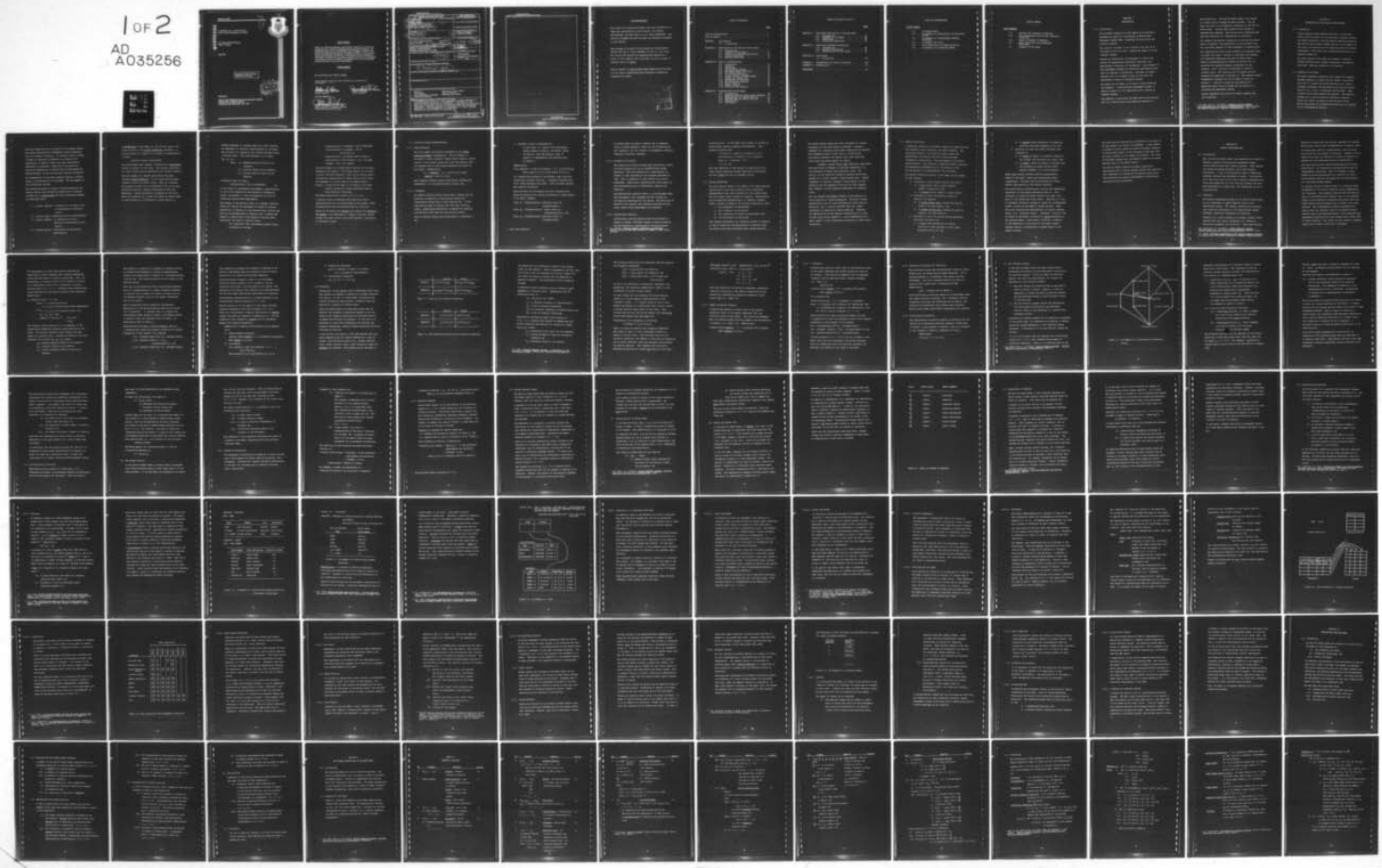
AD-A035 256 CANADIAN COMMERCIAL CORP OTTAWA (ONTARIO) F/G 9/2
A MODEL OF A PROTECTED DATA MANAGEMENT SYSTEM.(U)
JUN 76 M J GROHN F19628-76-C-0025
UNCLASSIFIED ESD-TR-76-289 NL

1 OF 2
AD
A035256



ESD-TR-76-289

A MODEL OF A PROTECTED DATA MANAGEMENT SYSTEM

I.P. Sharp Associates Limited
Ottawa, Canada

June 1976

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
HANSCOM AIR FORCE BASE, MA 01731

LEGAL NOTICE

When U. S. Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

OTHER NOTICES

Do not return this copy. Retain or destroy.

"This technical report has been reviewed and is approved for publication."

William R. Price

WILLIAM R. PRICE, Capt, USAF
Project Engineer

Donald P. Eriksen

DONALD P. ERIKSEN
Project Engineer

FOR THE COMMANDER

Stanley P. Dereska

STANLEY P. DERESKA, Colonel, USAF
Deputy Director, Information Systems
Technology Applications Office

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: ESD-TR-76-289	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): A MODEL OF A PROTECTED DATA MANAGEMENT SYSTEM.	5. TYPE OF REPORT & PERIOD COVERED
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Michael J. Grohn	8. CONTRACT OR GRANT NUMBER(s): F19628-76-C-0025
9. PERFORMING ORGANIZATION NAME AND ADDRESS: I.P. Sharp Associates Limited, Ottawa, Canada (Canadian Commercial Corp., Ottawa, Ontario) 387327	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: PE 62702F Project 2801
11. CONTROLLING OFFICE NAME AND ADDRESS: Deputy for Command and Management Systems, Electronic Systems Division, Hanscom Air Force Base, MA 01731	12. REPORT DATE: June 1976 / 13. NUMBER OF PAGES: 118
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): 121 p.	15. SECURITY CLASS. (of this report): UNCLASSIFIED / 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release; Distribution Unlimited.

Technical rept.,

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Security
Protection
Data Management Systems
Reference Monitor
Relational Data Bases
Bell-La Padula
Mathematical Model

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A mathematical model of a data management system embodying a military security policy is presented. Its basis in the Bell-La Padula model and the extensions required are narratively described. General notions of protection and the relational approach to data management are considered. A full, formal description of the model is included.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED 387 327

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ACKNOWLEDGEMENTS

This report has benefitted greatly from the contribution of ideas and explanations by David Bonyun, Gill Kirkby, Ted McDorman, and Shan Mitra of I.P. Sharp Associates. Lee Schiller of MITRE provided valuable and perceptive guidance to our efforts.

The revision of a draft of this report was stimulated by Captain William R. Price (ESD/MCI) of the U.S. Air Force. The clarity and accuracy of explanations improved as a result of his comments and criticisms, as well as those of Jonathan Millen of MITRE.

Marvin Schaefer of System Development Corporation provided us with useful explanations and interesting suggestions regarding our model.

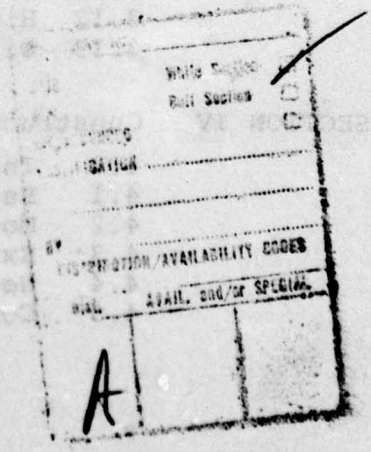


Table of Contents

Table of Contents (cont'd)

LIST OF ILLUSTRATIONS

Figure Number

LIST OF TABLES

Table Numbers

SECTION I

INTRODUCTION

1.0 Introduction

The principal objective of this report is to describe a mathematical model of a generalized, protected data management system (DMS), consisting of both proven and unproven programs.

The scope of the model is not limited to be that of an "end-user's" point of view. Instead the model is of the "overall" system.

Notions of "protection" are developed to involve the control of unauthorized observation (security) and modification (integrity) of information. Key elements of data management systems are identified and modelled when they are relevant to protection. The model is highly abstract, yet it is aimed to serve as the basis for engineering a family of protected systems.

The model will be shown to be valid in three differing environments: a dedicated data management system; a system in which it is an application and a network of computer systems.

The approach to developing the model was to use existing work as a starting point and modify and extend to a

generalized form. The Bell-La Padula model[1] was adopted as a basis since it seemed the most suitable. The new model was built on the essential components of the Bell-La Padula model. Certain model elements were changed or extended when required. New entities were introduced when they were essential for completeness or generality.

For example, the Multics orientation of the Bell-La Padula model is removed. The consistency of the new model with the relational approach to data management is demonstrated.

The organization of this report reflects the approach taken to produce the model. Section II describes the concepts and techniques underlying the Bell-La Padula model. Special considerations are covered in section III, to establish the required changes, extensions and new elements. Here, they will be described in terms of informal prose. The finalized set of changes and new elements are summarized in section IV. The complete formal mathematical treatment of the new model is given in section V. Section VI gives the resulting set of assertions which serve as axioms for the design of a protected data management system.

Several appendices help make this report complete and self-contained.

[1] D.E. Bell and L.J. La Padula, Computer Security Model: Unified Exposition and Multics Interpretation, ESD-TR-75-306, The MITRE Corporation, Bedford, Massachusetts, June, 1975.

SECTION II

DESCRIPTION OF THE BELL-LA PADULA MODEL

2.0 Introduction

Secure computer system modeling has been a continuing effort since the early 70's at Electronic Systems Division, the MITRE Corporation, and Case Western Reserve University. The Bell-La Padula model (MITRE) was one of the results of this work, and serves well as a starting point for further development. This model will be described narratively to enhance readability.

The major facets of the model are elements, invariant system characteristics, general mechanisms, and specific solutions. They are treated in this section.

2.1 Elements of the Model

The model represents abstractly the elements of computer systems necessary to study security issues. The active entities in a system are called subjects (e.g., users, programs, processes) and the passive entities are called objects (e.g., files, I/O devices, source code). Some entities can be both subjects and objects. The security problem is to control access of subjects to objects, where this control is based on some security policy. Mainly the control of unauthorized observation of information is addressed by security.

Security threats can be of a direct or an indirect nature. The direct observation of information without authorization is a breach of security. An indirect security threat is posed if classified information is allowed to be manipulated, or moved in such a way that unauthorized observation is possible. These can be controlled by forcing an observation to satisfy static requirements for appropriateness, as well as requiring explicit permission.

The presentation of the model elements requires a certain amount of terminology to be developed. The development of this terminology follows.

The generalized modes of access are distinguished by the varying effects that different accesses have on objects. A useful set of access modes and their identifying symbols are described below:

1) e access (execute - invocation of an object, with no observation or modification);
2) r access (read - observation with no modification);
3) a access (append - no observation and non-destructive modification of an object);
4) w access (write - observation and destructive modification).

A system state in the model is a set of four values, the first of which is the current access set, denoted b. A current access by a subject to an object is represented by a triple:

(subject, object, access-mode).

The triple means that "subject" currently has "access-mode" access to "object" in the state. The current access set b is a set of such triples representing all current accesses.

The next element of a system state within the model involves access permission. It will allow discretionary (dynamic) control of object access. That is, a subject may access an object only if permission to do so has been explicitly given by a subject authorized to give this permission. Access permission is modeled as a matrix M, where the component M_{ij} records the subset of access modes in which subject S_i is permitted to access object O_j.

Another component of a system state is a level function, the embodiment of security classifications in the model. This function (f) assigns to each subject and each object a security level. The level function F is a triple (f_s, f_o, f_c),

where: f_s = maximum security levels of all subjects;

f_o = security levels of all objects;

f_c = current security levels of the subjects.

A security level is a pair:

(classification, set of categories).

In a military or governmental environment, classification is one of the ordered set {unclassified, confidential, secret, top secret}. A set of categories is a set of formalized need-to-know compartments.

The purpose of the security level is to embody a security policy by allowing the assignment of levels to restrict observation of objects in a non-discretionary (static) way. This can be accomplished by requiring that a subject may observe an object only if the subject's security level "dominates" the object's security level.

The "dominates" ($\geq$) relationship between levels is defined as follows:

(classification 1, category - set 1) dominates

(classification 2, category - set 2)

if and only if

classification 1 is greater than or equal to classification 2 AND category - set 1 includes category - set 2 as a subset.

The last element of a system state concerns the structure imposed on the objects. The access control of an object is built into this structure. Bell and La Padula chose a hierarchical organization, H, where access to any object was dependent in certain ways on its parent object. Finally, a state of the model is a 4-tuple of the form:

(current access set, access permission matrix, level function, hierarchy).

The model notation for a state is (b, M, f, H).

A relation W must be defined, which specifies which characteristics are to be maintained by the system. These characteristics are collectively known as "security".

Inputs to the system are <u>requests</u> and outputs are <u>decisions</u>. The <u>system</u> is all sequences of (request, decision, state) triples with some initial state which satisfy a relation W on successive states.

2.2 Invariant System Characteristics

2.2.1 Simple Security

The first aspect of security considered is the simple security property (ss-property). The ss-property is satisfied if every "observe" access triple (subject, object, access) in the current access set b has the property that the maximum level of the subject dominates the level of the object. Symbolically,

$$(S, O, \underline{\text{observe}}) \in b \Rightarrow f_c(S) \succcurlyeq f_o(O) \text{ where}$$

observe is one of r or w.

This property protects objects from direct unauthorized observation, in a non-discretionary (static) way.

2.2.2 *-Property

An indirect security threat exists when a subject has the potential of observing a high security object and concurrently writing into a lower security one. If any high security data is written into the lower security object, unauthorized observation is possible by subjects with security lower than the high security object.

Such an indirect threat can be prevented by conforming to the

* - property[1], which is satisfied if:

in any state, if a subject has simultaneous observe access to object-1 and modify access to object-2, then the security level of object-1 is dominated by the security level of object-2.

Symbolically:

$(S,O_1,\text{observe}) \in b$ and $(S,O_2,\text{modify}) \in b \Rightarrow f_o(O_1) \propto f_o(O_2)$

where observe is r or w and modify is w or a.

If a subject has w-access to two objects, their security levels must be equal since the *-property requires that their levels dominate each other, since w-access involves both observe and modify.

