Computers & Security, 11 (1992) 357–369



# Some Extensions to the Lattice Model for Computer Security*

Jie Wu, Eduardo B. Fernandez and Ruiguang Zhang†

*Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431, USA*

Some extensions to the lattice model for computer security control are considered in this paper. These extensions include: (1) a lattice model suitable for a system in which security classes can be dynamically created; (2) definitions of class-combining operators other than the least upper bound in dynamic systems; and (3) a join-semilattice and a partial order model which have fewer restrictions than the lattice model. Several possible implementations of the extended lattice model are also considered. Lattices are categorized by the dimensions of their partial orders. A dimension-two lattice implementation, based on the concept of *NTree*, is considered for both static and dynamic systems. The implementation for dynamic lattices proposed here proves to be efficient.

*Keywords:* Computer security, Lattice security model, Multilevel security.

## 1. Introduction

### 1.1. Motivation

With the increasing applications of computers in life-critical applications [1] (air-traffic control, robotics, etc.) there are more demands for the study of *computer security*, and there are many books [2–5] that discuss approaches for building secure systems. In general, *access control*, *information flow control*, *inference control*, and *cryptographic control* are four measures [2] commonly used to achieve secure systems. We consider only information flow control in this paper.

Information flow control aims at preventing security leakage as information flows through the computer system. A common model for information flow control is the *multilevel model* [6], which provides an object classification scheme together with a set of rules. The same set of levels is used for both subjects and objects. For objects, a level represents the measure of sensitivity of the data in the object and therefore the level is also called *security level*. For subjects, the level is called *clearance level* and represents the clearance or privilege of subjects. A subject can access an object if its clearance level is greater or equal to the object's security level. One common and useful application of the level concept arises from marking subjects and objects with categories, for example NATO, administrative, financial. The rules of this model guarantee that information will flow from one object to another only if the target object has the same or a higher

*This research was supported by a grant from the State of Florida High Technology and Industry Council.

†Now at Motorola Inc., 1500 NW 22nd Avenue, Boynton Beach, FL 33426-8292, USA.

0167-4048/92/$5.00 

classification level than the source. This multilevel model has been applied to several projects such as the Military Message Experiment (MME) and the Air Force Data Services Center (AFDSC) version of Multics [7]. A variation of the lattice model is presented in [8] where each subject can change its level between two bounds.

A general problem with the multilevel security model is that it is restrictive and some situations cannot be modeled by it. Figure 1 shows an example of a university file system where the department chair (a subject) has access rights for both professor files (objects) and student files (objects). The professors can inspect student files or receive data from them; that is, information can only flow in the directions indicated. This situation cannot be modeled by the multilevel model, since that model requires a total ordering. A more flexible model, allowing partial orderings of objects was proposed by D. Denning [9]. That model proved to be valuable to certify security at compile time [10] and to verify the security of system specifications [2].

### 1.2. Denning's Lattice Model

This generalized model is called the lattice model of security because its elements form a mathematical structure called a "lattice." The *lattice model* is defined by a tuple with five components, ⟨SS, OS, CS, *, →⟩, where:

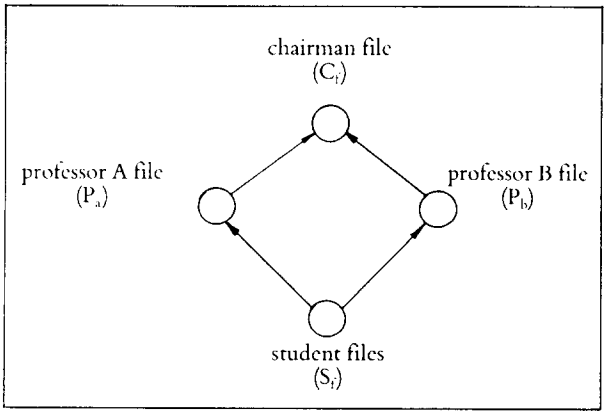


Fig. 1. University file system.

(1) SS is a set of subjects. These are the active agents capable of causing information flow.

(2) OS is a set of objects capable of storing information.

(3) CS is a set of security classes. Each object from the set OS is assigned a security class.

(4) * is a class-combining operator. For any operation involving two objects, the operator * specifies the class of the resulting object.

(5) → is a flow relation; this specifies the legal flow of information between any two security classes.

Furthermore ⟨CS, →, *⟩ constitutes a mathematical structure called a lattice, where CS is a finite partially ordered set with respect to the partial order defined by →. * is defined as a least upper bound (LUB or ⊕) operator on CS. $O^c_i$ represents the class to which object $O_i$ belongs. Sometimes we just call C the class for $O_i$ if it does not cause confusion. When two objects $O_i$ and $O_j$ from different classes are combined the resulting object should be put in $O^c_i * O^c_j$. Similarly to the multilevel model each subject is assigned a *clearance level* (or an upper bound). A subject can access an object if its clearance level is greater or equal to the object's security level. In the above university file system example, chairman, professors and students are subjects and their files are objects. Specifically, professor A can only access the student file and his own file.

In the following discussion, when considering the combination of two objects, we always mean two objects from different classes. Combining two objects from the same class is trivial, with the combined object being assigned to the same class.

