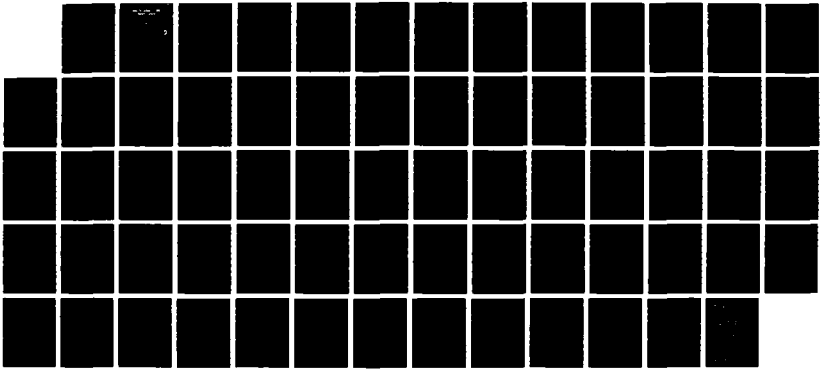
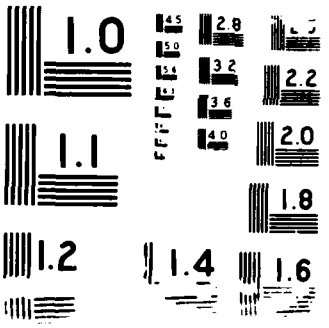
AD-A194 684 SECURE ACCESS CONTROL WITH HIGH ACCESS PRECISION(U) 1/1
NAVAL POSTGRADUATE SCHOOL MONTEREY CA G S HOPPENSTAND
MAR 88

UNCLASSIFIED F/G 12/8 NL




1.0
1.1
1.2
1.4
1.6
1.8
2.0
2.2
2.8
3.2
3.6
4.0
4.5
5.0

AD-A194 684

NAVAL POSTGRADUATE SCHOOL
Monterey, California

DTIC FILE COPY

DTIC
ELECTE
JUL 05 1988
D
E

THESIS

SECURE ACCESS CONTROL WITH HIGH ACCESS PRECISION

By

Gregory S. Hoppenstand

March 1988

Thesis Advisor: D.K. Hsiao

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) Code 52	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
SECURE ACCESS CONTROL WITH HIGH ACCESS PRECISION

12 PERSONAL AUTHOR(S)
Hoppenstand, Gregory S.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1988 March	67

16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Database security; Multilevel security; Computer security

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

When classified data of different classifications are stored in a database, it is necessary for a contemporary database system to pass through other classified data to find the properly classified data. Although the user of the system may only see data classified at the user's level, the database system itself has breached the security by bringing the other classified data into the main memory from secondary storage. Additionally, the system is not efficient as it could be because unnecessary material has been retrieved. This is a problem in access precision.

This thesis proposes a solution to the access precision and pass-through problems using a database counterpart to the mathematical concept of equivalence relations. Each record of the database contains at least one security attribute (e.g., classification) and the database is divided into compartments of records; Compartments are disjoint sets, where each compartment of records has the same aggregate of security attributes.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Prof. D.K. Hsiao	22b TELEPHONE (Include Area Code) (408) 646-2253 / 22c OFFICE SYMBOL Code 52Hq

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

GPO: Government Printing Office 1986—606-24.

#19 ABSTRACT (continued)

A suitable database model, the Attribute-Based Data Model, is selected, and an example of implementation is provided.

Accession For
NTIS GRA&I [X]
DTIC TAB [ ]
Unannounced [ ]
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

DTIC COPY INSPECTED 6

Approved for public release; distribution is unlimited.

SECURE ACCESS CONTROL
WITH
HIGH ACCESS PRECISION

by

Gregory S. Hoppenstand
Lieutenant, United States Navy
B.S., Purdue University, 1979

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

March 1988

Author: _____
Gregory S. Hoppenstand

Approved by: _____
Professor D.K. Hsiao, Thesis Advisor

_____
Associate Professor T. Wu, Second Reader

_____
Vincent Y. Lum, Chairman
Department of Computer Science

_____
James M. Fremgen,
Acting Dean of Information and Policy Sciences

ABSTRACT

When classified data of different classifications are stored in a database, it is necessary for a contemporary database system to *pass through* other classified data to find the properly classified data. Although the user of the system may only see data classified at the user's level, the database system itself has breached the security by bringing the other classified data into the main memory from secondary storage. Additionally, the system is not as efficient as it could be because unnecessary material has been retrieved. This is a problem in *access precision*.

This thesis proposes a solution to the *access precision* and *pass-through* problems using a database counterpart to the mathematical concept of equivalence relations. Each record of the database contains at least one security attribute (e.g., classification) and the database is divided into compartments of records; Compartments are disjoint sets, where each compartment of records has the same aggregate of security attributes.

A suitable database model, the *Attribute-Based Data Model*, is selected, and an example of implementation is provided.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. AN OVERVIEW - THE DEFINITIONS OF THE ISSUES

Two outstanding technical issues in controlling accesses to databases of a secure database system are articulated in this thesis and a solution is proposed Although these two issues are motivated and characterized separately, the solution to one of the two issues appears to also be the solution to the other.

The first issue is *access precision*. Access precision is defined as the ratio of the amount of accessed data that satisfies a user's query versus the amount of data that have been retrieved from secondary storage and placed in the main memory in response to that query.