Observing both the ss-property and the *-property will cause the levels of all objects accessed by a given subject to be neatly ordered:

Level (a - accessed-object) dominates level (w - accessed-object);

Level (w - accessed-object-1) equals level (w - accessed-object-2); and

Level (w - accessed-object) dominates level (r - accessed-object).

[1] read "star-property"

It proved useful to define a special set of subjects, called "trusted subjects", which are not constrained by the *-property, but are trusted not consummate a security-breaching information transfer.

2.2.3 Tranquility Principle

The tranquility principle states that the security levels of active objects will not be changed during normal operation.[2] This will disallow the classification of objects to vary according to the accesses performed on them. This is essential so that objects will not be downgraded. Additionally upgrading data could result in the overclassification of information, reducing its usability.

Non-discretionary security refers to an environment where the ss-property, the *-property and the tranquility principle are maintained by the system. The definition of security level ensures that clearance matching and the formal need-to-know requirements are observed.

2.2.4 Discretionary Security

Discretionary security policy allows an individual to extend to another individual access to objects based on his own discretion, constrained by non-discretionary

[2] J.K. Millen, Security Kernel Validation in Practice, Communications of the ACM, Volume 19, Number 5 (May 1976), 244-245.

security policy. In the model this property is called the discretionary security property (ds-property). This property requires that:

> if (subject-i, object-j, access-x) is in b, then access-x is recorded in the (subject-i, object-j) - component of M.
>
> i.e., $x \in M_{ij}$

Note that additions to the concept of security will not impact these properties because additional restrictions can only reduce the set of reachable states.

2.3 General Mechanisms

The first general result in the model is the basic security theorem which states that security can be guaranteed systematically when each alteration to the current state does not itself cause a breach of security. Thus security can be guaranteed systematically if, whenever (subject, object, access) is added to the current access set b, then:

1) the ss-property is maintained;
2) the *-property is maintained;
3) the tranquility principle is maintained; and
4) the ds-property is maintained.

This theorem establishes the "inductive nature" of security in that it shows that the preservation of security from one state to the next guarantees total system security.

The second general mechanism within the model is a direct consequence of the basic security theorem. A general framework for isolating single state transitions is devised. This framework relies on the "rule", a function for specifying a decision (an output) and a next state for every state and every request (an input):

(request, current-state) rule (decision, next state)

Then each class of requests is analyzed separately in a rule designed to handle that particular class. For clarity, no two rules are allowed to specify non-trivial changes for a given (request, current-state) pair. System "response" to the pair (request, current-state) is then defined to be the response of the rule for that request. This framework allows differing techniques for different rules.

The last general development centers on the relation of rule properties to system properties. The entire system specified by a set of rules satisfies all three security properties (ss, *, and ds) provided each rule itself introduces no exception to these properties. Moreover, the demonstration of the security preservation of a rule in most cases can be reduced to direct consideration of a small number of state alterations involved in a given state transition.

2.4 Specific Solutions

A particular solution to the model consists of specifications for requests, rules, decisions, and model elements. The particular solution in the Bell-La Padula report was specifically tailored for use with a Multics-based information system design. Two requirements the solution satisfied were: the provision of generally useful functions and appropriate accommodations to the effects of the Multics design on an implementation of this model.

The general functions relevant to the model can be grouped in four classes:

A. functions to alter current access (the set b);
   1) to get access (add a triple "(subject, object, access)" to b); and
   2) to release access (remove a triple from b);

B. functions to alter the security levels of subjects and objects:
   1) to change object level (change the value f_o (object) for some object), and
   2) to change current level (change the value f_c (subject) for some subject);

C. functions to alter the current access permission structure (the matrix M):
   1) to give access permission (to add an attribute to some component of the access permission matrix M), and

2) to rescind access permission (to delete an attribute from some component of M); and

D. functions to alter the object structure (the hierarchy H):

1) to create an object (to attach an object to the current tree structure as a leaf), and

2) to delete a group of objects (to detach from the hierarchy an object and all other objects "beneath" it in the hierarchy).

These rules reflect the main Multics characteristic affecting the model, namely the hierarchical object organization. The basic Multics mechanism for access control rely heavily on this object structure.

The second Multics characteristic involves the implemented counterpart of the access permission matrix M. This structure is called the Access Control List (ACL), and there is one stored with every object. Each ACL is a list of processes (subjects) allowed to access the corresponding object, as well as the modes of access allowed. These ACL's are contained and manipulated in an object's parent object (i.e., directory object). Therefore, "control" over an object is equivalent in Multics to write permission to the objects parent. Since object "creation" in Multics is the insertion of a new entry in the parent object, creation control is equivalent to append access to the object's parent.

The final Multics characteristic reflected in the model is the way access to an object is performed. A user request to access an object causes the user's surrogate (process) to access every object in the hierarchy in the path from the root directory to the segment of interest. The rules of the model require that the security level of an object dominate that of its parent object.

The Bell-La Padula solution provides a particular specification for a secure computer system that supplies a full complement of information processing capabilities while matching the special requirements of the Multics operating system environment.

SECTION III

SPECIAL CONSIDERATIONS

3.0 Introduction

When the Bell-La Padula model[1] was considered as a basis for modeling a general protected data management system, certain modifications were clearly required. Other changes and extensions seemed desirable, while some created uncertainty. A list of the potential areas of change was produced, and each was considered in detail. This section will treat each item on this list of special considerations. Since each subsection will use only the concepts that have been developed up to that point, the changes may not be in finalized form.

3.1 Integrity

The security mechanisms defined in the Bell-La Padula model are not sufficient to permit adequate control over modification access. For example, modification authorization cannot be restricted in a non-discretionary manner. Therefore, additional mechanisms are desirable.

Integrity is a static property of a dynamic system[2], which requires that the initial "soundness" of a system be maintained throughout its activities. Since modifications

[1] D.E. Bell and L.J. La Padula, Secure Computer System: Unified Exposition and Multics Interpretation, ESD-TR-75-306

[2] K.J. Biba, Integrity Consideration for Secure Computer Systems, in preparation, The MITRE Corporation, Bedford, Massachusetts.

potentially deteriorate the initial soundness of a system, modification control will be considered an integrity issue.

The authorized nature of a modification, and its actual correctness are two aspects of integrity. However, we will simplify our model by limiting the scope of integrity to address only the authorized nature of modification.

Like data observations, modifications should be directly controllable. Yet observation and modification must be independently controlled. Then, for example, certain observations can be permitted to a large set of users, while severely limiting the capability for modification, in a non-discretionary manner.

An indirect integrity threat exists in a situation which corresponds to the indirect security threat addressed by the *-property. That property requires that the security level of a modified object dominate that of any observed object, since otherwise information previously read with higher security may be passed into a location with lower security. The corresponding integrity mechanism will require that the integrity level of an observed object dominate that of any modified object, since otherwise data subsequently modified with high integrity may contain data of lower integrity. The control of this indirect integrity threat will be dually called the $*_i$-property.

The requirement for direct and indirect modification authorization control suggests that integrity mechanisms can be defined similar to security mechanisms. Then, for convenience, an integrity level can be defined to consist of a classification and a set of categories, as does a security level [c.f. § 2.1]. Integrity level dominance, $\succ_i$, can be defined to be the same as security level dominance. Symbolically:

an integrity level: $Z = (C,K)$

where C = a classification and

K = a set of formal need-to-know compartments.

If $z_1 = (C_1, K_1)$ is an integrity level, and

$z_2 = (C_2, K_2)$, then

$z_1 \succ_i z_2$ if and only if $C_1 \geq C_2$ and

$K_1 \supseteq K_2$.

The integrity level function (g) is a component of the system state, embodying integrity classifications in the model. The function assigns to each subject and each object an integrity level. Analgous to security, the level function g is a triple (g_s, g_o, g_c), where:

g_s = maximum integrity levels of all subjects;

g_o = integrity levels of all objects;

g_c = current integrity levels of the active subjects.

The purpose of integrity is to embody an integrity policy by allowing the assignment of levels to appropriately restrict the modification of objects in a non-discretionary (static) way. The integrity policy will be similar to the security policy.

Data will be protected from direct unauthorized modification if a system maintains a "simple integrity property". It requires that a subject can modify an object only if the maximum integrity level of the subject "dominates" that of the object.

Data is protected from an indirectly unauthorized modification if the system maintains what can be called the "$*_i$-property". It requires that if a subject has simultaneous modify access to object-1 and observe access to object-2, then the integrity level of object-1 is dominated by that of object-2.

Observing both the simple integrity property and the $*_i$-property will cause the integrity levels of all objects accessed by a given subject to be ordered:

g (r - accessed object) $\succcurlyeq$ g (w - accessed object);

g (w - accessed object-1) = g (w - accessed-object-2); and

g (w - accessed object) $\succcurlyeq$ g (a - accessed object).

The tranquility principle for integrity is defined as for security, and states that the integrity levels of active objects will not change during normal operation.

Non-discretionary integrity refers to an environment where the simple integrity, the $*_i$-property, and the tranquility principle, are maintained by the system. The definition of integrity level ensures that clearance matching and the formal need-to-know requirements are observed.

Discretionary integrity policy is already embodied in the discretionary security property [c.f. § 2.2].

Execute access (e) is not subject to the simple security property according to Bell and La Padula [c.f. § 2.2]. There are, nevertheless, integrity implications in execute - access, and these are explored in subsection 3.6. For the purposes of this subsection, execute will be considered exempt from integrity properties.

Symbolically, integrity considerations can be expressed as:

1) Simple-integrity property:

   (subject,object,modify) ε b => g_s(subject) $\geq$ g_o(object)

   where modify is a or w,

2) $*_i$-property:

   (S,O_1,modify) ε b and (S,O_2,observe) ε b =>

   $$g_o(O)_1 \propto g_o(O)_2,$$

   where modify is a or w and observe is r or w;

3) Tranquility principle:

$g_o(O)$ is constant if object O is active

(i.e., accessed by some subject);

4) Discretionary integrity:

$(S_i, O_j, \underline{x}) \in b \Rightarrow \underline{x} \in M_{ij}$

3.2 Protection

"Protection" is the complete set of mechanisms which cause data access in a system to conform to an access authorization policy. In the I.P. Sharp model, protection will include the Bell-La Padula notion of security with the addition of integrity as defined above.

Since the authorization of every data access must be checked, the procedure to determine whether or not the security and integrity properties are satisfied must be as simple and efficient as possible. To this end it is desirable to judiciously consolidate the security and integrity mechanisms, producing simple properties involving a minimum of entities.

It can be seen from Table 3.1 that the security and integrity properties are not easily combined for a type of access, since different levels (e.g., subject security level, object security level, object integrity levels for observe) are involved. This problem can be overcome if

	observe	modify
SECURITY	$f_s(S) \succ f_o(O_{obs})$	$f_o(O_{obs}) \prec f_o(O_{mod})$
INTEGRITY	$g_o(O_{mod}) \prec g_o(O_{obs})$	$g_s(S) \succ g_o(O_{mod})$

Table 3.1 Security and Integrity Properties

	observe	modify
SECURITY	$f_c(S) \succ f_o(O)$	$f_c(S) \prec f_o(O)$
INTEGRITY	$g_c(S) \prec g_o(O)$	$g_c(S) \succ g_o(O)$

Table 3.2 The Consolidated Security and Integrity Properties

the properties are re-defined in terms of the current-levels of the subjects. This is suggested by the fact that the current level can represent the current "limits" of the levels of objects a subject will be permitted to observe or modify[3]. The consolidation will proceed as follows:

A protection level is defined to be an integrity level appended to a security level. Symbolically the j^{th} protection level is:

$P_j = (C_j, K_j, C_j', K_j')$ where

C_j = a security clearance or classification;

K_j = a set of security categories;

C_j' = an integrity clearance or classificaiton; and

K_j' = a set of integrity categories.

Analogous to the Bell-La Padula level function f [c.f. § 2.1], subjects and objects are mapped into protection levels by the function (Π), which is a triple (Π_s, Π_c, Π_o), where:

Π_s = protection level limits for all subjects;

Π_c = current protection levels of the active subjects; and

Π_o = protection levels of all objects.

[3] D.E. Bell, Secure Computer Systems: A Refinement of the Mathematical Model, Volume III, ESD-TR-73-278, page 18.

The following definitions are consistent with the security and integrity mechanisms:

$\Pi_o(O) = (f_o(O), g_o(O))$ for object O;

$\Pi_s(S) = (f_s(S), g_s(S))$ for subject S; and

$\Pi_c(S) = (f_c(S), g_c(S)$ where $f_c(S) \propto f_s(S)$ and $g_c(S) \propto g_s(S)$.

To aid in the definition of protection "dominance" and properties, the properties summarized in table 3.1 are re-defined to involve current levels.

The main reason for the existence of current security levels is to allow dramatic simplifications of the *-property checks. If a subject wishes to write (both observe and modify) at a particular level, he will choose a current security level equal to that level. This follows from the fact that he can modify only dominating objects and observe only dominated ones.

Consequently, the *-property is re-written:

S modifies O => $f_c(S) \propto f_o(O)$

Table 3.2 gives the security and integrity properties when those in table 3.1 are re-defined in terms of a subject's current level. Then the DUAL nature of the integrity properties with respect to the security properties can be easily observed, since the dominance relationships are exactly opposite. This suggests that protection dominance be defined as a dominating security level AND a

dominated integrity level. Symbolically, if P_i is the i^{th} protection level, then $P_i = (C_i, K_i, C_i', K_i')$:

$P_i \succ' P_j$ iff: i) $C_i \geq C_j$;

ii) $K_i \geq K_j$;

iii) $C_i \leq C_j$; and

iv) $K_i \subseteq K_j$.

With this definition of protection dominance, protection properties analogous to the security properties can be defined by consolidating properties regarding a given access mode [c.f. table 3.2].

3.2.1 Simple Protection Property

A subject may observe an object only if the current protection level of the subject dominates ($\succ'$) the protection level of the object. This definition embodies both the modified simple security property and the modified $*_i$-property [c.f. table 3.2]. Symbolically:

(subject,object,observe) $\in$ b => Π_c(subject) $\succ'$ Π_o(object)

where observe is r or w.

3.2.2 *'-Property

A subject may modify an object only if the protection level of the object dominates the current protection level of the subject. This definition embodies both the modified *-property and the modified simple integrity property [c.f. table 3.2].

Symbolically:

(subject,object,<u>modify</u>) ε b => Π_c(subject) $\prec'$ Π_o(object)

where <u>modify</u> is <u>w</u> or <u>a</u>.