Using the lattice model, the university file system example of Fig. 1 can be represented as:

object set OS = {department chair file,
professor A file, professor B file,
student file},

and flow relations:

professor A file → department chair file

professor B file → department chair file

student file → professor A file

student file → professor B file

So the lattice model extends the basic flow-control mechanisms of the multilevel model, and that model is therefore a special case of the lattice model where the object classes are totally ordered.

### 1.3. Why Should We Reconsider the Lattice Model?

A basic property in any lattice is the fact that for every pair of nodes (classes) there exists a greatest lower bound. This property, however, is not used for flow control, since when two objects from different classes are combined, the new object is always put in one of their common upper bound classes. An important question is the existence of models which are less restrictive, but which have similar mechanisms to control information flow without complicating the definition of class-combining operator.

Another question is whether the LUB should be used for the * operation. In the university file system example, if there are three professors then a data file constructed out of two professor files has the same security level as one constructed out of three professors files. The LUB cannot support a fine resolution of security class definition; also this approach causes the problem of *class overclassification* [10], that is, the data are put in a higher security class than they should be. There are some mechanisms to overcome this problem, but all of them are costly. This overclassification problem is especially undesirable in a system where security classes can be dynamically created, as we will discuss later.

An efficient implementation of the lattice model, a problem not discussed in the original paper [9], is also an important factor for the practical usefulness of the model. If a lattice is represented in terms of a partial order →, then its space complexity (number of flow relations for the worst case) is $O(C_n^2)$, where $n$ is the total number of objects and $C_n^2$ is the binomial coefficient. For checking the relation → between two classes some algorithm is needed to search these $O(C_n^2)$ rules. A better lattice representation is needed to avoid complex search algorithms.

### 1.4. Contributions of this Paper

Three possible extensions to the lattice model are considered here. One model is constructed by deleting the greatest lower bound requirement in the lattice. The resulting model is shown to be a type of *join-semilattice*. Another model is derived by replacing the LUB by another class-combining operator. This model is called the *partial-order model*. A further extension is to consider definitions of the class-combining operator other than the LUB in a dynamic system, where any level of security resolution can be enforced.

An implementation of the lattice model (or any partially ordered system), based on the work of Sandhu [11], is considered where lattices (or partial order systems) are categorized in terms of dimension, and each class is represented by a vector of integers. By this representation we not only reach a space complexity of $O(n)$ in describing the lattice but we also eliminate complex search algorithms. Lattices with dimension two are considered as examples in both static and dynamic systems. We also improve the algorithm for dynamic systems given in [11].

### 1.5. Organization of the Paper

In section 2 we discuss three possible extensions to the lattice model. In section 3 the improved implementation of the lattice model is considered, with an emphasis on dynamic systems. Finally section 4 presents some conclusions.

## 2. Some Extensions to the Lattice Model

### 2.1. A Join-Semilattice Model

As was mentioned earlier, the greatest lower bound property is not used in the original lattice flow

control model, since whenever two elements from different classes are combined only their LUB is used to find the class to which the result belongs. When this property (restriction) is deleted from the lattice model, another mathematical structure called a join-semilattice [12] is derived. In general a join-semilattice is a lattice with no least element requirement, which can be interpreted as a model with no public information. The difference between these two models is illustrated by the example in Fig. 2 (based on an example in [2]), where a non-lattice model is transformed into a lattice model and a join-semilattice model.

Transformation from a non-lattice model to a lattice model is performed in the following way: for every two objects $O_i$, $O_j$ that have upper bounds but no LUB, create a new class (named $(O_iO_j)^c$) which is a common upper bound for $O^c_i$ and $O^c_j$, or $O^c_i \rightarrow (O_iO_j)^c$, $O^c_j \rightarrow (O_iO_j)^c$; and for any upper bound of $O^c_i$ and $O^c_j$, U, the condition $(O_iO_j)^c \rightarrow U$ is satisfied. So we actually create a LUB for $O^c_i$ and $O^c_j$. If $O^c_i$ and $O^c_j$ are two elements in a loop, then all the elements in the loop are compressed into a single class in the lattice. Actually it is possible to take an arbitrary model and transform it into a lattice model by adding some new classes (therefore enhancing security resolution) or compress several existing classes (therefore reducing security resolution) [2]. The difference between the lattice model and the join-semilattice model is that the latter needs one fewer class (the class low in Fig. 2) than the former. In general, the semilattice model needs fewer precedences (two fewer precedences in this example) than the lattice model.

### 2.2. A Partial-Order Model

An even less restricted security model for information flow, a *partial-order model*, is proposed now. A class set CS is called a partial-order model if for each pair of classes in CS there at least exists one *minimal upper bound* (an upper bound $U_m$ is a minimal upper bound if there does not exist an upper bound U such that $U < U_m$). $C_i * C_j$ is considered as the minimal upper bound of $C_i$ and $C_j$ if there is no other upper bound $C_m$ with $C_i \leqslant C_m$ and $C_j \leqslant C_m$, such that $C_m \leqslant C_i * C_j$. An upper bound $U_1$ is a *least upper bound* if for any other upper bound U, $U_1 < U$. In general a least upper bound must be a minimal upper bound, but not vice versa. A system is closed with respect to an operation * if for any two elements $E_1$ and $E_2$ in the system, $E_1 * E_2$ is still an element in the system; otherwise, the system