Ideally, the amount of accessed data satisfying a query should be equal to the amount of data retrieved by the database system. In this ideal situation, the access precision of the database system is said to be *absolute*. Absolute precision in access control of a database system is indicative of great control and high performance, where every piece of data retrieved in response to the query is authorized for release. To implement access control, conventional secure database systems use either the view mechanism or the query modification, or a combination of the two. Unfortunately, neither the view mechanism nor query modification can achieve absolute precision, since they are both filtering mechanisms.

The second issue is *pass-through*. When classified data with different classifications are stored in a database, it is necessary for a contemporary database system to pass

through other classified data in order to find the properly classified data. Although the user of the classified data will receive from the system the properly classified data, the database system has breached the security by bringing the other classified data into the main memory from the secondary storage. Ideally, a database system should not breach the security by bringing out unauthorized (or otherwise unwanted) data from the secondary storage but, instead, bring in only authorized (i.e., properly classified) data into the main memory. Database systems that do not suffer from the pass-through problem can survive crashes of the system software and hardware without releasing unauthorized data to the user. It is clear also that systems with the pass-through problem do not operate with absolute access precision either, so efficiency also suffers.

B. THE METHODOLOGY USED

The proposed solution to the access precision and pass-through issues uses a database counterpart to the mathematical notion of *equivalence relations*. In mathematics, an equivalence relation of sample data effectively partitions the sample data into mutually exclusive subsets of data. In the database system environment, equivalence relations will be identified based on attribute values and attribute-value ranges so that the database can be partitioned into mutually exclusive *compartments*. These attributes are the *security attributes*.

These secured compartments are collections of records where each compartment of records has the same aggregate of security attributes. Further, no two compartments have a common record with the same aggregate of security attributes. By referring to security attributes in parsing the attributes of predicates of a user query, the database system can decide a priori the needed compartments of records which satisfy the query. In this way,

only records in the proper compartments are accessed. Records in other compartments will not be passed through. This mode of access to the compartmentalized database via security attributes eliminates the pass-through problem.

The concepts of equivalence relation and secured compartments may also be employed for the improvement of access precision. By defining an equivalence relation of non-security attributes in terms of their attribute values and attribute-value ranges, we can effectively partition the database into *clusters* of records such that records of a cluster are characterized by the same set of attribute values and attribute-value ranges. Further, no two clusters contain a common record with the same set of attribute values and attribute value ranges. The equivalence relation and cluster assignments are defined in the *meta data* of the database. The records belonging to these clusters constitute the *base data* of the database.

After review of current data models, D. K. Hsiao's Attribute-Based Data Model [1] (ABDM) was chosen due to its use of the meta-data organization which defines the clusters (equivalence classes). In the ABDM (see Chapter IV for details), a user's query is first processed against the meta data. Attribute values and attribute-value ranges of the meta-data which appear in the user's query are then identified. A Cartesian product of these values and ranges is performed next, resulting in one or more sets of attribute values and attribute-value ranges. These sets are compared with the Cartesian product of the members of the meta data. If a set is a subset of a member, then the cluster defined by the member must have the records satisfying the user's query. Consequently, accesses are made directly to those and only those clusters of the base data whose defining Cartesian-product members contain the sets of attribute values and attribute-value ranges

determined by the query. Accesses to these clusters of the dase data are direct with no passing through other clusters of the base data.

This thesis uses the ABDM to demonstrate how the notion of equivalence classes and security attributes can be combined to develop a database system that can achieve very high access precision and also eliminate the pass-through problem. For perhaps the first time, the incorporation of better access control to a database system does not incur overhead in terms of inprecision and also improves the access precision as well. In fact, the new security mechanism will facilitate high performance and better control objectives.

C. SUPPORTING WORK

Characterization and identification of the pass-through and access precision problems is due to D. K. Hsiao [2], who also outlined the *security atom* concept which is the fundamental basis of the security methodology outlined in this paper. Work by E. Wong and T. C. Chiang [3] also provided the basis for some of the work.

D. ACKNOWLEDGMENTS

The author expresses sincere thanks to Captain J. Leonard, USN, who provided critical support to this project while he served as the Assistant Commander, Telecommunications and ADP Systems for the Naval Security Group Command; and to Carolyn Deverin, Manager for External Education for the Department of Defense National Computer Security Center, for her skillful assistance throughout the project.

II. DATABASE SECURITY

Information is a precious commodity in the modern world. Many organizations, even though they might not realize its preciousness, have assigned values to their information. In essence, the data maintained is analogous to a currency, and the computer and storage devices form a bank.

The Department of Defense assigns classifications to all of its documents, and has defined the value of each type of data depending on its classification level. This value is used to estimate the damage to the United States if this material is compromised. For example, the compromise of one classification results in *serious* danger to the security of the country while the compromise of a higher classification can result in *grave* danger.

The objective of database security is to effectively protect information maintained in a database. This protection of information is against unauthorized disclosure, alteration, or destruction [4].

A. SECURITY THREATS

The possible security threats to a computer system are too numerous to list here but it appears worthwhile to outline some of the major threats [4]. These include physical threats which are relatively easy to protect against such as the theft of the storage medium, line tapping, and cross talk from a secure line to an unsecure line.

Other threats which are more difficult to protect against include undetected hardware failures; failure of systems software protection mechanisms; incorrect specification of

security policy by the security administrator; malicious and non-malicious disabling of security mechanisms by a systems programmer; and fraudulent identification by a user.