It is apparent that

(subject,object,<u>w</u>) ε b => Π_c(subject) = Π_o(object),

since i) Π_c(subject) $\succ'$ Π_o(object) for observation, and

ii) Π_c(subject) $\prec'$ Π_o(object) for modification, and

iii) $\succ'$ is a partial ordering [c.f. § 5.2].

Observing both the simple protection property and the *'-property will cause the protection levels of all objects accessed by a given subject to be ordered:

Π(<u>a</u> - accessed-object) $\succ'$ Π(<u>w</u> - accessed-object);

Π(<u>w</u> - accessed - object-1) =Π(<u>w</u> - accessed-object-2); and

Π(<u>w</u> - accessed-object) $\succ'$ Π(<u>r</u> - accessed-object).

"Trusted subjects" [c.f. § 2.2.2] are not included in the model since they are considered to be proven programs, which are completely within the security perimeter and therefore not subject to the rules of the model.

3.2.3 Tranquility Principle for Protection

This principle states that the protection levels of active objects will not change during normal operation. It is required because it is essential that objects are not downgraded. Also this principle removes the risk of a varying level constituting a communication path.

Symbolically:

$$\Pi_o(O) = \text{constant for an object O.}$$

Non-discretionary protection refers to an environment where the simple protection property, the *'-property, and the tranquility principle are maintained by the system. The definition of protection level ensures that clearance matching and formal need-to-know requirements are observed.

3.2.4 Discretionary Protection

The discretionary protection property is defined to be the same as the discretionary security property, namely that if subject -i has $\underline{x}$-access to object-j, then $\underline{x}$ is recorded in the (subject-i, object-j) component of M (M_{ij}).

Symbolically:

$$(S_i, O_j, \underline{x}) \in b \Rightarrow \underline{x} \in M_{ij}.$$

3.3 The Directory System

In the Bell-La Padula model the basic access control mechanisms rely heavily on the hierarchical structure of the directory system.[4] In the new model the directory objects will serve the same purpose, but changes are desirable for the following reasons:

i) The path through the hierarchy must be specified in order to access an object. However, it will be impossible to access the object unless the subject has discretionary access to every directory object in the path; and

ii) The new directory system (unlike the hierarchical one) does permit searching for all dominated objects, which is very desirable in a general data management system.

Although the directory system will be completely transparent to the system user, its components are modelled to address protection issues adequately in this important system mechanism. A description of the new directory system now follows.

In § 5.2 it is proven that the complete set of protection levels [c.f. § 3.2] form a LATTICE with respect to protection dominance. Figure 3.1 illustrates some of the

[4] D.E. Bell and L.J. La Padula, Secure Computer System: Unified Exposition and Multics Interpretation, page 21.

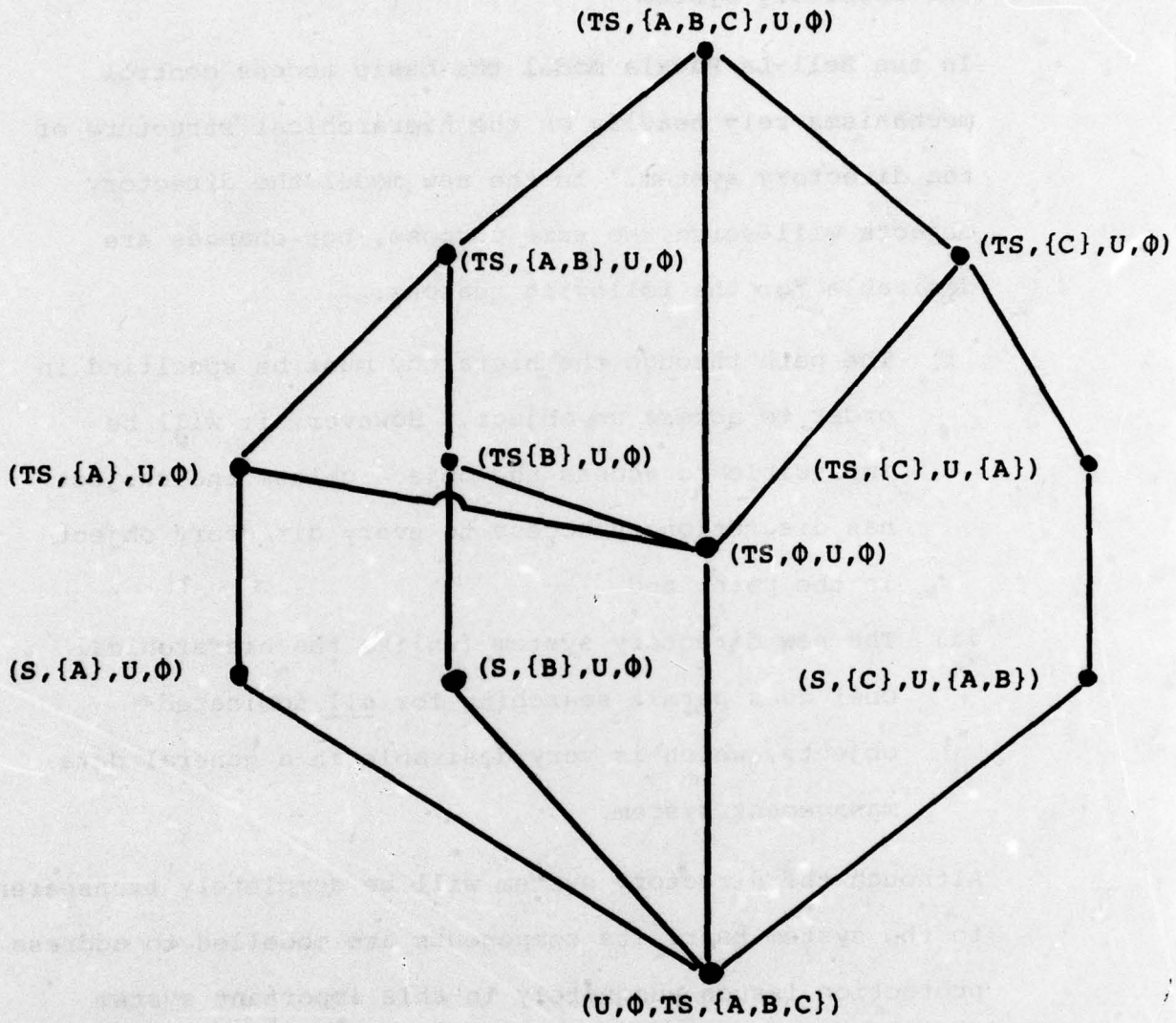


Figure 3.1 An Example of a Sub-Lattice of Protection Levels.

dominance relationships in an arbitrary subset of levels, which form a sub-lattice. The components of the new directory system required to protect the identifiers of such objects in a flexible manner are:

i) a partition ($\{L_1, L_2, \ldots, L_n\}$) of the set of all object identifiers at all protection levels into a set of directories, where each directory (L_i) contains the identifiers of all objects possessing a certain level (P_i). Each directory will itself possess that protection level, and be exempt from discretionary access control (in response to reason (i) above).

ii) a set of lattice directory functions:

   (a) a dominating function, λ_{α}, where λ_{α}(level) is a list of the levels of all existing directories which level dominates;

   (b) a dominated function, $\lambda_{\succ}$, where $\lambda_{\succ}$(level) is a list of levels of all existing directories which dominate level;

The directory functions will be important to facilitate a search for objects when their level (and even name) are not known [c.f. § 3.5.1]. For example, $\lambda_{\succ}$(subject's-current-level) will be vital to the servicing of a request like:

"Blindly append this data to object O, whatever its level is." Also, λ_{α}(subject's-current-level) will be required for the request:

"Identify the objects I am authorized to observe."

iii) a master directory, which contains an unordered list of the protection levels of all existing directories. This component is essential to the implementation of the λ -functions. An algorithm to achieve such an implementation is presented in appendix I.

The Bell-La Padula hierarchy allowed the path to an object to serve as a unique identifier of the object. A change from the use of a unique path per object will result in name uniqueness problems. When an object is created at a certain level, any information made available regarding an object with an identical identifier at a "higher" level will constitute a violation of the *'-property. A solution to this is to make an object's level part of its identification.

As an aid to name uniqueness for a given level, an indication of the creating subject may also form part of an object's identifier. Additionally this will allow some validation of certain operations (such as deletion) to be performed.

The new directory system will accommodate the "real-world" requirement for allowing the identifier (existence) of an object to be stored in a directory "lower" than the actual object (essence). For example, the title of a report may be confidential, while the contents are secret. To serve this requirement a code will be maintained with each identifier in a directory to indicate:

(i) the identifier is not found in a directory "lower" than the object; or

(ii) the identifier is found "lower" in level P_j directory; or

(iii) the object is actually "higher" at level P_j.

Note that this mechanism will allow storing an object's identifier in just <u>one</u> directory at a level "lower" than the object.

In conclusion, the new directory system will serve as the mechanism to allow unique identification of objects, in terms of a name and a protection level, to serve the requirements of a general protected data management system.

3.4 Protected Object Structure

When dealing with an object in a data base, it is reasonable and useful to distinguish between the descriptive and the value aspects of the object. This will allow a

data base to be self-descriptive and therefore system independent.

To model this distinction, for object O:

$O = (D_O, V_O)$ where

D_O represents the descriptive aspect, including current status; and

V_O represents the value aspect.

Another aspect of an object in a protected environment is that access to it must conform to discretionary access policy. This can be modelled by letting M_O represent the column of the Bell-La Padula access permission matrix, M[c.f. § 2.1], for object O. Let this be called the "permission matrix for object O", since it will contain all the accesses authorized to the object for a set of subjects. More specifically, M_O will consist of a set of pairs:

(subject,access)

The three aspects of a protected object, O, can be conveniently modelled by:

$O = (M_O, D_O, V_O)$.

3.5 The Access Function

In the Bell-La Padula model an object cannot be accessed until every directory object in the unique path to it has been accessed. In the new model the accessing of an object

will involve only one directory. This of course does not interest the end-user, since the new model's directory system will be for the most part invisible to him [c.f. § 3.3]. However, it is relevant to the "total" system point of view.

The direct access function, δ , is defined to serve the following vital functions:

(i) to model object access;

(ii) to model directory access;

(iii) to enforce protection requirements on accesses; and

(iv) to allow the directory system to be invisible to the end-users.

The remainder of this subsection describes the nature of accesses in the model, regardless of how the end-user visualizes them.

3.5.1 Access of Directories

The accessing of directories is essential to serve the end-user's requirements for object name and protection level management. Although this involves directory manipulation, an end-user will be aware only of requests involving object identifiers.

Examples of such requests are:

(i) "Identify the objects I am authorized to observe."

This request is essential since certain subjects will not possess perfect memories, and it is undesirable that such information be maintained in the "external" world of paper and ink. The servicing of this request will involve λ_{α}(subject's-current-level) [c.f. § 3.3];

(ii) "Create object O at level P".

A modification of the level P directory will be made, and λ_{α}(subject's-current-level) will be used to check for name uniqueness (O) in accessible directories.

The modelling of such directory-oriented activities consists of:

δ(subject,level,access) = directory, if the protection requirements are satisfied,

= 'FAILURE', if not,

where access = observe or modify.

For example, to model the performance of:

"Identify the objects I am authorized to observe":

δ(subject,P_i,observe) = L_i for all $P_i \in \lambda_{\prec}$(current-level),

where L_i is the directory possessing level P_i.

3.5.2 Access of Objects

Since access control is to rely heavily on the directory system [c.f. § 3.3], object access will be a function of directory entries. The subject submitting a request will not receive any information regarding consequent accesses which would constitute a violation of the *-property. For example, an append "up" must be "blind"; in that there can be no notice of success or failure.

Examples of requests for object access are:

(i) "Display the contents of object O at level P_i" and

(ii) "Append data at level P_1 to object O at P_2" ($P_2 \succ P_1$).

Such accessing of objects is modelled by:

δ(subject,L_i(O),access) = (M_o,D_o,V_o)[5] if the protection requirements are satisfied,

= 'FAILURE' if not,

where L_i(O) is a directory entry for object O.

[5] See protected object structure in § 3.4.

3.6 Protect Execute Access

The Bell-La Padula model does not protect access (e) with the simple security or the *-property [c.f. § 2.2]. However, there are potential protection threats involved.

A program is like any other object in that it possess a name, a protection level, a descriptor and a value set [c.f. § 3.4]. It is different in that the value set consists of an ordered set of requests for system services and accesses.

The execution of a program by a process involves the utilization of the program's request set by the process. This essentially constitutes an observation, especially if the program incorporates data. Therefore the simple security property is relevant [c.f. § 2.2].

There is an indirect modification threat involved in the execution of a program in the sense that a program can constitute a set of "modification instructions" to be applied to previously accessed objects. To ensure that such a set of instructions are sufficiently authorized to modify an object, the $*_i$-property should be satisfied by execute access [c.f. § 3.2].

The tranquility principle [c.f. § 3.2] requires that a program's protection level not be changed in response to an execution. However, the subject of the program's requests is the process, at its current protection level.

The protection of execute access will be enhanced if it is subject to discretionary control.

Since execute access is subject to the simple protection property (simple security and $*_i$-properties), the tranquility principle and discretionary control, for the purposes of the model, execute can be considered to be read access.

3.7 Change the Set of Access Modes

In the Bell-La Padula model, A is a set of access attributes, or modes, in which a subject may access an object in the system. More specifically, A = {e,r,w,a}[c.f.§2.1].

To simplify the model, this set can be reduced as follows: Execute access (e) can be removed since section 3.6 establishes it to be identical to read (observation); and write access (w) can be removed since it is a combination of observation and modification.[6]

The resulting access mode set will then be:

A = {o,m} , where:

o = observe access, which involves the extraction or utilization of data from an object; and

[6] D.E. Bell & L.J. Padula, Secure Computer System: Unified Exposition and Multics Interpretation, page 7.

m = modify access, which involves modifying the object without any observation of it. Bell and La Padula call this an append (a). Note that a destructive modification (replace) will require both observe and modify access.

This set of two access modes is sufficient, since the protection properties [c.f. § 3.2] specifically address only these two.

