The prevailing folk wisdom in databases recognizes two important constructors in databases: tuple constructor and set constructor. We feel that there is much to be gained by adding the dimension constructor to this list. The various forms of seamlessness, uniformity and scalability of our SQL like language represent a promise to industry and users for a smooth transition from classical databases to temporal, spatial and security databases with oids and type hierarchy.

REFERENCES


[JS90] S. Jajodia and R. Sandhu. Polyinstantiation integrity in multilevel relations. Proceedings of...


**Additional References to our works related to this paper**