Even more insidious threats exist such as a *Trap Door*, which is defined by the DoD Trusted Computer System Evaluation Criteria [5] (the "Orange book") as "a hidden software or hardware mechanism that permits system protection mechanisms to be circumvented. It is activated in some non-apparent manner (e.g., a special "random" key sequence at a terminal)." Another threat, *Trojan Horse*, "is a computer program with an apparently or actually useful function that contains additional (hidden) functions that surreptitiously exploit the legitimate authorizations of the invoking process to the detriment of security. For example, making a 'blind copy' of a sensitive file for the creator of the Trojan Horse." [5]

1. Special Characteristics of Database Security

There are also many special threats to the security of a database, some of which are applicable because of the special nature of databases. Some of the work on the security of databases have grown out of the work on secure operating systems. The basis for many of the security models in operating systems is that an *access rule* specifies the types of access a *subject* can have for an *object*. In the context of a database, the subject is often the user and the object is the data which is being accessed. The special characteristics of a database which differentiate its security requirements from an operating system include [4]:

- There are more objects to be protected in a database.
- The lifetime during which data is used normally is longer in a database.
- Database security is concerned with differing levels of granularity, such as a file, record type, field type, and field occurrence.

- Operating systems are concerned with the protections of physical resources. In database systems the objects can be complex logical structures, a number of which can map to the same physical data objects.
- There are different security requirements for the different database system architectural levels: internal to the system, conceptual to the user, and external to the transaction.
- Database security is concerned with the semantics of data, not just its physical characteristics.

2. Special Threats to a Database System

Two major types of data are threatened in a database system: *meta data*, which is information about the database, and *base data*, which consists of the records of the database itself.

a. Direct Disclosure of Data

This includes disclosure by the database system itself or by the individual. This paper is concerned with inadvertant disclosure of data by the system, not the individual. Such faulty disclosures are often termed *spillage*.

b. Modification/destruction of Data

There are additional threats other than inadvertant disclosure. A malicious user can do severe damage to a database by modifying or destroying data. There have also been documented events where computer "viruses" entered major computer systems via modem and destroyed major files.

c. Inference

Inference in database systems is a very difficult problem to solve. In general, an inference attack occurs when a user infers unauthorized data from other knowledge he has obtained from the system legally or which he already holds. These include both semantic and statistical inferences. Much of the current work to solve

inference problems uses rule-based AI techniques [6].

d. Aggregation

Aggregation threats are deductions of sensitive information using the overall impression given by a data aggregate. It follows that aggregation threats are the opposite of inference threats.

e. Trojan Horse

In the case of database systems, a *Trojan Horse* can refer to a transaction which lies hidden within another transaction. When the later transaction is being executed, the hidden transaction then attacks the database and breaches its security.

B. COUNTERMEASURES

Countermeasures are methods of protecting the data in a database. The three major methods are: *access controls, inference controls,* and *backup-and-recovery* procedures [7]. The first two methods are active protection and the latter passive. This thesis is only concerned with the access control issue.

1. Access Controls

Access is defined in [5] as "a specific type of interaction between a subject and an object that results in the flow of information from one to the other." At this point more precise definitions of subject and object are required. By the same reference, *subject* is "an active entity, generally in the form of a person, process, or device that causes information to flow among objects or changes the system state." An *object* is "a passive entity that contains or receives information." An access to an object potentially implies an access to the information it contains. Examples of objects are: records, blocks, pages, segments, files, directories, and programs, as well as bits, bytes, words, fields, processors,

video displays, keyboards, clocks, etc.

Access controls involve in the assignment of users rights and privileges for the purpose of controlling access to the database.

2. View Mechanisms and Query Modifications

The *view mechanism* defines a view, or subset of attributes, on a relation (file) in a relational (non-relational) database system. The user is restricted from accessing those attribute values of the relations (files) whose attributes are not in the view. Views serve to filter out the sensitive data.

In the *query modification* mechanism, a user's query is modified according to the user's data access rights by appending a permit or deny clause as a conjunction to the original query. The combined query is then submitted to the query processor of the database system as the query of the user. In effect, the user is restricted to the logical intersection of the original query and the added restricting clause.

The effects of both methods are to control the data accessed. However, in the case of the view mechanism, access precision is very poor because extra data must be brought into the main memory for filtering, and query modification, although access precision is good, tends to be inefficient because the system must perform additional query processing, i.e., processing both the user query and security clause.

C. MULTILEVEL SECURITY

A multilevel secure system is defined by [5] as "a class of system containing information with different sensitivities that simultaneously permits access by users with different security clearances and needs-to-know, but prevents users from obtaining access to information for which they lack authorization."

The security methodology outlined herein is designed to be used in the design and implementation of a multilevel secure database. Such a database would allow users with different access rights to use the same database and ensure through security protection that data rights would be given only in accordance with the rights of the user.

Such a database would be characterized by efficiency in storage, since there would not have to be physical partitioning by classification (e.g., separate disk packs for each aggregate of security attributes), and efficiency in update, since multiple copies of data would not have to be kept.

III. EQUIVALENCE RELATIONS

The concept of an equivalence relation is intrinsic to the understanding of the security methodology proposed in this thesis. In effect, we will demonstrate a method using an equivalence relation to partition a set S of records comprising a database into disjoint, non-empty equivalence classes. Thus, equivalence classes as sets of records will enable the chosen data model to address each set directly without having to *pass through* records in other sets, i.e., other equivalence classes.

A method will be outlined to ensure each equivalence class will correspond to a different combination of security attributes. For example, a set of records classified as TOP SECRET would be in one equivalence class, and those which are only SECRET would be contained in another. Each equivalence relation R that partitions S into these equivalence classes for security purposes will be a security attribute (it will be shown in IV that every directory attribute in the chosen data model participated in an equivalence relation).