3.8 Change the Request Set

In the Bell-La Padula model, a request is an input to the system which has an associated rule which specifies the output [c.f. § 2.3]. Such requests are grouped according to the model elements involved in servicing the request (i.e., relevant to the rule). Since the new model has a different set of elements, the set of requests will be different also.

In the new model, requests will be grouped according to type of access to the model elements which contain the information relevant to a data management system. Subsection 3.7 established two types of access: observe and modify. Subsection 3.4 describes three important model elements: an object's permission matrix, description and values. The importance of requests involving object names and levels is established in subsection 3.3.

Therefore, there are eight classes of requests when they are grouped by access on model elements. Table 3.3 gives a list of this set of request classes.

An important consideration is to determine the implications of combinations of these request classes. One point is that both Q_X^O and Q_X^M (where X ε {L,M,D.V}) are required in order to perform a destructive modification (replace) of data in model element X. Another is that a Q_D^O access is needed for the correct performance of an access to the corresponding values. Discretionary access control will require a Q_M^O access request before an object access can be performed, to confirm that the access is authorized.

Aside from the relationships mentioned above, request classes are distinct from and independent of each other, so combinations of them cause no problem.

CLASS	ACCESS MODES	MODEL ELEMENTS
Q_L^O	observe	directory
Q_L^M	modify	directory
Q_M^O	observe	permission matrix
Q_M^M	modify	permission matrix
Q_D^O	observe	object description
Q_D^M	modify	object description
Q_V^O	observe	object values
Q_V^M	modify	object values

Table 3.3 Table of Classes of Requests

3.9 Elaboration on Subjects

The Bell-La Padula model is very detailed regarding the secure access of data objects, and less detailed about the nature of subjects. This topic will be explored now.

A subject is an active entity in the system.[7] Active entities include users (persons) and processes (programs in execution).

A program is defined to be an ordered set of requests [c.f. § 3.6], some of which may be requests for protected objects. Other requests are control requests, such as changing the order of request servicing. A process is created when the requests in a program are "executed" (observed and serviced) on behalf of a user by the system, to be performed concurrently with other processes. The invoked process "inherits" the user's protection level, and becomes the subject of all requests in any program executed.

When a process creates another process, the *-property requires that the flow of data can only be in one direction if the created process is assigned a level different from the creating one. For example, parameters can be passed only to a process executing "higher". Status information (such as success or failure) can be returned only from a process executing "lower".

[7] Secure Computer System: Unified Exposition and Multics Interpretation, page 5.

It is desirable that an active subject not change its protection level during normal operation. If it could, then the success or outcome of a given program could depend on whatever peculiar activities the process was previously involved in. This would make results less predictable. Additionally, varying protection levels could result in communication paths.

Therefore the tranquility principle [c.f. § 3.2] will be extended to apply to subjects as well. Symbolically:

$$\Pi_c(\text{subject}) = \text{a constant.}$$

If multi-level activities are to be performed, two alternate techniques to performing them are:

(i) Cause multiple independent processes of varying levels to perform the activities; or

(ii) A subject can sign-off the system (dropping all active accesses) and sign-on again at a different level.

An important distinction must be made between users and programs. Certain identifiable users (persons) may be allowed to reclassify objects to a lower protection level, using proven programs, since this capability can be considered protected by the user's protection attribute. That is, the validity of the reclassification is the

responsibility of a user's judgement, within the scope provided by his protection attribute. However, a process (program) cannot be allowed to reclassify an object since it may contain an unproven program, and this constitutes a protection threat.

Persons transfer their protection attribute to their computer processes through an input device such as a card reader or terminal keyboard. Since the scope of the model does not include entities external to the data management system, users and I/O devices will not be included in the set of subjects.

In conclusion, studying the nature of processes results in no need being identified for changing the model in any way.

3.10 The Relational Approach

Since the model is to represent data management systems, data structures and operations must be embodied in it. The relational approach to data management was chosen for the following reasons:[8]

1) to provide a high degree of data independence. Then access paths can be established (independently of programs) to involve the data accessed most frequently;
2) to provide a simple and consistent data model, and make it broadly usable;
3) to introduce, through relational algebra, a theoretical foundation (albeit modest) into data base management; and
4) to raise application programming to a new level, where data (relations) are treated as operands instead of being processed element by element, in ad hoc ways.

In the following subsections concepts from the relational approach will be given, and the terms required will be defined. Suitable data management mechanisms, such as a directory system and an access function will be described as well.

[8] E.F. Codd and C.J. Date, Interactive Support for Non-Programmers: The Relational and Network Approaches, IBM Research Report RJ 1400, San Jose, California, June 1974, page 4.

3.10.1 Relations

An important purpose of a data management system is to manage sets of data elements and the relationships among them. A data element is an atomic item of data which is not considered to be sub-divided. Consider a set of sets of data elements $S_1, S_2 \ldots, S_n$, which are not necessarily distinct. R is a relation on these n sets if it is a subset of the Cartesian product of some of the sets of data elements.[9] Symbolically:

$$R \subseteq S_1 \times S_2 \times \ldots \times S_n$$

R consists of a set of tuples, where each tuple has its first element from S_1, its second element from S_2, and so on. The set S_j is referred to as the j^{th} domain of R. Thus, R is essentially a "table" of data elements in rows and columns, where each row consists of a tuple of "related" data elements.

A key K of a relation R is a subset of domains of R such that:[10]

(i) In each tuple of R, the value of K uniquely identifies that tuple; and

(ii) No domain in K can be discarded without destroying property (i).

[9] E.F. Codd, Further Normalization of the Data Base Relational Model, Data Base Systems, Courant Computer Science Symposium 6, Prentice-Hall, Englewood Cliffs, New Jersey, 1972, 33-64.

[10] E.F. Codd, A Relational Model of Data for Large Shared Data Banks, Communications of the ACM, Volume 13, Number 6 (June), 1970, 377-387

There must always exist at least one key, which may be the whole of R, and there can clearly be several. For operational purposes, one of the keys is arbitrarily designated a prime key, which cannot have an undefined value in any tuple of a relation. Any other keys or domains of R may contain undefined values. All keys must preserve properties (i) and (ii) above. It can be useful to keep track of which domains form keys for those procedures or queries which wish to uniquely identify a tuple with different combinations of domains (e.g. for normalization).

A relationship between relations exists when tuples of one relation are related to the tuples in another in some way. One method of making this relationship explicit is by defining a new relation and giving it a name. Each row in this relation consists of a tuple key from one relation followed by a tuple key from a related tuple in the other relation. Every tuple-to-tuple relationship is so indicated. For example, figure 3.2 illustrates the SUPPLIES relationship between the SUPPLIER and PARTS relations.

RELATION: SUPPLIER

KEY: NAME

NAME	STREET	CITY	SPECIALITY
ACME	2949 Lexington	Toronto	General
HI-VALU	10 Place Bonaventure	Montreal	Electronics
J.C. GOODS	50 Main Street	Carp	Plumbing

RELATION: PARTS

KEY: PART NUMBER

PART NUMBER	PART DESCRIPTION	QUANTITY ON HAND
A1014-J	Green Widget	100
AB76-110	Rubber Duck	17
A147661	Metal Funnels	43
H92-101	Small Calculator	11
J441-1977	Plastic Pipe	203
J10	Box of Oakum	31
X12765-110	Black Box	2

Figure 3.2 An Example of a Relationship between Relations

(continued on next page)

Figure 3.2 - continued:

RELATION: SUPPLIES (A stored relationship between SUPPLIER and PARTS).

(The whole relation forms the prime key.)

KEY OF SUPPLIER: NAME	KEY OF PARTS: PART NUMBER
ACME	A1014-J
ACME	AB76-110
ACME	A147661
HI-VALU	H92-101
J.C. GOODS	J441-1977
J.C. GOODS	J10

Figure 3.2 An Example of a Relationship between Relations

Normalization is a process of removing undesirable functional dependencies from the information stored in a relation.[11] Appendix II gives a detailed description of the normalization of relations.

There are difficulties with the automatic normalization of relations by the system. Functional dependencies must be

[11] E.F. Codd, Normalized Data Base Structure: A Brief Tutorial, IBM Research Report RJ935, San Jose, California, November 1971.

stated before it can occur. This makes automatic normalization undesirable. Therefore, automatic normalization is not a part of the model's relational approach.

In a relational data management system distinctions can be made between types of relations.[12] Primary relations are those defined independently and tuples are inserted in them directly. Derived relations are defined using relational operators,[13] such as union or join, on multiple primary relations. Snapshots are derived relations which have an independent existence after they are created. Views are derived relations which consist of the definition of their derivation. Thus views continue to reflect changes in the primary relations, Snapshots do not. Figure 3.3 gives an example of a view.

[12] D.C. Tsichritzis, On Implementation of Relations, Technical Report CSRG-35, Computer Systems Research Group, University of Toronto, May 1974.

[13] E.F. Codd, Relational Completeness of Data Base Sublanguages, IBM Research Report RJ987, San Jose, California, March 1972.

Define View: CLUB - OFFICIALS - MAILING-LIST = PROJECT(NATURAL-JOIN OF CLUB-OFFICIALS AND PERSONNEL ON NAME) TO INCLUDE NAME AND ADDRESS

CLUB-OFFICIALS-MAILING-LIST = is a view of two relations

NAME	ADDRESS

CLUB OFFICIALS

CLUB	POSITION	NAME
BASKETBALL	VICE-PRES	Smith, A.
GLEE	PRESIDENT	Jones, M.
TOASTMASTERS	TREASURER	Green, T.

PERSONNEL

NAME	ADDRESS	PHONE	AGE	SALARY
ABBOT, B.	15 Shea Rd.	237	35	$16,000
GREEN, T.	110 Carling	119	29	15,000
JONES, A.	4 Selkirk	399	19	9,000
JONES, M.	4 Selkirk	701	19	9,500
SMITH, A.	RR#2	800	24	12,800

Figure 3.3 An Example of a View

3.10.2 Protection in a Relational Data Base

An important issue is the decision as to how a relational data base should be mapped into the set of protection levels. Our approach to arriving at a decision was to study the restrictions which resulted from alternate mapping assignments.

Some assignment possibilities were quickly rejected because of unacceptable disadvantages. Assigning protection to a whole data base (collection of relations) is too coarse an assignment and would impede data sharing. On the other hand, assigning protection to data elements would introduce an unacceptable amount of overhead in any proposed implementation.

It is possible to assign levels as a function of individual data values. For example, in a personnel data base it may be possible for an employee to see but not modify his own salary information. This approach is difficult to manage since levels will change as data changes.

These considerations suggested choosing a data structure "between" a data element and a data base.

3.10.2.1 Tuple Assignment

If protection levels are assigned to the tuples of a relation, there are restrictions on certain tuple activities. A tuple cannot change its protection level when such a change would constitute a communication path. For example, suppose the tuples of a "VEHICLE-CARGO" relation describe the particular cargo a vehicle is transporting. A communication path is established if a tuple is visible to unclassified users when tissue paper is transported, but invisible in the case of nuclear weapons.

There would be a problem in defining a directory system to support the assignment of protection to individual tuples. The relations might each need several aliases each having a different protection level. An alias would have a protection level consistent with a subset of tuples in the overall relation. Management of such a heterogeneous directory/relation schema would be difficult.

There is also the possibility that a high-level tuple must contain exactly the same data as a low-level tuple. This situation cannot be accommodated by the definition of relation keys [c.f. § 3.3.1].

3.10.2.2 Domain Assignment

If protection levels are assigned to the domains of a relation,[14] restrictions on activities arise. For instance, for a relation with domains of varying levels, for any observable tuple, only those domains with a protection level dominating the subject's are modifiable (*'-property). This can cause problems with tuple-oriented reading or updating. For example, it may be necessary to insert a tuple into a relation a domain value at a time (when each has a different level), signing on a different level for each domain value. This technique can be inconvenient.

If the prime key of a tuple is at a higher protection level than the other domains, the key will be unobservable to some subjects and the tuple will be invalid. Therefore the formation of relations will be restricted to those in which all domain levels dominate that of the prime key.

In the special case where every tuple is completely observable and modifiable, all domains must possess this same level, and this can be viewed as protection assignment to a relation.

[14] This approach has been implemented for Multics by System Development Corporation, and is described in: T. Hinke and M. Schaefer, Secure Data Management System, System Development Corporation, Report number RADC-TR-75-266, Santa Monica, California, November 1975.

3.10.2.3 Relation Assignment

The preceeding considerations led to our choice of assigning protection levels to relations, since it would cause no restrictions on either tuple-oriented or domain-oriented activities. This assignment is consistent with the use of relations as "units" of information, since the notion of a relation will become a "table" of data at a certain level.

There are some problems with this assignment, such as potential data overclassification, and inconsistencies caused when "low-level" relations are deleted in spite of there being "high-level" relations which reference their tuples (usually by a key value). But these problems will not cause an unauthorized access.

3.10.3 Relations and the Model

Since protection levels are to be assigned to relations and the model assigns levels to objects, it follows that a relation is represented by a model object. This subsection will illustrate how relation entities may be represented given the correspondence of object to relation. These illustrations are intended to show that the model entities are sufficient to represent relational entities at a less abstract level than the (theoretical) model.

3.10.3.1 Structures

The unique identification of a relation in terms of a name and a protection level will be managed by the directory system [c.f. § 3.3]. To enhance name uniqueness, all names will include an indication of their creating subject.

The identifiers of entities other than primary relations will be found in the single set of directories. Therefore an indication of "type" of object is required with each identifier.

For example, user identifiers will be maintained in the directories. View and snapshot identifiers will be found there as well. A view will be stored as a "program", giving the definition of its deviation. A snapshot is actually a primary relation which happens to have been formed according to a view definition. Clearly a type indicator is required to allow a single set of directories to allow the accessing of a variety of objects. Such an indicator could be found in an object's identifier.

Both primary and derived relations will possess a permission matrix M_R for relation R [c.f. § 3.4]) which will consist of a set of pairs, (subject,access), for all accesses authorized to a set of subjects.

The "essence" of a view will consist of the definition of its derivation. It is beyond the scope of this report to specify exactly how this definition will be represented.

The descriptor of the primary relation R, D_R, will consist of a set of tuples, including one for every domain in the relation. Each such tuple will contain:

(domain name, role, maximum data length, data type)

where:

domain name identifies the domain;

role indicates in which keys (including prime) this domain takes part, and whether or not the domain is indexed [c.f. § 3.12.3];

maximum data length specifies the largest data element which the domain can accommodate; and

data type will indicate characteristics of the data, such as encoding schemes, or format (i.e., YY/MM/DD).