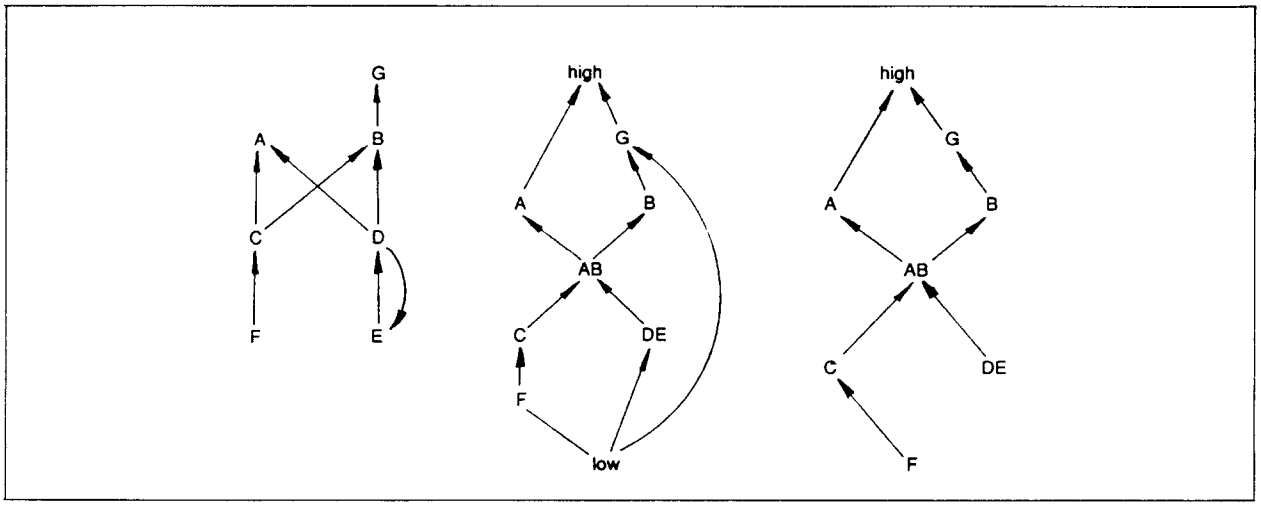


Fig. 2. (a) A non-lattice model. (b) Lattice model. (c) Join-semilattice.

is non-closed. In this work we do not separate the discussion for *closed systems* and *non-closed systems* (or open systems) under the operation *. In a static system, $C_i * C_j$ is not allowed if there are no minimal upper bounds in the class set, while in a dynamic system a new class is created for the result of such an operation. In Fig. 3, both A and B are minimal upper bounds of classes D and E, but not least upper bounds for D and E (actually there is no LUB for D and E).

In the partial-order model, object O in class C is represented by $O^{c}_{\{c1,c2,\ldots,cn\}}$, which means object O is in class C, and the set $\{C_1, C_2, \ldots, C_n\}$ is called the *object history* of object O, indicating that object O was constructed from objects in $C_1$, $C_2$, ..., and $C_n$. This representation does not show the order and the number of the compositions. This means that there may be several objects coming from the same class. Objects are defined as *simple* or *compound*. An object is compound if there is more than one class in its object history. A simple object is a special case of a compound object with the form $O^{c}_{c}$ (see Fig. 3). The class-combining operation is defined as:

$$O^{cv}_{\{cj1,cj2,\ldots,cjm\}} * O^{ct}_{\{ci1,ci2,\ldots,cim\}} = O_{cv}{}^{cu}$$

where

$$Cu = C_{j_1} * C_{j_2} * \ldots * C_{j_m} * C_i * C_{i_2} * \ldots * C_{i_n}$$

$$Cv = \{C_{j_1}\ C_{j_2}, \ldots, C_{j_m}\} \cup \{C_{i_1}, C_{i_2}, \ldots, C_{i_n}\}$$

There may exist several minimal upper bounds. If in Fig. 3 initially there are four simple objects in classes B, D, and E, then the class combining operation on $O_1$ and $O_3$ will create a new object $O_5$ in class A or class B. Let us assume it will be created on class A, that is:

$$O_3{}^{D}{}_{D} * O_1{}^{E}{}_{E} = O_5{}^{D*E}{}_{\{D\} \cup \{E\}} = O_5{}^{A}{}_{\{D,E\}}$$

An operation on $O_5$ in A and $O_4$ in B will create a new object $O_6$ which still remains in B. Since $O_4{}^{B}{}_{\{B\}} * O_5{}^{A}{}_{\{D,E\}} = O_6{}^{B} * D * E_{\{B\} \cup \{D,E\}} = O_6{}^{B}{}_{\{B,D,E\}}$. So this definition enforces the information flow policy without producing the overclassification problem. By using the partial-order model the example in Fig. 2 can then be transformed into the structure (for a closed system) of Fig. 4. The class High is not needed if a non-closed system is used.

Comparing the above two extensions with the lattice model for this example, the partial-order model is the closest to its original model, that is, the model which has the least changes in the security resolution of the original model. But the partial-order model requires a history for each object in the model. Though the number of security classes in general is static and it is relatively small, the number of objects is dynamic. So the overhead in keeping the object history may be significant when many class-combining operations are performed.