A. SOME SET THEORETICAL NOTIONS

We assume that the reader is familiar with the notion of a *set*, a collection of objects (*members* of the set) without repetition. Finite sets may be specified by listing their members between brackets. For example, {2,4} denotes a set consisting of the even numbers 2 and 4.

Sets may also be specified:

{ x | P(x) }

{ x in A | P(x) }

The first notation is read "the set of objects x such that P(x) is true," where P(x) is some property of or statement about objects x. The second reads: "the set of x in set A such that P(x) is true." [8]

B. RELATIONS AND EQUIVALENCE RELATIONS

A (binary) *relation* is a set of pairs. The first component of each pair is chosen from a set called the *domain*, and the second component of each pair is chosen from a (possibly different) set called the *range*. An alternate definition is that a binary relation, R, from set A to set B, is a subset of A x B (the *cartesian* product of sets A and B). If R is a relation, and (a,b) is a pair in R, then common notation is aRb.

1. Properties of Relations

We say a relation R on set S is:

a. reflexive if aRa for all a in S;

b. irreflexive if aRa is false for all a in S;

c. transitive if aRb and bRc imply aRc;

d. symmetric if aRb implies bRa;

e. asymmetric if aRb implies that bRa is false.

2. Equivalence Relations

A relation R that is reflexive, symmetric, and transitive is said to be an *equivalence relation*. An important property of an equivalence relation R on a set S is

that R partitions S into disjoint nonempty equivalence classes. That is

$$S = S_1 \cup S_2 \cup ...$$

where for each i and j, where i is not identical to j:

a. $S_i \cap S_j$ = null set;

b. for each a and b in S_i, aRb is true;

c. for each a in S_i and b in S_j, aRb is false.

The S_i's are called *equivalence classes*.

IV. THE ATTRIBUTE-BASED DATA MODEL

To characterize my access control method, a data model is proposed. This model, known as the *attribute-based data model,* developed by D. K. Hsiao [1] and extended by E. Wong and T. C. Chiang [3], characterizes two kinds of data: base data and meta data. The base data consists of records and the meta data consists of information about the base data, i.e., about the records.

A. THE BASE DATA

Logically, a database consists of different files of records. Each file contains a collection of records which are characterized by a unique set of directory keywords. Thus, a record is composed of two parts. The first part is a series of attribute-value pairs or keywords. An attribute-value pair, i.e., *keyword,* is a member of the Cartesian product of the attribute set and the value domain of the attribute. As an example, < POPULATION, 25000 > is an attribute-value pair having 25000 as the value for the population attribute. A record contains at most one attribute-value pair for each attribute defined in the database.

Certain attribute-value pairs of a record are called *directory keywords* of the record, because either the attribute-value pairs or their *attribute-value ranges* are kept in a *directory* for identifying the records. Those attribute-value pairs which are not kept in the directory are called *non-directory keywords*. The rest of the record is the textual data, which is referred to as the *record body*.

An example of a record with three keywords and a record body is shown below.

(< Category, US City Description >, < City, Monterey >,
< Population, 25000 >, {The city of Monterey was
founded in ... })

The angle brackets, <,>, enclose an attribute-value pair, i.e., keyword. The curly brackets, {,}, include the record body. The first attribute-value pair of all records of a file, by convention, is the same. In particular, the attribute is CATEGORY and the value is the category name (CATEGORY is equivalent in concept to a logical file). A record is enclosed in the parenthesis. For example, the above sample record is from the USCityDescription category. The use of brackets and parentheses is for presenting a record in a linear form without due consideration of its physical format. With a linear form, we can refer to individual records directly and easily.

It is also important to note that all of the above constructs are well defined in terms of other constructs. The only two undefined constructs are the *attribute set* and the *value domain*. Intuitively, we know that an attribute is the characteristics of the value. For the base data, the only rule placed on a record is that no two attributes are identical. This rule ensures that in a record all of the attribute-value pairs are single-valued and that there are no redundant (identical) attribute-value pairs. For example, the following records are not allowed:

(<CATEGORY, USCity Description>, <City, Monterey>,
<Population, 1000000>, <Population, 25000>,
{The city of Monterey was founded ...})

(<CATEGORY, USCityDescription>, <City, Monterey>,
<City, Monterey>, <Population, 25000>,
{The city of Monterey was founded ...})

The first has two different population figures. Therefore, the population attribute is not single-valued. The second repeats the city name which results in a redundancy.

The records of the files of a database constitute the *base data* of the database. In general, given a record of attribute-value pairs, one cannot distinguish the directory keywords from the non-directory keywords unless one refers to the meta data which is discussed in Section C.

B. IDENTIFYING RECORDS

For the user, the records of the database may be identified by utilizing keyword predicates. A *keyword predicate* (for brief, *predicate*) is a 3-tuple consisting of an attribute, a relational operator (=, >, <, ≥, ≤), and an attribute value, e.g., POPULATION > 20000 is a keyword predicate. More specifically, it is a greater-than predicate. Combining keyword predicates in *disjunctive normal form* characterizes a *query* of the database. The following query

(CATEGORY= USCityDescription and CITY = Monterey) or
(CATEGORY= USCityDescription and CITY = San Jose)

will be satisfied by all of the records of the USCityDescription category with the CITY of either Monterey or San Jose. For clarity, parentheses are used for bracketing conjunctions in a query. Thus, we have the following general format of a query:

$$(P_{11} \text{ and } P_{12} \text{ and } \cdots \text{ and } P_{1n_1}) \textit{ or } (P_{21} \text{ and } P_{22} \text{ and } \cdots \text{ and } P_{2n_2}) \textit{ or } \cdots \textit{ or } (P_{m1} \text{ and } P_{m2} \text{ and } \cdots \text{ and } P_{mn_m})$$

In disjunctive normal form, the predicates of a conjunction must not be contradictory.