The first of the descriptor tuples will be a special "status tuple", which will be observable but not modifiable. The system will automatically keep its information up-to-date. It is beyond the scope of this report to define

exactly how the information in the status tuple is represented, but it will include:

current status which can be reserved or not reserved;

current size indicates the current number of tuples;

maximum size indicates the maximum number of tuples; and

historical information will indicate when the relation was last updated.

The values portion of a primary relation, V_R, will consist of a matrix of data values. The data values in a column of the matrix will conform to the length and type specified in the corresponding descriptor tuple.

Figure 3.4 illustrates the data structure which support primary relations.

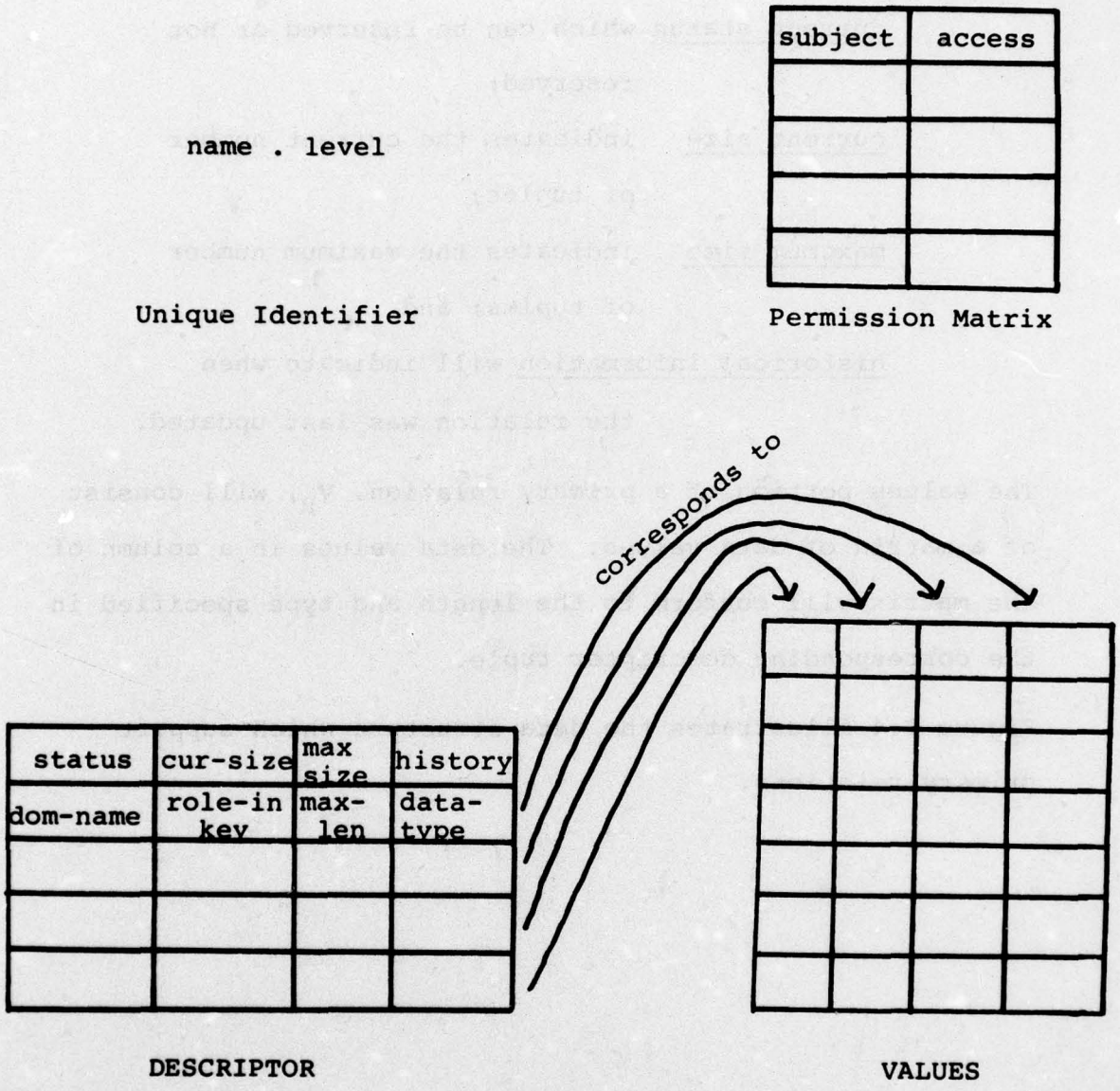


Figure 3.4 Data Structures of Primary Relations

3.10.3.2 Operations

Operations in the model are activities performed in response to a request [c.f. § 3.8], which can be either an observe or modify to a directory, a permission matrix, a descriptor or values.

Table 3.4 lists some basic relational data structures and operations, and indicates the correspondence between them and the eight classes of requests. The purpose of the table is to illustrate that the request classes adequately and reasonably represent basic relational data base operations.

One very important aspect of a relational data base is the set of operators which are in the relational algebra.[15] Examples of these are UNION, INTERSECTION, PROJECTION, RESTRICTION and JOIN. An analysis of these operations will lead to the conclusion that they can be implemented[16] in terms of the basic operations in table 3.4.

[15] E.F. Codd, A Relational Model of Data for Large Shared Data Banks, Communications of the ACM, Volume 13, Number 6, June 1970, 377-387.

[16] D.C. Tsichritzis, On Implementation of Relations, Technical Report CSRG-35, Computer Systems Research Group, University of Toronto, May 1974.

ENTITIES	Basic Operations: CREATE	OBSERVE	APPEND	CHANGE	DELETE	RESERVE	RELEASE
relation name	Q_L^M	Q_L^O		Q_L^M	Q_L^M		
protection level	Q_L^M	Q_L^O			Q_L^M		
access permission matrix	Q_M^M	Q_M^O	Q_M^M	Q_M^M	Q_M^M		
relation status	Q_M^M	Q_D^O					
domain descriptor	Q_D^M	Q_D^O	Q_D^M	Q_D^M	Q_D^M		
domain	Q_D^M	Q_V^O	Q_V^M	Q_V^M	Q_V^M		
tuple	Q_V^M	Q_V^O	Q_V^M	Q_V^M	Q_V^M		
key values	Q_V^M	Q_V^O	Q_V^M	Q_V^M	Q_V^M		
primary relation	Q_D^M	Q_V^O	Q_V^M	Q_V^M	Q_V^M	Q_D^M	Q_D^M
view	Q_D^M	Q_V^O	Q_V^M	Q_V^M	Q_V^M	Q_D^M	Q_D^M

Table 3.4 Basic Relational Data Management Operations

3.10.4 Multi-Level Activities

Important activities such as data sharing and forming derived relations (i.e., views) involve several different protection levels for a given operation.

There is a difficulty in multi-level data sharing in that a subject's observation of a "lower" level relation must not be noticable to a "lower" level subject. Then there can be no "read-reservation" to ensure the consistency of data observed in a lower level relation. Therefore, data must be re-read whenever an intervening modification destroys its consistency. However, this may never be possible if the time to read data is greater than the rate of modification.

Since views [c.f. § 3.3.1] can involve two relations (i.e., JOIN) with different levels, it is not immediately obvious how to assign levels to them. Modifying a view means changing the definition of a view, not any data. Therefore a view is an observation mechanism, making the simple protection property relevant. This suggests that the level of a view must dominate all levels of the relations in its definition. This will ensure authorized observation with the view. The same holds true for snapshots. Therefore a subject will sign-on the system at

the level of the derived relation he wishes to define, at a level dominating all base relations.

3.10.5 Conclusion

Subsection 3.10 has illustrated how the model adequately represents the structures and operations found in the relational approach to data management.

The consistency of the model with the requirements of a relational data base suggests the feasibility of implementing a system based on the model.

3.11 Object Creation

In the Bell-La Padula model object creation is performed by attaching a "leaf" to a "tree" of data objects. The attaching of the leaf requires the appending of an entry (branch) for the object in the parent object, the new model will have an analogous notion of object creation, which is described below.

3.11.1 Data Objects

According to the new model, object creation is performed when a user submits a "create-object" request to the system, naming the object and assigning it a level. This is

modeled by Q_L^M [c.f. table 3.4], which will cause the object's name to be "registered" in the appropriate directory.

Object creation is somewhat more involved when the name is to be registered "lower" than the object. This activity requires a modification to two directories, and a corresponding observation to check for success or failure. This will require that the subject perform object creation at two protection levels. The creation activity will consist essentially of:

(i) sign-on at the "low" level, and register the object in the directory, indicating the "higher" level of the actual object;

(ii) sign-off "low" and sign-on at the "high" level;

(iii) observe the "lower" level directory and check the consistency of the relevant entry;

(iv) register the object at the "high" level, indicating that it's registered at a "lower" level as well; and

(v) initialize[17] the object.

[17] Initialization involves setting the permission matrix to be the tuple: "(creating-subject,modify)", which is required since non-existent objects can not be subject to discretionary access control. Additionally, the "status tuple" must be created [c.f. § 3.10.3.1].

3.11.2 Re-Classifying Objects

An object possesses a certain protection level by virtue of the fact that its name resides in the directory for that level [c.f. § 3.3]. A re-classification of the object would require a "movement" of the name to another directory. The *'-property [c.f. § 3.2] will prohibit this movement to be "down". Therefore if the capability to re-classify objects is made available, only upgrading objects is permissible.

3.12 Hidden Objects

Hidden objects are entities in the model which are not explicitly observable, but rather are more subtle combinations of characteristics of the model. Although some of these entities will be completely invisible to the user, they may constitute a protection threat if they deal with data objects of differing protection levels. These possibilities are explored in this section.

3.12.1 Backup/Recovery

Backup and recovery can be viewed as hidden objects since they can be performed automatically by the system without user awareness. However, they can be consciously invoked by a user.

The main purpose of the backup/recovery subsystem is to allow for the storing and retrieval of copies of data objects in a protected ARCHIVE. This archive is protected because only the backup and recovery processes are able to access it. This is accomplished by fixing the M(ARCHIVE) in the model, so the backup process has append access only, and the recovery process has read-only access.

The backup process will allow appending to the archive a copy of any object to which a subject has access. The recovery process may be invoked for only those objects to which a subject has both observe and modify access. This is because recovery is really a destructive modification. Therefore, a user with only modify access cannot directly recover an object.

Recovery will replace an object with the specified copy of it from the archive. Generations of copies of an object preceeding the last one backed up will be recoverable.

Backup and recovery cannot operate as normal processes. In order to function, backup must run as system high level if it is to observe all relations. Backup would then have to store the relations at the system high level. In order to

read these stored relations, recovery would also have to operate at the system high level. Recovery would then have to be a trusted[18] process to restore any relations that were classified at less than system high level.

3.12.2 Encoding Domains

The main purpose of encoding domains is to serve as a basis for the transformation of representations of data (data compression). For example figure 3.5 illustrates an encoding domain where relative position is a code which is more convenient to manage than the character string it represents.

Since all data (potentially of different protection levels) is maintained in a single encoding domain for any given data type, there are aspects relevant to protection. To make these aspects irrelevant at the model level, all encoding domains will be considered implicit in the "correct" access function δ [c.f. § 3.5].

[18] By "trusted" process is meant one constrained to execute a verified and protected system program.

The conclusion is that the model can satisfactorily represent the usage of encoding domains.

relative position	encoding domain
1	Ottawa
2	Montreal
3	Toronto
	. . .

Figure 3.5 An Example of an Encoding Domain

3.12.3 Indexes

In a relational data base, an "index" is an ordering of the values in a domain of a relation, so tuples may be accessed in that order. Indexes also make for more efficient access to any tuple which can be referenced by that domain.

Two types of indexes can be required:

(i) User-defined indexes may be needed to enable users to obtain more efficient data management, This would be accomplished by anticipating a heavy load of queries and specifying which

domains would make useful indexes. Since an index involves protected data (perhaps duplicating it) protection issues are relevant. When relating indexes to the new model, they will be considered to be part of the relations' values component, V_R [c.f. § 3.10.3], and therefore be protected by standard mechanisms;

(ii) System-defined indexes will automatically index relations whenever system performance warrants it. The system sould monitor performance over a reasonable length of time (e.g., a week), before deciding which pattern of indexes would best suit system performance. Such indexes offer no protection threats since they are invisible and automatic.

If system-defined indexes and the principles by which they are formed are not kept completely invisible, system performance itself could constitute a communication path if it were observable by all subjects.

3.12.4 Race Conditions

"Race conditions" involve the effects of multiple subjects simultaneously requesting access to a certain object. The resolution of race conditions (and deadlock) must not violate the *'-property. Therefore "higher-level" processes must re-issue access requests [c.f. § 3.10.4], so that "lower-level" processes cannot observe effects of their (higher) object access.

3.13 Different Environments

It is necessary to check that the generality and flexibility of the model is adequate to be applicable in very different environments. The applicability of the model to three contemporary environments will be discussed.

3.13.1 A Dedicated DMS

A dedicated data management system is the kind most clearly represented by the model. Then all system mechanisms could serve the criteria and constraints necessitated by protection. The operating system would be essentially split in two:

1) a systems services part; and
2) a security kernel, monitoring access requests.

3.13.2 A Partitioned System

In a partitioned system the DMS is implemented as an application program on a computer system possessing a secure operating system. The operating system must be secure or otherwise the data base could be accessed in an unauthorized manner from other applications, and possibly from the DMS itself.

A complication is that certain required facilities and data may be outside of the protected DMS. The "security perimeter" is the collection of security related functions which ensure that access to protected objects conforms to the requirements of protection [c.f. § 6.1]. It is beyond the scope of this report to define the security perimeter, other than to point out that it must conform to the protection requirements of the model.

3.13.3 A Network of Protected Systems

A network consists of a set of interconnected protected systems. The main unique characteristic of the network is that the subject of an access need not be in the same node of the system as the target object. Then the request, and the resulting decision can be passed through a number of combinations of connecting nodes. The access method δ will establish if the object exists, and the best route to access it.

A subject's current protection attribute is maintained with the request through all intermediate nodes, to allow for non-discretionary access control at the target node. The subject's identity must be maintained with the request as well, to allow for discretionary access control.