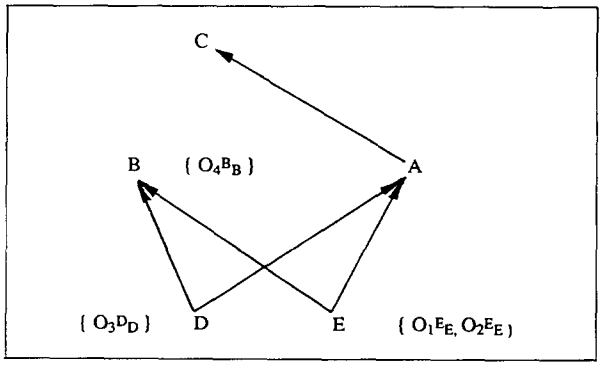


Fig. 3. A partial-order model.

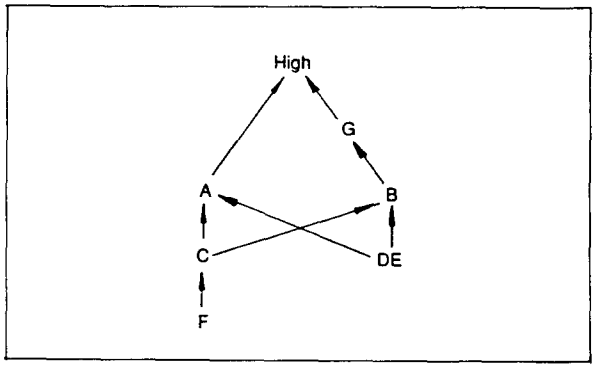


Fig. 4. A partial-order model for the example in Fig. 2.

### 2.3. Definition of Class-Combining Operator in a Dynamic System

We will consider an extension of the existing lattice model in another direction: considering other possible definitions of the class-combining operator (*). If $O_i$, $O_j$ are two objects, $O^c_i * O^c_j$ is the class where a newly created object is assigned. In a static system there are five possible definitions for

$$O^c_i * O^c_j = \begin{cases} O^c_i \\ O^c_j \\ O^c_i * O^c_j = O^c_i \oplus O^c_j \\ \text{other upper bounds} \\ \text{others} \end{cases}$$

$O^c_i$, $O^c_j$ and "others" cannot be selected because they would violate the information flow policy, although in some situations an information flow policy violation does not necessarily cause a security violation. For example, a combination of a SECRET paragraph from a TOP-SECRET file and a SECRET file should generate a SECRET file. To incorporate the ability to do this, a class of trusted subjects together with a set of appropriate mechanisms should be included in the system. However, the presence of such mechanisms (such as the one used in SIGMA [13]) makes it difficult to determine the actual security policy enforced by the system and complicates the user interface. Other upper bounds are possible but this choice will cause overclassification of data. Therefore $O^c_i \oplus O^c_j$ is the only suitable definition for $O^c_i * O^c_j$ in static systems.

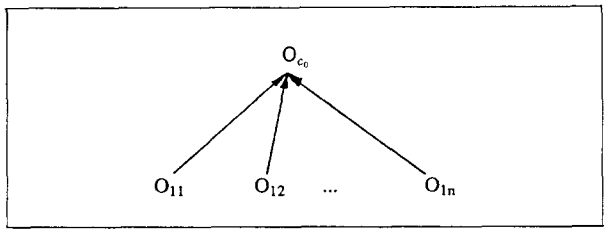


Fig. 5. Low resolution in the combination of objects.

The problem with the definition of $O^c_i * O^c_j = O^c_i \oplus O^c_j$ is that it cannot provide fine security resolution for some systems. Figure 5 shows a case where a combination of two objects from two classes belongs to the same security class of an object created by combining $n$ objects from $n$ different classes, that is:

$$O^c_{1i} \oplus O^c_{1j} = O^c_{11} \oplus O^c_{12} \oplus \ldots \oplus O^c_{1n} = O^c_{c_0}$$

$$(1 \leqslant i \leqslant j \leqslant n)$$

This situation is especially undesirable in systems where new security classes can be created dynamically. The following class-combining operator definition is based on the policy of creating a new security class for each newly created object.

$$O^c_i * O^c_j = \begin{cases} (O_i O_j)^c & \text{if } O^c_i, O^c_j \text{ have no relation} \\ \text{MAX}\{O^c_i, O^c_j\} & \text{if } O^c_i, O^c_j \text{ are related} \end{cases}$$

where $(O_i O_j)^c$ is a newly created class and three rules are added in the lattice:

$$O^c_i \rightarrow (O_i O_j)^c$$

$$O^c_j \rightarrow (O_i O_j)^c$$

$$(O_i O_j)^c \rightarrow O^c_i \oplus O^c_j$$

and MAX represents an operation that selects the highest security class of the two classes. This process is illustrated in Fig. 6 where $O^c_i$ and $O^c_j$ are not related.

The drawback of this approach is the number of newly created object classes. A system that begins with $n$ objects will end up with $2^n$ objects solely by the class-combining operator (not including the

insertion of new classes from the outside). A trade-off is needed to balance the degree of security resolution and the number of security classes. Another policy is possible in which new classes are added only when they are required for some security reasons, otherwise no new classes are added, so that the LUB is still used as * operation.