For instance, we cannot have (Population > 10000 and Population < 10000) as a conjunction, since the two predicates contradict each other, resulting in no records whose population features can be both greater and less than 10000.

For the database system, the records of the database are identified by unique *record ids*. These ids are assigned to the records at the time that the records are being entered into the database. They are unique, since no two records have the same ids. Id assignment for records is a function of the database system. For purposes of explanation, we simply use arbitrary letters and numerals such as R1, R2, ... , R100, ... to designate the record ids.

C. THE META DATA

The *meta data* is stored information about the base data. Collectively, the various meta-data constructs form the *directory* of the database. The directory has the following constructs: attributes, descriptors, and clusters. An *attribute* is used to represent a certain common property of the base data as defined before, e. g., POPULATION is an attribute that corresponds to actual population features in the database. A *descriptor* is used to describe a range of values that an attribute can have. For example (10001 ≤ POPULATION ≤ 15000) is a possible descriptor for the attribute POPULATION. The descriptors that are defined for an attribute, e.g., population ranges, are mutually exclusive in terms of their values. Now the notion of a cluster can be defined. A *cluster* is a collection of records such that every record in the cluster satisfies the same set of descriptors. For example, all records with POPULATION between 10001 and 15000 may form one cluster whose descriptor is the one given above. In this case, the cluster satisfies the set of a single descriptor. In reality, a cluster tends to satisfy a set of multiple

descriptors. For the meta data, the only rule we place on a record is that a record must have at least one directory attribute. The attributes of the records that appear in the directory are *directory* attributes.

The condition that the descriptors defined for a given attribute have mutually exclusive attribute values is a necessary and important one. Mathematically, the descriptors of the attribute serve to derive equivalence classes which effectively partition the database into mutually exclusive sets of records. These record sets are, of course, termed clusters here. Thus, database clusters, equivalence classes and database partitions in this context are synonymous.

The sizes of the clusters are not uniform. Some clusters may have several records; other clusters may have only a few; while still other clusters may not have any. To keep track of the directory attributes for which descriptors are formed, we utilize an attribute table. For all of the descriptor sets that effectively partition the database into clusters, we maintain a descriptor-set table. Finally, to know which records belong to which clusters, we use a cluster table. More specifically, the directory is organized in three tables: *the attribute table (AT), the descriptor-to-descriptor-id table (DDIT) and the cluster-definition table (CDT),* examples of which are given in Figure 4.1. AT maps directory attributes to the descriptors defined on them. A sample AT is depicted in Figure 4.1 (a). DDIT maps each descriptor to a unique descriptor id. A sample DDIT is given in Figure 4.1 (b). CDT maps descriptor-id sets to cluster ids. Each entry consists of a unique cluster id, a set of descriptor ids whose descriptors define the cluster, and ids of the records that are in the cluster. A sample CDT is shown in Figure 4.1 (c).

We have a number of observations in reviewing the sample tables. Our first observation is that AT is too small. In reality, any sizable database may have hundreds, if not thousands, of attributes in the database. For our illustration, we simply use three.

Attribute	Attribute Type	DDIT Entry
POPULATION	A	D11
CITY	C	D21
CATEGORY	B	D31

(a) An Attribute Table (AT).

Id	Descriptor
D11	0 ≤ POPULATION ≤ 50000
D12	50001 ≤ POPULATION ≤ 100000
D13	100001 ≤ POPULATION ≤ 250000
D14	250001 ≤ POPULATION ≤ 1000000
D21	CITY = Cumberland
D22	CITY = Columbus
D23	CITY = Monterey
D24	CITY = Toronto
D31	CATEGORY = CanadaCityDescription
D32	CATEGORY = USCityDescription

Dij: Descriptor j for attribute i.

(b) A Descriptor-to-Descriptor-Id Table (DDIT).

Id	Desc-Id Set	Text-Id.
C1	{D11,D21,D32}	R1,R2
C2	{D14,D22,D32}	R3
C3	{D12,D23,D32}	R4
C4	{D14,D24,D31}	R5

(c) A Cluster-Definition Table (CDT).

Figure 4.1 The Directory Tables

Our next observation is that there are different kinds of descriptors in DDIT, descriptors that deal with attribute-value ranges and descriptors that deal with distinct attribute values. More specifically, there are three classifications of descriptors. A *type-A* descriptor is a conjunction of less-than-or-equal-to predicate and a greater-than-or-equal-to predicate, such that the same attribute appears in both predicates. For example, ((POPULATION ≥ 50001) and (POPULATION ≤ 100000)) is a type-A descriptor. A *type-B* descriptor consists of only an equality predicate. (CATEGORY = USCityDescription) is an example of type-B descriptor. Finally, a *type-C* descriptor consists of the name of an attribute only. The type-C attribute defines a set of type-C sub-descriptors. *Type-C sub-descriptors* are equality predicates defined over all unique attribute values which exist in the database. For this reason, the type-C sub-descriptor are also of type-B. For example, the type-C attribute CITY forms the type-C sub-descriptors (CITY=Cumberland), (CITY=Columbus), (CITY=Monterey) and (CITY=Toronto), where "Cumberland", "Columbus", "Monterey", and "Toronto" are the only unique database values for the CITY in the base data. Unlike type-B descriptors, type-C descriptors are *on-demand* or *dynamic* descriptors. If for a type-C attribute, a type-C subdescriptor for a particular attribute value is not available in DDIT, a new type-C subdescriptor is automatically generated and inserted into DDIT. Hence, we have the notion of on-demand or dynamically available type-C subdescriptors. We note that we include the descriptor type in AT, as seen by the column titled Attribute Type in Figure 4.1 (a).