As the protected data flows back through intermediate nodes, any access to the flowing data must not violate non-discretionary or discretionary control requirements.

The communication channels between nodes must of course be extremely secure so that information is not tapped off. Additionally, security is required to ensure that a subject's identity may not be changed and falsified during its transmission from one node to another. The problem of protecting these links is, however, beyond the scope of this report. In a like manner, the links into a dedicated system from remote terminals must be secure.

Thus, the model is considered adequate for a protected network environment.

SECTION IV

CONSTRUCTING THE NEW MODEL

4.0 Introduction

The Bell-La Padula model was accepted as a basis for the new model to take advantage of:

(i) the foundation work done;

(ii) recognized terminology; and

(iii) proven techniques.

An important consideration in the construction of the new model was to make it as small and simple, as possible. This would enhance the clarity of the model elements and operations, and simplify demonstrating the correspondence between specifications and the model. All model entities are considered to be essential in that their removal would impact the model in an undesirable way.

The new model was constructed from:

(i) essential Bell-La Padula model entities;

(ii) changed Bell-La Padula model entities;

(iii) extended Bell-La Padula entities; and

(iv) new entities.

4.1 Essential Bell-La Padula Model Elements

A summary of the Bell-La Padula model entities which are considered essential for the new model follows [c.f. § 2.1]:

(i) A SUBJECT is an active entity;

(ii) An OBJECT is a passive entity;

(iii) An ACCESS is a type of activity performed on an object by a subject;

(iv) CLASSIFICATIONS are a fully ordered set;

(v) CATEGORIES are partially ordered with respect to subsetting; and

(vi) all inputs are in the form of REQUESTS.

4.2 Modified Bell-La Padula Entities

A list of modified Bell-La Padula model entities and a summary on how they were changed for the new model is given in this subsection.

(i) The simple security property is changed to use the subject's <u>current</u> security level rather than <u>maximum</u> level in specifying non-discretionary authorization of observation [c.f. § 3.2].

(ii) The *-property is changed to use the subject's <u>current</u> security level rather than the levels of the accessed objects in specifying non-discretionary authorization of modification [c.f. § 3.2].

(iii) The organization of the directory objects is changed to offer more flexible and powerful capabilities [c.f. § 3.3].

(iv) The set of Access Modes is redefined to consist solely of observe and modify [c.f. § 3.7].

(v) The set of requests is changed to reflect the change in model entities [c.f. § 3.8].

4.3 Extended Bell-La Padula Entities

A list of extended entities, and a summary of how they were extended is given in this subsection.

(i) A security level was extended to include an integrity level, producing a protection level [c.f. § 3.2]. Correspondingly, the security level functions (f_s,f_o,f_c) were extended to produce (Π_s,Π_o,Π_c). The simple protection and *'-properties followed.

(ii) The directory system was extended to allow searching for objects, and the potential determination of <u>all</u> dominance relationships [c.f. § 3.3].

(iii) An object in the protected DMS was defined to consist of three parts: a permission matrix, a descriptor and a value set [c.f. § 3.4].

(iv) Protection requirements were extended to apply to execute access [c.f. § 3.6].

(v) The tranquility principle was extended to apply to subjects as well as objects [c.f. § 3.9].

4.4 New Entities

Summaries of the entities which are newly defined for the new model are given in this subsection.

(i) Integrity levels, level functions and properties are defined to be dual in nature to the security entities, and are relevant to authorized modification control (not "correctness" or "soundness") [c.f. § 3.1].

(ii) The set of lattice directory functions (λ) will be used to search directories [c.f. § 3.3].

(iii) The direct access function (δ) makes object access more flexible, and is identified as the mechanism enforcing the protection requirements [c.f. § 3.5].

4.5 Conclusion

The set of essential changes to the Bell-La Padula model is now complete. This section has presented them in summarized form.

SECTION V

THE FORMAL DESCRIPTION OF THE NEW MODEL

5.0 Introduction

The finalized model can now be formally described. The style of presentation will be similar to that of the Bell-La Padula report.[1] Section IV summarizes the entities which will constitute the new model. The model presentation in this section will consist of a table of model elements, relevant terminology, and a set of protection properties.

5.1 Elements of the Model

Table 5.1 lists the elements in the model and gives the symbols that represent them. The semantics are included as well, but are highly summarized. Although the relevant terminology is given in section 5.2, it may be necessary to refer to a previous section for a more thorough explanation.

[1] D.E. Bell and L.J. La Padula, Secure Computer System: Unified Exposition and Multics Interpretation, Appendix.

TABLE 5.1

ELEMENTS OF THE MODEL

SET	ELEMENTS	SEMANTICS	SECTION
S	$\{s_1, s_2, \ldots, s_n\}$	subjects: processes (programs in execution)	3.9
A	{observe, modify}	access attributes: types of access subjects are able to perform; observe: extract or use information from a model element; modify: alter a model element (without observation)	3.7
V	$\{V_1, V_2, \ldots, V_m\}$, V_i = {data elements}, $i = 1, \ldots, m$	value-sets: sets of data elements; e.g. relations; programs and messages	3.4
D	$\{D_1, D_2, \ldots, D_m\}$, D_i = {elements describing V_i}, $i = 1, 2, \ldots, m$	descriptors: sets of descriptive elements, where each set describes a value-set	3.4

SET	ELEMENTS	SEMANTICS	SECTION
M	$\{M_1, M_2, \ldots, M_m\}$, $M_i = \{(S,a): S \in S \text{ and } a \in A\}$, $i = 1, \ldots, m$ where M_i indicates that certain subjects are authorized to access V_i, D_i and M_i itself, in given modes.	permission matrices:	3.4
O	$\{O_1, O_2, \ldots, O_m\}$, $O_i = (M_i, D_i, V_i)$, where: $M_i \in M$, $D_i \in D$, $V_i \in V$, $i = 1, 2, \ldots, m$	objects: the inactive entities in a system: data, relations, programs and messages	3.4
ID	$\{O_{id_1}, O_{id_2}, \ldots, O_{id_m}\}$ O_{id_j} = a character string associated with object O_j, $j = 1, 2, \ldots, m$	identifiers:	3.3
C	$\{C_1, C_2, \ldots, C_t\}$, $C_1 > C_2 > \ldots > C_t$	classifications: clearance level of a subject; classification of an object	2.1
K	$\{K_1, K_2, \ldots, K_r\}$	categories: special access privileges	2.1
P	$P_1, P_2, \ldots, P_p$ with partial ordering relation $\succ'$, $P_i = (C_i, K_i, C_i', K_i')$, where: $C_i, C_i' \in C$ and $K_i, K_i' \subseteq K$	protection levels: the protection attributes, each composed of a security level and an integrity level. The "protection dominance" ($\succ'$) relation is described in subsection 5.2.	3.2

SET	ELEMENTS	SEMANTICS	SECTION
Π	an element (Π_s, Π_o, Π_c) is in $\Pi \subseteq P^S \times P^O \times P^S$ if and only if for each $S_i \in S$: $C_c \leq C_s$ and $K_c \subseteq K_s$ and $C_c' \leq C_s'$ and $K_c' \subseteq K_s'$, where: $\Pi_s(S_i) = P_s = (C_s, K_s, C_s', K_s') \in P$ $\Pi_o(O) = P_o = (C_o, K_o, C_o', K_o') \in P$ and $(O \in \mathcal{O}$ or $O \in \mathcal{ID})$ $\Pi_c(S_i) = P_c = (C_c, K_c, C_c', K_c') \in P$	protection level vectors: Π_s: subject protection level limit function Π_o: object protection level function Π_c: current protection level function	3.2
L	$\{L_j\}$ $L_j = \{(O_{id}, \text{code}): O_{id} \in \mathcal{ID}$ and $(\Pi_o(O) = P_j$ or $\Pi_o(O_{id}) = P_j)\}$, $P_j \in P$, $j \in \{1, 2, \ldots, p\}$; L is a partition of all object identifiers and their associated codes into directories; L is (more formally) an equivalence class[2] of identifiers with the same protection level;	set of directories:	3.3

[2] I.N. Herstein, Topics in Algebra, Blaisdell Publishing Company, Toronto, Canada, 1965, page 4.

SET	ELEMENTS	SEMANTICS	SECTION

code = zero if $\Pi_o(O) = P_j$ and there is no $i \in \{1,2, \ldots, p\}$, $i \neq j$, such that $P_j \succ' P_i$ and $(O_{id}, P_j) \in L_i$;

code = $-P_i$ when there is such an i;

code = P_k if $\Pi_o(O) = P_k$, $k \neq j$, $k \in \{1,2, \ldots, p\}$ and $P_k \succ' P_j$.

code indicates when an object's identifier is registered in a directory at a level "lower" than the actual object.

L	(L,λ), where:	lattice directory system:	3.3

$\lambda = \{\lambda_{\prec'}, \lambda_{\succ'}\}$

is a set of lattice directory functions,

where:

$\lambda_{\prec'}(P_i) = \{(P_j):L_j \in L, P_j \prec' P_i$ and $j \in \{1,2, \ldots, p\}\}$

is a list of dominated directory protection levels; and

$\lambda_{\succ'}(P_i) = \{(P_j):L_j \in L, P_j \succ' P_i,$ and $j \in \{1,2, \ldots, p\}\}$

is a list of dominating protection levels;

$P_i \in P$, $i = 1,2, \ldots, p$.

SET	ELEMENTS	SEMANTICS	SECTION
Q	$\cup \{Q_L^O \; Q_L^M \; Q_M^O \; Q_M^M \; Q_D^O \; Q_D^M \; Q_V^O \; Q_V^M\}$ where:	access requests: these are classes of requests each of which is a set of requests consisting of either an observe or modify to one of: a directory L_i; a permission matrix M; a descriptor, D; or a value set, V.	

$Q_L^O = \{q:$ q is an observe access request to some $L_i \in L\}$

$Q_L^M = \{q:$ q is a modify access request to some $L_i \in L\}$

$Q_M^O = \{q:$ q is an observe access request to some $M \in M\}$

$Q_M^M = \{q:$ q is a modify access request to some $M \in M\}$

$Q_D^O = \{q:$ q is an observe access request to some $D \in D\}$

$Q_D^M = \{q:$ q is a modify access request to some $D \in D\}$

$Q_V^O = \{q:$ q is an observe access request to some $V \in V\}$

$Q_V^M = \{q:$ q is a modify access request to some $V \in V\}$

SET	ELEMENTS	SEMANTICS	SECTION
δ	is a function:	direct access function:	3.5
(i)	$\delta(S,P_i,q) = L_i$	(i) To access directories;	
		if: $P_i \in \lambda_{\prec}(P_c)$ when $q \in Q_L^O$,	
		or $P_i \in \lambda_{\succ}(P_c)$ when $q \in Q_L^M$,	
		$S \in \mathcal{S}$, $P_i \in \mathcal{P}$, $\Pi_c(S) = P_c$, and $L_i \in L$;	
		= 'FAILURE' if not.	
(ii)	If: $S \in \mathcal{S}$, $\Pi_c(S) = P_c$, $O_i \in \mathcal{O}$,	(ii) To access objects;	
	$L_j \in L$, $L_j(O_i) = O_{id_i} \in \mathcal{ID}$; and		
	if: $a \in A$ is the access involved in $q \in \mathcal{Q}$, and $(S,a) \in M_i$;	(discretionary access control)	
	Then: $\delta(S,L_j(O_i),q) = M_i$[3]	is a permission matrix access;	
		if: $\Pi_o(O_i) \in \lambda_{\prec}(P_c)$ when $q \in Q_M^O$,	
		or $\Pi_o(O_i) \in \lambda_{\succ}(P_c)$ when $q \in Q_M^M$,	
		$= D_i$ a descriptor access;	
		if: $\Pi_o(O_i) \in \lambda_{\prec}(P_c)$ when $q \in Q_D^O$,	
		or $\Pi_o(O_i) \in \lambda_{\succ}(P_c)$ when $q \in Q_D^M$,	
		$= V_i$ a value set access;	
		if: $\Pi_o(O_i) \in \lambda_{\prec}(P_c)$ when $q \in Q_V^O$,	
		or $\Pi_o(O_i) \in \lambda_{\succ}(P_c)$ when $q \in Q_V^M$;	
		= 'FAILURE' otherwise.	

[3] Object creation [c.f. § 3.11.1] is modeled by:

(i) $\delta(S,P_k,q) = L_k$ where $q \in Q_L^M$ and $\Pi_c(S) = P_k$;

(ii) $\delta(S,P_j,q) = L_j$ where $q \in Q_L^M$ and $\Pi_c(S) = P_j$;

(iii) $\delta(S,L_j(O_i),q) = (M_i,D_i,$ null) where $q \in Q_M^M$, $\Pi_c(S) = P_j$, $M_i = (S,$ modify) and D_i = status-tuple [c.f. § 3-11].

5.2 Terminology

The terminology in this subsection will be relevant to those model elements in table 5.1 which are not self-explanatory.[4]

Suppose that U is a set and R is a binary relation defined on U, with elements of U denoted by small leters a, b, c, ..., etc.

reflexive: R is reflexive if xRx for each x in U.

antisymmetric: R is antisymmetric if xRy and yRx implies x = y for each x and y in U.

transitive: R is transitive if xRy and yRz implies xRz for each x, y and z in U.

partial ordering relation R: R is a partial ordering relation if R is reflexive, antisymmetric, and transitive.

protection dominance, $\succ'$ [c.f. § 5.1]:

$P = \{P_1, P_2, \ldots, P_p\}$, where: $P_i = (C_i, K_i, C_i', K_i')$; C_i and C_i' are in C; K_i and K_i' are subsets of K.

Define the relation $\succ'$ on P as follows:

$(P_i, P_j) \in \succ' \equiv P_i \succ' P_j \equiv (C_i, Ki, C_i', K_i') \succ' (C_j, K_j, C_j', K_j')$

$\equiv P_j \prec' P_i \equiv P_i$ dominates P_j with respect to protection

[4] Many of the definitions are taken from the appendix in the report: Secure Computer System: Unified Exposition and Multics Interpretation.

$P_i \succ' P_j$ if and only if (i) $C_i \geq C_j$;

(ii) $K_i \geq K_j$;

(iii) $C_i' \leq C_j'$; and

(iv) $K_i' \leq K_j'$.