### 2.4. Comparison of the Different Models

Table 1 shows a summary of different security models by listing their advantages, disadvantages, and suitable applications.

This table can be used to select the most suitable model for a given application. Models can also be combined based on application needs; for example, the dynamic class generation for combined objects can be used together with the partial order model. On the other hand, several models can be used in a single application when necessary.

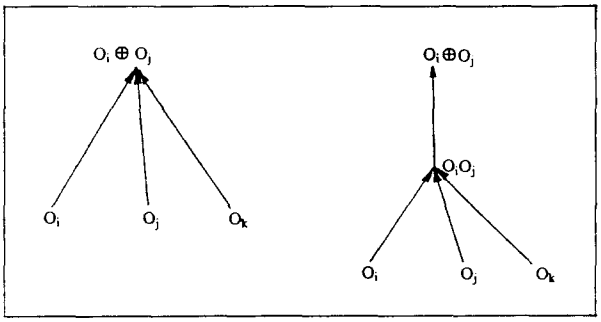


Fig. 6. Creating a new security class.

## 3. Efficient Implementation of the Lattice Model

We present now some implementation ideas for the lattice model. Most of these ideas are related only to the structure of the partial order and therefore are also useful to implement lattices or partial orders that describe other models; for example, Sandhu's lattices [11] describe structuring of user or resource groups with respect to access control (not information flow).

### 3.1. Classification of the Lattice by Dimension

A set of vectors can be used to represent a lattice model [11]. For each class $C_i$ in a particular lattice model, a vector $(V_{i_1}, V_{i_2}, \ldots, V_{i_n})$ is assigned, where the $V_{i_j} (1 \leqslant j \leqslant n)$ are integers, and $C_i$ is of a higher order than $C_j$ if and only if $V_{i_k} \leqslant V_{j_k}$ for all $k$ from 1 to $n$. This representation can also be used to describe the extended models proposed here, that is, the semilattice and partial-order models. This set of vectors is called a *realizer* with a size of $n$. The *dimension* [14] of a lattice model is the size of its smallest realizer.

TABLE 1. Comparison of the different models

| | Advantages | Disadvantages | Suitable applications |
|---|---|---|---|
| Lattice model | Well-defined model | No support for fine security resolution and strict restriction on the application structure | Static systems |
| Join-semilattice model | Less restriction on application structures than the lattice model | No support for fine security resolution | Static systems |
| Partial-order model | Least restriction on application structures | Overhead in keeping object history | Static systems that require the application structure to remain relatively fixed |
| Dynamic class generation for combined objects | Provides any degree of security resolution | Too many classes generated dynamically | Dynamic systems where fine resolution is the objective |

Figure 7 shows some lattice models for different dimensions, for example dimension 1(a), and dimension 3 (b). The example shown in Fig. 1 is a lattice model with dimension 2. A model with dimension 2 is called *Ntree* in [11]. The Ntree representation greatly reduces storage requirements and simplifies the algorithm for determining the ordering of two classes.

### 3.2. A Static Lattice Implementation

The vector assignment is straightforward for static systems [11]. One possible assignment strategy for a lattice L (with dimension $n$) is as follows:

(1) Determine all the total orders for L (through the procedure of topological sorting); these total orders are labeled as $L_i$.

(2) Select a realization R of L in which $R=\{L_1, L_2, \ldots, L_n\}$, and $L_1 \cap L_2 \cap \ldots \cap L_n = L$, $n$ is the dimension of P, and $L_1 \cap L_2 \cap \ldots \cap L_n$ is defined as the set of ordered pairs $\{(u, v) | (u, v) \in L_1 \wedge (u, v) \in L_2 \wedge \ldots \wedge (u, v) \in L_n\}$.

(3) Assign a vector $(V_{i_1}, V_{i_2}, \ldots, V_{i_n})$ to each security class $C_i$ in the lattice, in which $V_{ij}$ $(1 \leq j \leq n)$ is the position of $C_i$ in the total order $L_j$.

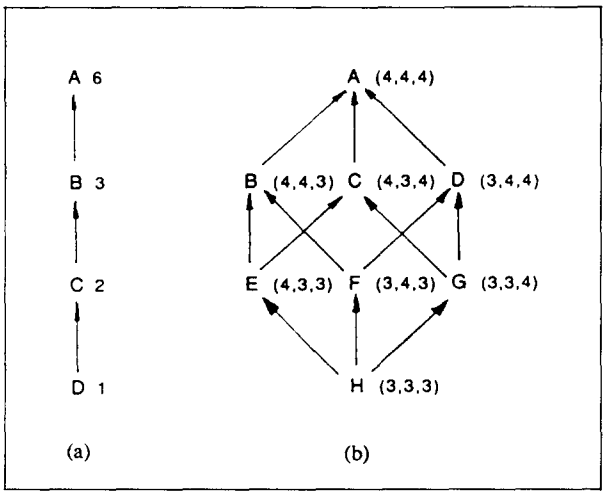


Fig. 7. Lattice models with different dimensions.