In reality, DDIT tends to be large, in fact, much larger than AT. It constitutes the bulk of the directory. The use of type-C descriptors tends to increase the size of DDIT

(and, therefore, the directory) enormously. On the other hand, type-A descriptors tend to conserve the directory space, since they do not use distinct values (such as social security numbers). Instead, they use value ranges. As long as there are reasonably small number of ranges, the number of type-A descriptors for a given attribute can be small.

We also observe that each descriptor, no matter what type, can partition the database into two parts - with a record of the database belonging in one partition but not in the other partition. As we introduce an additional descriptor for the same attribute, one of the partitions is further divided into two. Thus, for a given attribute the number of resulting partitions is *linearly* proportional to the number of descriptors introduced. For example, in referring to Figure 4.1 (b), if there was only one descriptor ($0 \leq$ POPULATIONS $\leq$ 50000) in DDIT, then the database records would be partioned into two sets of records whose population is in the range and whose population is not in the range. Should we introduce an additional descriptor ($50001 \leq$ POPULATION ≤ 100000), then the records would be partitioned into three sets of records - those whose population is under 50001, those whose population is in the range of 50001 and 100000, and those whose population is beyond 100000. Clearly, the number of record sets (partitions) is one greater than the the descriptors on POPULATION used. This is linear in proportion.

When two or more attributes are utilized in the descriptors, the number of record sets (partitions) is proportional to the *product* of the total number of descriptors for each of the different directory attributes. For example, in referring to Figure 4.1 (b) again, we have the following statistics:

The number of descriptors of the attribute POPULATION is 4;

The number of descriptors of the attribute CITY is 4;

The number of descriptors of the attribute CATEGORY is 2.

Thus, the maximal number of partitions is a product of 4, 4 and 2, i.e., 32, since 32=4×4×2. In other words, there could be 32 mutually exclusive sets of records in the database. In fact, we can characterize these record sets using the descriptors that have defined them. Mathematically, we are forming the Cartesian product of Di where Di is the set of descriptors for attribute i. Let

$$Di = \{Di1, Di2, ..., Di_{j_i}\}$$

where there are j_i descriptors of attribute i. The Cartesian product D1×D2× ... ×Di has the following members:

$$\{\{D11, D21, ..., Di1\}, \{D12, D22, ..., Di2\}, ..., \{D1j_1, D2j_2, ..., Dij_i\}\}.$$

We require that the descriptors used in DDIT have the *covering property*. The covering property states that (1) for the type-A descriptors of an attribute, all of the possible attribute values in the database for the attribute are covered by the attribute-value ranges of the descriptors and (2) for the type-B (or type-C) descriptors of an attribute, all of the possible attribute values in the database for the attribute appear in the type-B (or type-C) descriptors. Thus, each descriptor set defines a cluster of records. Specifically, there are potentially J clusters, where

$$J = j_1 \times j_2 \times ... \times j_i$$

It is important to note that although the potential number of clusters is 32 in this example, the CDT keeps track of only four of the 32 clusters. Thus, the CDT only keeps track of those clusters that contain records. As new records are being entered into the

database, existing clusters may contain more records and new clusters may be formed. The cluster size is reflected by the number of record ids kept in the CDT. The new clusters are reflected by the new entries in the CDT.

Although individual clusters may have different sizes, the average size of the cluster tends to become smaller, i.e., fewer records, as the number of descriptors which characterize the clusters increase. There are two factors in the increase of the number of descriptors. One factor is that there are more descriptors for a given attribute, i.e., there are more Dij for a given i. The other factor is that there are more attributes as well, i.e., i is a large number.

D. THE RETRIEVE OPERATION

To search for records in the database, the user specifies a search request. The request has the following form:

RETRIEVE (query) [target-list] [by-clause]

where the literal indicates the name of the operation, parentheses enclose the mandatory qualification and brackets enclose the optional qualifications. A sample search request is given in Section D.2.

1. The Six Execution Steps

Step 1: *Processing the Query against the Attribute Table.*

The query in the search request is in a disjunctive normal form:

$$(P_1 \hat{} P_2 \hat{} \ldots \hat{} Pr_1)_1 \vee (\ )_2 \vee \cdots \vee (\ )_s.$$

We note that each P_i is a predicate. The predicate may either be an equality predicate with a unique attribute value, e.g., (City = Monterey); or a predicate with attribute-value ranges, i.e., (5001 ≤ Population < 10000). In particular, there are r_1 predicates in the first conjunction and s conjunctions in the query.

Let r_i be the number of predicates in the i-th conjunction, then there are r number of predicates in the query where

$$r = r_1 + r_2 + r_3 + \ldots + r_s.$$

It follows that there are at most r different attributes, since each predicate contains an attribute.

The processing of the query against the attribute table (AT) is to determine which of the r attributes in the query are in the AT. Those that do not appear in AT are non-directory attributes. Only directory attributes are utilized in this step. Non-directory attributes and their predicates are utilized in a later step.

Assuming that t attributes are found in AT, we then have s conjunctions in the query, t directory attributes, and (r-t) non-directory attributes in the query. These t directory attributes found in AT play an important role in the next step of meta data

processing. They form the *directory-attribute list* of the query. This first step is termed the *attribute-determination* step.