Theorem 5.1: $\succ'$ is a partial ordering.

Proof: 1) $\succ'$ is a reflexive since $P_i \succ' P_i$,

since (i) $C_i \geq C_i$;

(ii) $K_i \geq K_i$;

(iii) $C_i' \leq C_i'$; and

(iv) $K_i' \leq K_i'$.

2) $\succ'$ is antisymmetric since [$P_i \succ' P_j$ and $P_j \succ' P_i$] implies $P_i = P_j$, since:

(i) $C_i \geq C_j$ and $C_j \geq C_i \Rightarrow C_i = C_j$;

(ii) $K_i \geq K_j$ and $K_j \geq K_i \Rightarrow K_i = K_j$;

(iii) $C_i' \leq C_j'$ and $C_j' \leq C_i' \Rightarrow C_i' = C_j'$; and

(iv) $K_i' \leq K_j'$ and $K_j' \leq K_i' \Rightarrow K_i' = K_j'$

3) $\succ'$ is transitive since:

(i) $C_i \geq C_j$ and $C_j \geq C_k \Rightarrow C_i \geq C_k$;

(ii) $K_i \geq K_j$ and $K_j \geq K_k \Rightarrow K_i \geq K_k$;

(iii) $C_i' \leq C_j'$ and $C_j' \leq C_k' \Rightarrow C_i' \leq C_k'$; and

(iv) $K_i' \leq K_j'$ and $K_j' \leq K_k' \Rightarrow K_i' \leq K_k'$.

$\therefore$ $\succ'$ is a partial ordering.

partially ordered set: P is a partially ordered set with to $\succ'$, since $\succ'$ is reflexive, antisymmetric, and distributive.

upper bound: If P is a partially ordered set, an element $P_u \in P$ is called an upper bound of P if $P_u \succ' P_i$ for all $P_i \in P$.

least upper bound (l.u.b.): An upper bound P_l of P is said to be the least upper bound (l.u.b.) of P if every upper bound P_u of P satisfies $P_u \succ' P_l$.

lower bound: If P is a partially ordered set, an element $P_b \in P$ is called a lower bound of P if $P_b \prec' P_i$ for all $P_i \in P$.

greatest lower bound (g.l.b.): A lower bound P_g of P is said to be the greatest lower bound (g.l.b.) of P if every lower bound P_b of P satisfies P_b of P satisfies $P_b \prec' P_g$.

lattice: A lattice is a partially ordered set such that any two elements of it possess both a l.u.b. and a g.l.b.[5]

[5] D.E. Rutherford, Introduction to Lattice Theory, Oliver & Boyd Ltd., Edinburgh and London, 1965, page 4.

Theorem 5.2: P is a lattice with respect to $\succ'$.

Proof [c.f. § 5.1]:

Let P_i and P_j be elements of P.

1) $P_k = (\max\{C_i, C_j\}, K_i \cup K_j, \min\{C_i', C_j'\}, K_i' \cap K_j')$
is the l.u.b. of $\{P_i, P_j\}$ since:

a) $P_k \in P$ since (i) $\max\{C_i, C_j\}, \min\{C_i', C_j'\} \in C$
and (ii) $K_i \cup K_j, K_i' \cap K_j' \subseteq K$;

b) P_k is an upper bound of $\{P_i, P_j\}$
since $P_k \succ' P_i$ and $P_k \succ' P_j$ by construction;

c) P_k is the least upper bound since if it
were not, there would be an element
$P_x \in P$ such that $P_k \succ' P_x \succ' P_i$.
Then $C_k \geq C_x \geq C_i$ and $C_k \geq C_x \geq C_j$
But since $C_k = \max\{C_i, C_j\}$ this would
produce a contradiction unless $C_x = C_k$.
Similar arguments for the other
components of P_k will produce the
conclusion that $P_x = P_k$. Then P_k is the
l.u.b. for P.

2) $P_b = (\min\{C_i, C_j\}, K_i \cap K_j, \max\{C_i', C_j'\}, K_i' \cup K_j')$
is the g.l.b. of $\{P_i, P_j\}$ as the result of
an argument which is dual to that in (1).

P is a lattice since any two elements of it
have a g.l.b. and a l.u.b.

AD-A035 256 CANADIAN COMMERCIAL CORP OTTAWA (ONTARIO) F/G 9/2
A MODEL OF A PROTECTED DATA MANAGEMENT SYSTEM.(U)
JUN 76 M J GROHN F19628-76-C-0025
UNCLASSIFIED ESD-TR-76-289 NL

2 OF 2
AD
A035256

END

DATE
FILMED
3-77

the notation A^B: Suppose A and B are sets. The notation A^B denotes the set of all functions from B to A. For example, suppose A = {a,b} and B = {1,2}; then A^B consists of

$f_1 = \{(1,a), (2,b)\}$,

$f_2 = \{(1,b), (2,a)\}$,

$f_3 = \{(1,a), (2,a)\}$, and

$f_4 = \{(1,b), (2,b)\}$.

cartesian product: Suppose A and B are sets. The cartesian product of A and B, denoted A x B, is defined by

$A \times B = \{(a,b): a \in A \text{ and } b \in B\}$.

5.3 Protection Properties

"Protection" is a term which refers to the ability of the model to have its activities (object access) conform to a given policy. A "property" refers to those attributes or conditions which pertain to an access and give relationship between a subject's protection level, and that of an object. The four following properties of the new model cause object access to conform to Department of Defense policy.

To describe the protection properties consisely, notation from table 5.1 will be used. Let $\underline{a} \in A$ be an access.

5.3.1 Simple-Protection

The new model possesses the <u>simple-protection property</u> iff no access to a model element violates the following condition:

$$(\underline{a} = \text{observe}) \Rightarrow \Pi_c(S) \succ' \Pi_o(O).$$

5.3.2 *'-Property

The new model possesses the <u>*'-property</u> iff no access to a model element violates the following condition:

$$(\underline{a} = \text{modify}) \Rightarrow \Pi_c(S) \prec' \Pi_o(O).$$

Note that the simple protection property and the *'-property require that for a destructive modify (replace) involving both observation and modification:

$$(\text{observe and modify}) \Rightarrow \Pi_c(S) = \Pi_o(O)$$

by the antisymmetric property of $\succ'$.

5.3.3 Tranquility Property

The new model possesses the <u>tranquility property</u> iff:

$\Pi_o(O)$ = constant, for any object O; and

$\Pi_c(S)$ = constant, for active subject S.

5.3.4 Discretionary Protection Property

The new model possess the <u>discretionary protection property</u> iff no access to an object violates the following condition:

$$(\delta(S, L_j(O_i), q) = O) \Rightarrow (S,a) \in M_i$$

(The meaning of the symbols are given on page 5-7 of table 5.1.)

SECTION VI

ASSERTIONS REGARDING PROTECTION

6.0 Introduction

Protection has been defined to refer to the ability of a model to have its activities (such as object access) conform to a given policy [c.f. § 5.3]. This is achieved by defining properties which are not to be violated by actions of the system. Examples of these are given in subsection 5.3.

The parameters of the protection properties are: 1) the observation control attribute (security level [c.f. § 2.2]); and 2) the modification control attribute (integrity level [c.f. § 3.1]), both possessed by the subject and the object involved in an access. These two mechanisms can conveniently be defined to be duals in their roles as protective mechanisms [c.f. § 3.2].

The fundamental principles causing object access in the model to conform to U.S. Department of Defense policy are given as a set of axioms in § 6.1. This choice of axioms is justified in § 6.2.

6.1 Axioms of Protection

The fundamental principles of protected object access are given in this subsection as six axioms. The symbols used in the axioms are:

f_c = a function mapping active subjects into observation control attributes;

f_o = a function mapping objects into observation control attributes;

g_c = a function mapping active subjects into modification control attributes;

g_o = a function mapping objects into modification control attributes; and

M = the access permission matrix of the Bell-La Padula model [c.f. § 2.1], where M_{ij} records the modes in which subject S_i is permitted to access object O_j.

6.1.1 Axiom I: Direct Disclosure Control

Subject observes object implies f_c(subject) dominates f_o(object) with respect to observation control attributes.

6.1.2 Axiom II: Indirect Disclosure

Subject modifies object implies f_o(object) dominates f_c(subject) with respect to observation control attributes.

6.1.3 Axiom III: Direct Modification

Subject modifies object implies g_c(subject) dominates g_o(subject) with respect to modification control attributes.

6.1.4 Axiom IV: Indirect "Contamination"

Subject observes object implies g_o(object) dominates g_c(subject) with respect to modification control attributes.

6.1.5 Axiom V: Tranquility Principle

A) f_c(subject) and g_c(subject) are constant for any subject; and

B) f_o(object) and g_o(object) are constant for any object.

6.1.6 Axiom VI: Discretionary Access Control

A) subject observes object implies 'observe' is registered in $M_{(subject,object)}$.

B) subject modifies object implies 'modify' is registered in $M_{(subject,object)}$.

6.2 Justification of the Axioms

The definition of the observation and modification control attributes includes a classification or clearance level,

and a set of formal categories [c.f. § 2.1 and § 3.1]. This corresponds to the situation in a military or governmental environment, where people and documents can receive both types of formal security designations.[1]

The axioms in § 6.1 involve security and integrity levels rather than protection levels, since protection is not a primitive mechanism of the model, but rather is a composite of security and integrity [c.f. § 3.2]. The axioms are intended to be relevant to primitive functions of the model.

The axioms serve as a useful set of principles on which to base access authorization. They will cause object access to conform to U.S. Department of Defense policy in the following ways:

Axiom I will prevent the direct unauthorized disclosure of information by requiring that the observation control attribute of a subject "dominate" that of an object. That is, the non-discretionary requirements for observation are met.

Axiom II will prevent indirect disclosure of information by not allowing information of a certain level to be copied to a lower level where axiom I could be violated.

[1] Secure Computer System: Unified Exposition and Multics Interpretation, page 9.

Axiom III will prevent direct unauthorized modification of information by requiring that the modification control attribute of a subject "dominate" that of an object. That is, the non-discretionary requirements for modification are met.

Axiom IV will prevent indirect "contamination", by ensuring that unauthorized data can not be copied into authorized data. As well, since observe and execute are equivalent [c.f. § 3.6], this axiom will prevent the indirect threat of an authorized process executing an unauthorized program.

Axiom V will make the activities and results of processes more predictable, and eliminate the risk of the nature of level varying allowing indirect communication paths to be established.

Axiom VI will enfore discretionary control policy by requiring that every access of an object be explicitly authorized.

SECTION VII

CONCLUSIONS

7.0 Conclusions

The model described in section V adequately represents the key entities in a data management system, with the emphasis on the protected access of objects by subjects. This conclusion follows from the range of special consideratons in section III. In particular, the relational approach to data management [c.f. § 3.10] causes no difficulty for the model.

Each entity in the model is essential, since the model would be undesirably impacted if any one of them were removed.

Since the model conforms to the six axioms presented in § 6.1, it embodies Department of Defense access policies, and can serve satisfactorily as the basis for a design of a protected data management system.

APPENDIX I

IMPLEMENTING THE LATTICE DIRECTORY FUNCTIONS

A1.0 Introduction

The directory of a protected (with respect to the duals, security and integrity) data management system is segmented according to the protection attributes of the data. The segments are isolated points on the lattice of all possible protection attributes.

This lattice is based on a partial order, protection dominance [c.f. § 3.2] $\not\succ$. In searching for information to release (if permitted) to a user, the system must scan not only the directory segment at the same protection node point as the user but all directories at nodes dominated by the user's node.

To facilitate the comprehension of this concept, a function, $\lambda_{\prec}$, was introduced. It takes as its argument the level of the node of the user and returns all the currently active levels of directories at nodes which the user's node dominates. Another function $\lambda_{\succ}$ returns the levels of nodes dominating the user's node.

This paper explores a possible implementation of these functions based on a simple technique for identifying each of the nodes in the complete theoretical lattice. The method of identifying nodes will be found to correspond directly to the properties of protection from which the lattice arises.

A1.1 Node Identification

The protection attribute of an object consists of four elements: the security level, the set of security categories, the integrity level, and the set of integrity categories. The security levels and the integrity levels both are fully ordered. The subsets of the set of security categories and the set integrity categories are only partially ordered.

A subset of a set may be represented by an incidence vector. The set is given an arbitrary ordering and the presence or absence of each element of the set in the subset may be recorded by a 1 or 0 in the corresponding position of the incidence vector. The vector, then, is a series of 0's and 1's and this may be interpreted as a binary number. The decimal equivalent may easily be determined. Let d(s) be the decimal representation of a logical string, s.

It should be noted in passing that a necessary (but not sufficient) condition for a subset, A, to be itself contained in a subset, B, (i.e., $A \subseteq B$) is that:

d(Incidence vector of A)$\leq$d(Incidence vector of B).

Let S be the set of security levels

Let C(S) be the cardinality of this set. Let C(S)=a

Let I be the set of integrity levels

Let C(I)=b

Let T be the set of security categories

Let $t \subseteq T$ be a subset of security categories

Let $C(T)=x$

Let U be the set of integrity categories

Let $u \subseteq U$ be a subset of integrity categories

Let $C(U)=y$

Let a protection attribute be

(s,t,i,u) where $s \in S$, $t \subseteq T$, $i \in I$, $u \subseteq U$.

We identify the lattice node corresponding to this protection attribute by

(j,k,l,m) where

$j \in \iota a$; ιa is 1,2,...a corresponding to security level

k= d(incidence vector of t); k is decimal representation of subset t

$l \in \iota b$; ιb is 1,2,...b corresponding to integrity level

m= d(incidence vector of u); m is decimal representation of subset u

The protection lattice has, as does every lattice, a minimal and a maximal element. A minimal element is one which is dominated by every other element. A maximal element is one which dominates every other element. These two, with their node identifications, are given below.

	Protection Attribute	Node Identification
Minimal	$(s', \emptyset, i'', U)$	$(1, 0, b, 2^y - 1)$
Maximal	$(s'', T, i', \emptyset)$	$(a, 2^x - 1, 1, 0)$

where s' is least element of S

s" is greatest element of S

i' is least element of I

i" is greatest element of I

Notice that the protection system requires that dominance, $\succcurlyeq'$, be defined:

$$(s_1, t_1, i_1, u_1) \succcurlyeq' (s_2, t_2, i_2, u_2) \text{ iff}$$
$$s_1 \geq s_2,\ i_1 \leq i_2,\ t_1 \supseteq t_2,\ u_1 \subseteq u_2$$

A1.2 Algorithms

Two algorithms are presented:

1) To determine how many nodes there are in the theoretical sublattice dominating or dominated by a particular node.
2) To determine, given two nodes, whether one dominates the other. This is equivalent to determining whether one is in either sublattice defined by the other as mentioned above.