This vector assignment is used when the lattice dimension can be predetermined. In Fig. 1, we first determine all the total orders: $L_1 = C_f P_a P_b S_f$, $L_2 = C_f P_b P_a S_f$. Since $L_1 \cap L_2 = \{(C_f, P_b), (C_f, P_a), (P_b, S_f), (P_a, S_f), (C_f, S_f)\}$ = the lattice, then $R = \{L_1, L_2\}$ is a realizer based on step 3 of the algorithm above. (1, 1) is assigned to C, (2, 3) to $P_a$, (3, 2) to $P_b$, (4, 4) to S. Figure 7 shows two graphs with different dimensions (Fig. 7(a) with dimension 1 and Fig. 7(b) with dimension 3).

If the dimension cannot be defined a priori, it has been proved [15] that there is no way to determine the dimension without enumerating all possible realizers, and this is very costly. One simple method is to select the first realizer for the vector assignment, but the number of components in a vector is in general greater than the lattice dimension. In other words, we trade off memory for fast vector assignment.

A criterion for judging a method for lattice implementation is the simplicity of the operator * implementation. The Ntree representation supports a relatively simple*(or LUB) operation. The vector for each class is stored in an array, say A. When two objects from two different classes (with index $i$ and $j$, respectively) are combined, it is required to find the class to which the resulting object belongs. Algorithms 1 and 2 (written in pseudo Pascal) show the implementation of the * operation in both closed and non-closed systems.

Algorithms 1 and 2 always search up starting from index max($i$,$j$). Vectors are stored in one of the topological sorting orders to guarantee that the first upper bound (of A[$i$] and A[$j$]) is the LUB.

*Algorithm 1*: For a closed system it finds the address of the class to which the combination of two objects from A[$i$] and A[$j$] belongs.

```
procedure FIND_LUB_IN_CLOSED_SYSTEM (i, j) return LUB;
  ( s: = max(i, j);                    { s is the starting search position}
    Find : = false;
    while not Find do
        ( if A[s] ≥ A[i] and A[s] ≥ A[j]
          then  ( Find: = true;
                  LUB: = s;
                );
          s: = s + 1;
        )
  )
```

*Algorithm 2*: For a non-closed system it finds the position of the class to which the combination of two objects from A[*i*] and A[*j*] belongs.

```
procedure FIND_LUB_IN_NON_CLOSED_SYSTEM (i, j) return LUB;
  ( s: = max(i, j);                    {s is the starting search position}
    Find : = false;
    while not Find and _s ≤ End_Of_Array do
        (  if A[s] ≥ A[i] and A[s] ≥ A[j]
           then ( Find: = true;
                  LUB: = s;
        );
           s: = s + 1;
        );
    If not Find then return Combination_Is_Not_Allowed
  )
```

### 3.3. A Dynamic Lattice Implementation

It is possible that new security classes be added dynamically. This can be achieved through a dynamic refinement process [11]. The refinement replaces an element in a lattice by a sublattice. Figure 8 shows an example of such a dynamic refinement.

We will consider here the conservation refinement, where the dimension of the lattice is not changed throughout the refinements (the lattices of Fig. 8(a), (b) have the same dimension). The elements to be replaced are called *exploded group* (e.g., B in Fig. 8), while the rest of the elements are called *unexploded group*.

A good vector assignment for dynamic assignment should keep constant the vectors for unexploded groups during the refinement process. One assignment which has this property is the *Quota offset numbering* [11], which assumes that the total number of new classes to be added is predetermined. A quota $q[C]$ that represents the total number of new classes (these are the new classes that will replace C) is assigned to class C.

Suppose that R = {L1, L2} is a realizer for a two dimensional lattice, where:

$$L_1 = C_1, C_2, \ldots, C_m$$

$$L_2 = C_1', C_2', \ldots, C_m'$$

The positions of a class $C_i$ in $L_1$ and $L_2$ correspond to the values of elements in the vector associated with $C_i$. In quota offset numbering the positions of $C_1$ and $C_1'$ are still assigned the value 1, and the position of $C_i(C_i')$ is defined as:

$$p[C_i(C_i')] = p[C_{i-1}(C_{i-1}')] + q[C_{i-1}(C_{i-1}')]$$

The main disadvantage of quota offset numbering is that it is sometimes impossible to predetermine the quotas, because one needs to know not only the total number of elements created dynamically, but also the distribution of these elements.

TABLE 2. Quota offset numbering for Fig. 8(a)

| | $q[C]$ | $p[L_1]$ | $p[L_2]$ | | Vector |
|---|---|---|---|---|---|
| A | 10 | 1 | 1 | A | (1, 1) |
| B | 2 | 11 | 31 | B | (11, 31) |
| C | 20 | 13 | 11 | C | (13, 11) |
| D | 5 | 33 | 33 | D | (33, 33) |

$L_1$ = ABCD, $L_2$ = ACBD.