Step 2: *Processing the Directory Attributes against the Descriptor-to-Descriptor-Id Table.*

Once the directory attributes for the query are found in AT, we can then use the directory-attribute list of the query to locate the descriptors in DDIT whose attributes are identical to the attributes on the list. Locating descriptors in DDIT is a rather simple task. For a given attribute, Ai, a descriptor j for the attribute has the id Dij. As an id number, Dij is either identical to the number Di1 or follow the number Di1. We note that Di1 is placed in AT along with Ai. Thus, the search involves the following algorithm, implemented for each conjunction in the query:

(1) Pick an attribute Ai from the directory-attribute list appearing in the conjunction.

(2) Use Ai's id number, e.g., Di1 in AT, to access the DDIT entry.

(3) Does value in DDIT meet requirements of predicate?

(4) If yes, then we have found a descriptor. We then record its corresponding descriptor id, Dij.

Repeat (3) and (4) for the next DDIT entry. Since there are only a finite number of entries for the descriptors of the same attribute, the repetition will eventually be terminated. These steps are then repeated until each attribute appearing in the conjunction has been defined.

At this point, for the attribute Ai, there are k_i descriptors and their id's determined, i.e., Di1, Di2, . . . , Dik_i. The set of descriptors identified by Di1, Di2, . . .

Dik_i is called the *Ai-descriptor group*. Their ids form the *Ai-descriptor-id group*. There is an Ai-descriptor(-id) group for each attribute Ai in the attribute list. If there has been a directory-attribute list of t attributes from Step 1, there are now t Ai-descriptor groups one for each i. In addition, there are t Ai-descriptor-id groups. Thus, for each Ai, there are two sets: the Ai-descriptor group and the Ai-descriptor-id group. For example, if A_1 is the attribute POPULATION and the descriptor is (50001 < POPULATION < 100000), then the A_1-descriptor group is {(50001 < POPULATION < 100000)} and the A1-descriptor-id group is {D12}. This example is based on the meta data of AT and DDIT in Figure 4.1. We note these groups are singletons, i.e., sets of only single elements. In practice, they are not restricted to singletons. However, the groups are by nature small, containing only a few elements. We call these Ai-descriptor-id groups D_i.

The rest of this step is to form the Cartesian product of the Ai-descriptor-id groups for i = 1, . . ., t,

$$D_1 \times D_2 \times \ldots \times D_t.$$

Step 3: *Determining the Cluster Ids.*

With the given descriptor-id groups from step 2, it is an easy task to determine the corresponding cluster ids by searching CDT. The search is a simple comparison of each given descriptor-id group with the descriptor-id sets in CDT. If there is a match, then a cluster id is found in CDT for the descriptor-id group. A match means that either the group is identical to the set or the group is a subset of the set. The time complexity for this comparison is at worst O(mn) where m is the number of given descriptor-id groups from step 2 and n is the number of descriptor-id sets in CDT. In practice, m is

usually a smaller number, even though n may be large. Further, the table search may be replaced by a B-tree search, since CDT is well suited for the B-tree organization. The time complexity for the search can be reduced to O(m log n).

Once all of the cluster ids are found, their clusters are determined. The number and identification of records that are in the clusters are also determined. This is all done with CDT. We term this step the *cluster determination* step.

Step 4: *Determining the Record Ids and Addresses*

For each cluster determined in step 3, there is a corresponding list of one or more record ids. Record ids uniquely identify the records. In this step, the unique identifiers are converted into disk addresses for the records. Once converted, disk accesses commence. Therefore, this step is termed the *address-generation-and-record-access* step.

Step 5: *Selecting the Records.*

After a record is brought into the main memory from the disk storage, the record is checked against the query. We observe that the previous four steps applied to the query of the operation have resulted in having these records brought into the main memory. We might conclude that these records by definition satisfy the query and that there is no need for further checking of the records against the query. What we have to be aware of is that a query is of predicates of keywords. If none of the keywords is a non-directory keywords, then all the keywords in the query are directory keywords. They are, of course, all used in the previous steps to determine the records. Consequently, we might conclude that in the case of the all-directory-keyword query, this

step is superfluous. However, this is not the case due to use of query with attribute-value ranges. On the other hand, when the user issues a query, the user does not necessarily know the difference between directory and non-directory keywords. The user-issued query is likely to consist of non-directory keywords. Non-directory keywords are not a part of the meta data of the database. They do not appear as attributes, descriptors or descriptor ids in AT, DDIT and CDT. Consequently, the non-directory keywords have never assisted the system software in narrowing the search space for records on the disks. However, non-directory keywords and predicates serve as additional qualifications of the records. Unless a newly accessed record qualifies the non-directory predicates, the record is not qualified for the entire query. Records that qualified for both directory and non-directory predicates are selected for either outputting or optional processing. This step is therefore called the *record selection* step.

Step 6: *Processing the Records.*

The final step, i.e., the sixth step, is the *record processing* step. Record processing involves two distinct functions: the value extraction function and the aggregation function. In the value extraction function, not all of the attribute values of a record are needed for an operation. The attribute values needed are usually indicated by the user via the target-list attributes. Thus, with the given *target list*, the system software extracts the attribute values for outputting. The aggregation function performs an aggregate operation such as maximum, minimum, or mean, as indicated in the operation. When indicated, an aggregate operation is always carried out after the value extraction operation. In summary, the record processing step consists of two optional phases: value extraction and aggregate operations.

2. A Sample Search

We now review the step-by-step processing of a sample search operation. Consider the example where the search operation is as follows:

RETRIEVE ((CATEGORY = USCityDescription) and (20000 < POPULATION < 90000) and (REGION = West Coast)) (CITY).