A1.3 Sublattices

To approach this problem we consider the complete lattice. It may be seen that the lattice exists between:

$$(1,0,b,2^{Y}-1) \text{ and } (a,2^{X}-1,1,0)$$

There are (a-1) variations for the first parameter to move from 1 to a; (b-1) variations for the third parameter to move from b to 1.

The second and fourth parameters cause more difficulty.

In moving from a full set of x items to an empty set through all possible subsets, we can divide the process into stages. The first stage consists of considering all those subsets missing just 1 item. Then those missing 2 items from the second stage etc.. That is there are x stages for the second parameter and y for the fourth.

We define a <u>level</u> of a node to be the total number of variations and stages which must be passed through to get to the maximal element.

Let h (node) be the level of that node.

Then $h\ (1,0,b,2^{Y}-1) = (a-1)+(b-1)+y=a+x+b+y-2$ and $h(a,2^{X}-1,1,0)=0$

The level of an arbitrary node

$$h(j,k,l,m) \text{ is } (a-j)+(x-q(k))+(l-1)+q(m)$$

where q(d(s)) is the number of 1's in S.

The number of nodes in the lattice is the number of different combinations of the variations in the initial conditions. In the full lattice there are:

$$a \times 2^x \times b \times 2^y = ab2^{x+y} \text{ points}$$

At any intermediate point, the number of nodes in the lattice from the point in question to either the minimal element or the maximal element can easily be determined. Given the point (j,k,l,m) the sublattices to the minimal element and to the maximal element have nodes as follows:

$$\text{To } (1,0,b,2^y-1);\ j \times 2^{q(k)} \times (1+b-l) \times 2^{y-q(m)}$$
$$\text{To } (a,2^x-1,l,0);\ (1+a-j) \times 2^{x-q(k)} \times 1 \times 2^{q(m)}$$

A1.4 Comparisons

Two arbitrary nodes (j_1,k_1,l_1,m_1) and (j_2,k_2,l_2,m_2) may either be comparable (with respect to the partial ordering) or not. If they are this is equivalent to saying that the least node which dominates both is one of them (in fact, the greater) and the greatest node which each dominates is the other, (the lesser). If they are not comparable then the least upper bound and the greatest lower bound are each nodes other than the two which are being considered. The least upper bound and greatest lower bound define the smallest sublattice in which both nodes co-exist.

To determine whether two nodes are comparable (i.e., one dominates the other) it is necessary to determine first $d'(k_1), d'(k_2), d'(m_1), d'(m_2)$ where d' is the inverse function corresponding to d. That is, d' converts an integer to a binary number, and then treats the result as a binary or logical string.

Then, (j_1,k_1,l_1,m_1) and (j_2,k_2,l_2,m_2) are comparable if and only if one of the two following compound conditions hold:

$$j_1 \leq j_2$$

and $l_1 \geq l_2$

and $d'(k_1) = (d'(k) \wedge d'(k_2))$

and $k'(m_1) = (d'(m_1) \vee d'(m_2))$

in which case $(j_1,k_1,l_1,m_1) \prec (j_2,k_2,l_2,m_2)$

or else

$$j_1 \geq j_2$$

and $l_1 \leq l_2$

and $d'(k_1) = (d'(k_1) \vee d'(k_2))$

and $d'(m_1) = (d'(m_1) \wedge d'(m_2))$

in which case $(j_1,k_1,l_1,m_1) \succ (j_2,k_2,l_2,m_2)$

A1.5 The Functions λ_α , λ_ρ

To determine, from among all the active nodes, which are dominated by a particular node a two phase process is suggested. It is assumed that a list of all active nodes (those nodes for which there is a directory) exists as an n by 4 matrix, composed of the identifications of these active nodes. (This is called the master directory in § 3.3),

Given a reference node identifier, (j,k,l,m), the first phase is to determine from the list of active nodes a subset of candidate node identifiers. This is done by obtaining a row mask, R, over the matrix determined by, in the case of λ_α , (R_α=(Col 1≤j) ∧ (Col 2≤k) ∧ (Col 3≥l) ∧ (Col 4≥m)) by using R_α to reduce the list into a sublist which is, in effect, a first approximation of the required list of active nodes dominated by (j,k,l,m), it is then possible to proceed to the second phase.

The second phase is to consider each of the remaining identifiers and to check that:

$$d'(\text{Col } 2) = d'(\text{Col } 2) \wedge d'(k)$$

and $$d'(\text{Col } 4) = d'(\text{Col } 4) \vee d'(k)$$

Using these last conditions as a further mask on the sublist, the result is the set of node identifiers dominated by the reference node.

To find all active nodes dominating the reference node, it is necessary only to reverse all the signs above.

A1.6 *APL* Implementation

The concepts considered here are very readily implemented in the (I.P. Sharp) *APL* programming language. In particular:

1) d(s) becomes 2⊥S
2) d'(N) becomes ((⌈2⍟N)⍴2)⊤N
3) q(N) becomes +/((⌈2⍟N)⍴2)⊤N

APPENDIX II

NORMALIZATION OF RELATIONS

A2.0 Introduction

Normalization is a step-by-step reversible process of replacing a given collection of relations by successive collections in which the relations have a progressively simpler and more regular structure.[1] It is always a decomposition of one relation into many. The objectives of normalization are:

1) To perform a function by means of a simpler collection of relational operations than would otherwise be necessary;
2) To free the collection of relations from undesirable insertion, update and deletion (functional) dependencies; and
3) To reduce the need for restructuring the collection of relations as new types of data are introduced.

[1] E.F. Codd, Normalized Data Base Structure: A Brief Tutorial, IBM Research Report RJ935, San Jose, California, November 1971.

A2.1 Functional Dependence

The concept of functional dependence is vital to the understanding of normalization. Domain B of relation R is functionally dependent on domain A of R if each value in A never has more than one value in B associated with it under R. The notation for this is:

$$R.A \rightarrow R.B$$

Examples of undesirable functional dependencies are:

1) an insertion anomaly, which exists when data cannot be inserted into a relation because the prime key value is not available;
2) a deletion anomaly, which exists when deleting a tuple causes the loss of more data than is desired; and
3) an update anomaly, which exists when updating a domain value in a tuple requires multiple tuple updates to maintain the consistency of the information.

Suppose that A, B, C are three distinct domains of a relation R (hence R is of degree 3 or more). Suppose that all three of the following time-independent conditions hold:

R.A→R.B, R.B↛R.A,

R.B.→R.C.

From this we may conclude that two other conditions must hold:

R.A.→R.C. R.C.↛R.A.

and we may represent the entire set of conditions on A, B, C as shown in the following diagram. Note that R.C.→R.B. is neither prohibited nor required.

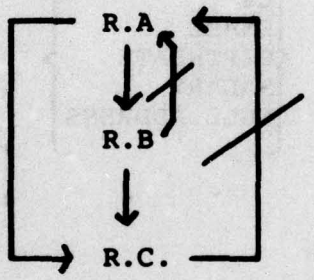


Figure A.2.1 Transitive Dependence of C on A under R

In such a case we say that C is transitively dependent on A under R. In the special case where R.C.→R.B. also, both B and C are transitively dependent on A under R.

A2.2 Example of Normalization of Relations

The following structured file is used to illustrate the three normalization processes.

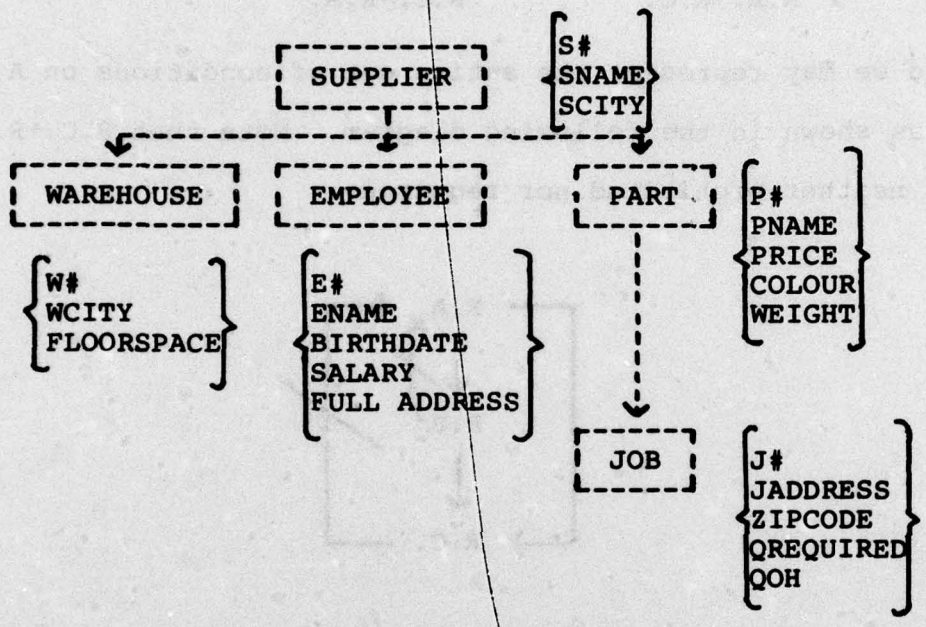


Figure A.2.2 A Structured File of Unnormalized Relations.

A2.2.1 First Normal Form (Flat File)

A relation is in first normal form, if every domain in the relation is a simple domain (i.e., no domain is a relation of degree greater than 1).

Transformation to First Normal Form

S(S#, SNAME, SCITY, W*, E*, P*)

W(W#, WCITY, FLOORSPACE)

E(E#, ENAME, BIRTHDATE, SALARY, FULLADDRESS)

P(P#, PNAME, PRICE, COLOUR, WEIGHT, J*)

J(J#, JADDRESS, ZIPCODE, QREQUIRED, QOH)

First normal form relations (flat Files)

S'(S#, SNAME, SCITY)

SW'(S#, W#, WCITY, FLOORSPACE

SE'(S#, E#, ENAME, BIRTHDATE, SALARY, FULLADDRESS)

SP'(S#, P#, PNAME, PRICE, COLOUR, WEIGHT)

SPJ'(S#, P#, J#, JADDRESS, ZIPCODE, QREQUIRED, QOH)

A2.2.2 Second Normal Form

- A relation is in second normal form, if the relation is in first normal form, and each nonprime domain in the relation is fully functionally dependent upon every candidate key.

Transformation To Second Normal Form

- First normal form relation

  SP'(S#, P#, PNAME, PRICE, COLOUR, WEIGHT)

  - where

    P# → PNAME, PRICE, COLOUR, WEIGHT

    ⇣

- Second normal form relations

  SP"(S#, P#)

  P"(P#, PNAME, PRICE, COLOUR WEIGHT)

- First normal form relation

  SPJ'(S#, P#, J#, JADDRESS, ZIPCODE, QREQUIRED, QOH)

  - where

    S#, P#, J# → QREQUIRED, QOH

    J# → JADDRESS, ZIPCODE

    ⇣

- Second normal form relations

  SPJ"(S#, P#, J#, QREQUIRED, QOH)

  J"(J#, JADDRESS, ZIPCODE)

A2.2.3 Third Normal Form

A relation is in third normal form if the relation is in second normal form, and each nonprime domain in the relation is nontransitively dependent upon every candidate key.

It is assumed that all nonprime domains are totally independent of each other; i.e., no values of any nonprime domain are functionally depend upon the values of any other nonprime domain, and each nonprime domain can assume any value without respect to the value of any other nonprime domain in the same relation.

Transformation To Third Normal Form

Second normal form relation

J"(J#, JADDRESS, ZIPCODE)

- where

J# → JADDRESS

JADDRESS ↛ J#

JADDRESS → ZIPCODE

- thus

J# → ZIPCODE

Third normal form relations

J'''(J#, JADDRESS)

A'''(JADDRESS, ZIPCODE)

REFERENCES

1. Bell, D.E., Secure Computer Systems: A Refinement of the Mathematicl Model, ESD-TR-73-278, Volume III, The MITRE Corporation, Bedford, Massachusetts, April 1974.

2. Bell, D.E. and La Padula, L.J., Secure Computer System: Unified Exposition and Multics Interpretation, ESD-TR-75-306, The MITRE Corporation, Bedford, Massachusetts, July 1975.

3. Biba, K.J., Integrity Considerations for Secure Computer Systems, in preparation, The MITRE Corporation, Bedford, Massachusetts.

4. Codd, E.F., A Relational Model of Data for Large Shared Data Banks, Communications of the ACM, Volume 13, Number 6, June 1970.

5. Codd, E.F., Normalized Data Base Structure: A Brief Tutorial, IBM Research Report RJ 935, San Jose, California, November 1971.

6. Codd, E.F., Relational Completeness of Data Base Sublanguages, IBM Research Report RJ 987, San Jose, California, March 1972.

7. Codd, E.F., Further Normalization of the Data Base Relational Model, Data Base Systems, Courant Computer Science Symposium 6, Prentice-Hall, Englewood Cliffs, New Jersey, 1972, 33-64.

8. Codd, E.F., and Date, C.J., Interactive Support for Non-programmers: The Relational and Network Approaches, IBM Research Report RJ 1400, San Jose, California, June 1974.

9. Herstein, I.N., Topics in Algebra, Blaisdell Publishing Company, Toronto, 1965.

10. Hinke, T.H. and Schaefer, M., Secure Data Management System, Final Technical Report, RADC-TR-75-266, (Performing organization: System Development Corporation), Rome Air Development Centre, A.F.S.C., New York, November 1975.

11. Millen, J.K., Security Kernel Validation in Practice, Communications of the ACM, Volume 19, Number 5, May 1976, 244-245.

12. Rutherford, D.E., Introduction to Lattice Theory, Oliver and Boyd Limited, Edinburgh and London, 1965.

13. Tsichritzis, D.C., On Implementation of Relations, Technical Report CSRG-35, Computer Systems Research Group, University of Toronto, Toronto, Ontario, Canada, May 1974.