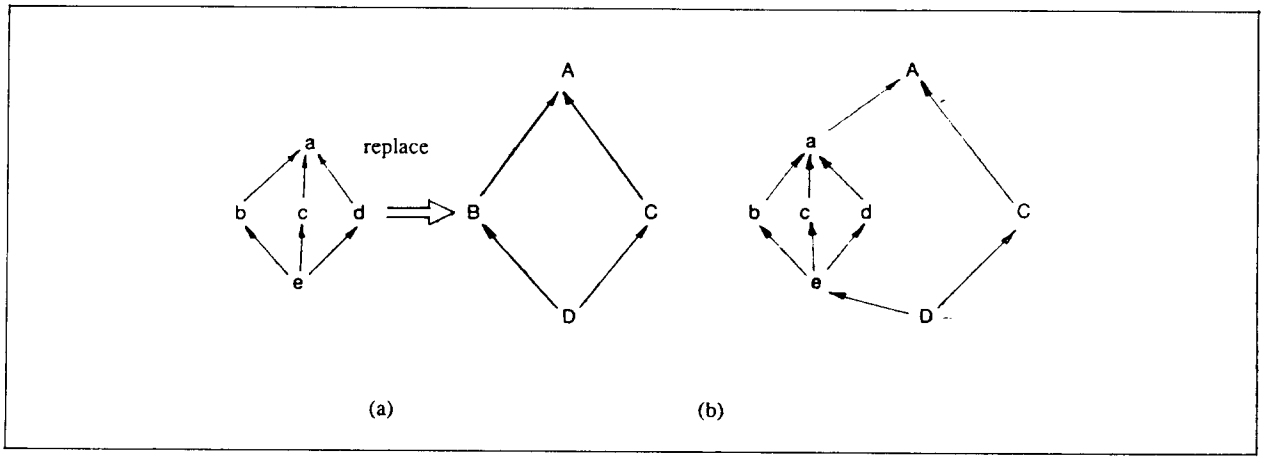


Fig. 8. A dynamic refinement: (a) before refinement; (b) after refinement.

We propose here a much simpler vector assignment for dynamic refinement. In this method, we do not need any a priori knowledge of the total number of elements created dynamically. The only changes in the vectors are that values are defined to be positive real numbers instead of positive integers as before. The initial assignment is exactly the same as the vector assignment in the static lattice, or the $i$th component of the vector assigned for class C is the position of C in the total order $L_i$. Suppose this position number is $P_c$, and we have a refinement to replace C by a sublattice with $n$ elements. Since $P_c - 1$ and $P_c + 1$ have been used for other classes, these expanding elements should be assigned to real numbers within $(P_c - 1, P_c + 1)$ (to keep the relationship between the exploded and unexploded groups). Considering that the classes with position $P_c - 1$ and $P_c + 1$ might also be exploded groups, we restrict the range from $(P_c - 1).6$ to $P_c.6$ (exclusive of $P_c.6$) or $[(P_c - 1).6, P_c.6)$ (note that the range $((P_c - 1).5, Pc.5]$ is also possible). In order to sparsely assign these $n$ classes across this region (this is important to reduce number of bits to represent a real number since each class can be further refined), the first ten elements should be assigned to $(P_c - 1).6$, $(P_c - 1).7$, $(P_c - 1).8$, $(P_c - 1).9$, $P_c$, $P_c.1$, $P_c.2$, $P_c.3$, $P_c.4$, $P_c.5$. When $n$ is greater than 10, one more digit is added which can represent one hundred more elements. Further refinements are also possible in these newly created elements. If a refinement (with $n$ elements in the sublattice) is applied to an element with a vector $(a_1\ a_2 \ldots a_m, a_1'a_2' \ldots a_m')$, the $n$ new vector assignments should be selected from $[a_1a_2 \ldots (a_m - 1).6, a_1a_2 \ldots a_m.6)$ for the first vector component and $[a_1'a_2' \ldots (a_m' - 1).6, a_1'a_2' \ldots a_m'.6)$ for the second vector component. For simplicity, the selection for both components should be the same in the sense that they have the same set of newly added digits. For example in Fig. 8:

$L_1$ = ABCD $L_2$ = ACBD

We have a refinement (with five elements in the sublattice) for B with vector (2, 3). The sets we select are (1.7, 1.9, 2.1, 2.3, 2.5) for first element (2), where (7, 9, 1, 3, 5) is a set of newly added digits. So for the second component, we have a set (2.7, 2.9, 3.1, 3.3, 3.5). The actual assignment is also based on the structure of the sublattice. Since we have a realizer:

$L_1$ = abced

$L_2$ = adcbe

Then

a = (1.7, 2.7) b = (1.9, 3.3) c = (2.1, 3.1)

d = (2.3, 2.9) e = (2.5, 3.5)

For the unexploded group we have A = (1, 1) C = (3, 2) D = (4, 4).

This idea can be easily incorporated at the coding stage. Suppose a byte is the smallest addressable unit. Initially each element in a vector is implemented by several bytes (the number of bytes used depends on the number of classes). When a refinement is initialized $(2^8 \times i + j$ elements), $(i + 1)$ more bytes are used which are appended to the element to be refined. These bytes are separated from the initial bytes by a decimal point. In the $(i + 1)$ byte after the decimal point, only the first $\log_2 j$ most significant bits are used. The extra $(8\text{-}\log_2 i)$ bits can be used for further refinement.

For Example 8, B is now represented as (00000010, 00000011). Since the refinement to B is with five elements, the first $\log_2 5 = 3$ bits of the newly added byte will be used (the rest of the bits can be considered as 0s). One possible selection can be:

{00000000, 00100000, 01000000, 10000000, 11000000}

or {00, 20, 40, 80, c0} in hexadecimal notation,

so a = (02.00, 03.00)

b = (02.20, 03.80)

c = (02.40, 03.40)

d = (02.c0, 03.20)

e = (02.80, 03.c0)

This type of coding can also be applied to *Conservative quota offset numbering* [11], where a different type of refinement is considered.

## 4. Conclusions

Some extensions to the lattice model for multilevel computer security are considered in this paper. A class-combining operator other than LUB is considered for dynamic systems. It is shown that this definition gives a finer precision in class organization. Two models for multilevel security systems are proposed: join-semilattice and partial-order. Both models are less restrictive than the lattice model. The join-semilattice model slightly extends the lattice model by deleting the GLB requirement. The partial-order lattice, covering all the loop-free systems, is a much more general model, although its data-combining operations are more complicated than the ones in the lattice model. With these models, there is more flexibility to select a security model suitable for different applications. Some possible lattice and partial-order implementations are considered for both static and dynamic systems based on the Ntree concept. An efficient implementation for dynamic lattices and partial orders has also been proposed. These models and implementations can be of value for flow analyzers such as FDM, Gypsy, or SRM [4]. As indicated in [16] all of the currently used mandatory security models can be described by lattices.

## Acknowledgment

Haiyan Song provided valuable criticism of this paper and her contribution is gratefully acknowledged.

## References

[1] P. G. Neumann, On hierarchical design of computer systems for critical applications, *IEEE Trans. on Softw. Eng., SE-12*, 9 (September 1986) 905–920.

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

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

[4] M. Gasser, *Building a Secure Computer System*, Van Nostrand Reinhold, New York, 1988.

[5] C. P. Pfleeger, *Security in Computing*, Prentice-Hall, Englewood Cliffs, NJ, 1989.

[6] D. E. Bell and L. J. LaPadula, Secure computer systems: unified exposition and Multics interpretation. *Rep. ESD-TR-75-306*, Mitre Corp., Bedford, MA, March 1976.

[7] C. E. Landwehr, C. L. Heitmeyer and J. McLean, A security model for military message systems, *ACM Trans. Computer Systems*, 2 (August 1984) 198–222.

[8] S. N. Foley, A model for secure information flow, *Proc. 1989 IEEE Symp. on Security and Privacy*, IEEE Computer Society Press, Washington, 1989, pp. 248–259.

[9] D. E. Denning, A lattice model of secure information flow, *Commun. ACM*, 19 (May 1976) 236–243.

[10] D. E. Denning, Certification of programs for secure information flow, *Commun. ACM*, 20 (July 1977) 504–513.

[11] R. S. Sandhu, The NTree: a two dimension partial order for protection groups, *ACM Trans. Computer Systems*, 6 (May 1988) 196–220.

[12] J. L. Mott et al., *Discrete Mathematics for Computer Scientists and Mathematicians*, Prentice-Hall, Englewood Cliffs, NJ, 1986.

[13] S. R. Ames and D. R. Oestereicher, Design of a message processing system for a multilevel secure environment, *Proc. AFIPS 1978 National Computer Conf.*, 47 (June 1978) 765–771.

[14] B. Dushnik and E. W. Miller, Partially ordered sets, *J. Math.*, 63 (1941) 600–610.

[15] M. C. Golumic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[16] J. K. Millen, Models of multilevel computer security. In *Advances in Computers*, Vol. 29, Academic Press, New York, 1989, 1–45.

**Jie Wu** received a B.S. degree in computer engineering in 1982, an M.S. in computer science in 1985, both from Shanghai University of Science and Technology, Shanghai, People's Republic of China, and a Ph.D. in computer engineering from Florida Atlantic University, Boca Raton, Florida, in 1989.

During 1985–1987, he taught at Shanghai University of Science and Technology. Since August 1989, he has been a member of the faculty in the Department of Computer Science and Engineering, Florida Atlantic University. He has authored over 30 technical papers in the areas of fault-tolerant computing, distributed algorithms, interconnection networks, Petri net theory, and automatic programming. His current research interests include parallel and distributed processing, fault-tolerant computing, and computer security.

Dr. Wu is a member of the IEEE and Upsilon Pi Epsilon.

**Ruiguang Zhang** received a B.S. in electrical engineering from Shanghai University of Science and Technology, Shanghai, People's Republic of China in 1985, and an M.S. in computer engineering from Florida Atlantic University, Boca Raton, Florida, in 1989.

During 1985–1987, she worked at the Shanghai Railway Institute of Science and Technology, as an assistant engineer. She was with the United States Control Systems Corp., West Palm Beach, Florida between 1989 and 1991. She is currently a software engineer in the Paging Products Group at Motorola Inc., Boynton Beach, Florida.

**Eduardo B. Fernandez** is a professor of the Department of Computer Science and Engineering at Florida Atlantic University in Boca Raton, Florida. He has worked at the NASA Satellite Tracking Station, Santiago, Chile, taught at the University of Chile and the University of Miami, and researched the security and performance aspects of database systems at the Los Angeles Scientific Center of IBM. His current research interests are in fault-tolerant systems, computer security, database systems, and computer architecture.

Fernandez holds an M.S. degree in electrical engineering from Purdue University and a Ph.D. in computer science from UCLA. He has authored three books and a large number of papers. He is a senior member of IEEE.