This search request will receive all of the names of the U.S. cities that are in the west coast region and have a population between 20,000 and 90,000 per city. We note that the query of the operation consists of a conjunction of three predicates - two equality predicates and one combined less-than-and-greater-than predicate. The target list has the attribute CITY. In referring to our often used AT in Figure 4.1, we discover in Step 1, CATEGORY and POPULATION are directory attributes because they have appeared in AT. REGION is not in AT. It is therefore a non-directory attribute. The implication is that the non-directory attributes are a part of the records. They may be used in the form of predicates of the query for selecting those records whose non-directory keywords qualify the predicates.

Our other observation is that in referring to DDIT in Figure 4.1 the two directory attributes will result in three descriptors (namely, CATEGORY=USCityDescription, 0<POPULATION<50000, 50001<POPULATION<100000) and three descriptor ids (i.e., D11,D12 and D32). This is the result of step 2. Furthermore, in step 2 we form the Cartesian product of two sets--the CATEGORY-descriptor-id group {D32} and POPULATION-descriptor-id group {D11,D12}. The product {D11,D12} × {D32} will result in two descriptor groups that

cover the query: {D11,D32} and {D12,D32}. These are the final products of step 2.

Once the descriptor-id groups are obtained, they are used to match the descriptor-id sets of CDT in Figure 4.1. We note that there are two such matches: {D11,D21,D32} and {D12,D23,D32}. Corresponding to these two descriptor-id sets are their cluster ids, C1 and C3, and their record ids, R1, R2 and R4. In other words, there are three records from two clusters that may qualify for the query. This is the result of step 3.

In step 4, the record ids (R1, R2 and R4) are transformed into the disk addresses of the records. This transformation is done by a built-in algorithm. Three records are retrieved from the secondary storage and placed in the buffers of the main memory. Let us look at them. They are depicted as follows:

(<CATEGORY, USCityDescription>, <CITY, Cumberland>, <POPULATION, 15000>, <REGION, West Coast>, {The City of Cumberland was founded in . . .})

(<CATEGORY, USCityDescription>, <CITY, Cumberland>, <POPULATION, 25000>, <REGION, East Coast>, {The City of Cumberland was founded in . . . })

(<CATEGORY, USCityDescription>, <CITY, Monterey>, <POPULATION, 65000>, <REGION, West Coast>, {The City of Monterey was founded in . . . })

Of course, the first two satisfy the descriptor set of {(CATEGORY, USCityDescription), (CITY =Cumberland), (0≤POPULATION<50000)}. The third satisfies the descriptor set of {(CATEGORY, USCityDescription), (CITY = Monterey), (50001 ≤ POPULATION < 100000)}. However, all three must be checked against the query. The first does not satisfy the query, since the population figure in the record does not fall into the population range of the query. Consequently, this record is not selected for processing. In the second, the three directory keywords all meet the qualification of the query.

Unfortunately, the non-directory keyword does not. The query requires the west coast region whereas the record is of the east coast region. Thus, the second record is disqualified. It is not selected for further record processing. Finally, the third is checked against the query. Both the directory and non-directory keywords qualify for the query. The record therefore satisfies the query. Consequently, this is the only record to be produced at the end of step 5.

In step 6, we process the record for value extractions and aggregation operations. Since no aggregation operations are specified for this operation, the only processing required is the target-list handling (i.e., the value extraction). CITY has been specified as the only attribute on the target list. The attribute value of the record for the attribute CITY is therefore extracted from the record and outputted. On the basis of the sample operation and sample records, the net output of this entire exercise of six steps is the simple string value, Monterey.

V. THE METHODOLOGY

As discussed in Chapter IV, one of the requirements of ABDM is that the descriptors defined for a given attribute have the covering property and that they have mutually exclusive attribute values. Further, we established that the CDT is a table of equivalence classes, each of which is disjoint and the union of which (all the records in the CDT) forms the complete database. The relation R defined over all clusters in the database is: *each record in a cluster has the same set of directory attribute descriptors.*

From this, it is clear a database can be partitioned into mutually exclusive sets based upon security classification if one or more security attributes are inserted into each record of the database and treated as directory attributes by ABDM.

Id	Desc-Id Set	Rec-Id.
C1	{D11,D21,D31,TS}	R1,R2

Id	Desc-Id Set	Rec-Id.
C2	{D11,D21,D31,S}	R3
C3	{D12,D22,D32,S}	R5

Id	Desc-Id Set	Red-Id.
C4	{D12,D22,D32,C}	R4

Figure 5.1 A Set of Cluster-Definition Tables (CDT)

A. DISCUSSION AND A SIMPLE EXAMPLE

In Chapter IV, only one CDT was used for the database. For the security methodology, we will have a separate CDT for each different combination of security attributes. There will be additional discussion of this later in the chapter. Assume Figure 5.1 is the set of CDTs resulting from the creation of a database using the AT and DDIT tables provided in Figure 4.1. *TS* stands for TOP SECRET; *S* stands for SECRET; and *C* for CONFIDENTIAL *(U* would stand for UNCLASSIFIED).

The letters stand for security attributes but these are not listed in the AT and DDIT. The reason for this will also be explained later.

Looking at the CDT, cluster C1 has population less-than-or-equal-to 50,000, Cumberland is the city, it is a CanadaCityDescription category (file), and the classification is TOP SECRET. C2 has all the same directory keywords except for the classification attribute. C2 is a cluster of SECRET records.

More specifically, the highest classification of data in the cluster C1 (containing records R1 and R2) is TOP SECRET and the highest classification of data in record R3 is SECRET.

> RULE 1: Every record in the database must contain all the security attributes used by the system and a value for each attribute. In many instances, this value will indicate the presence or absence of a special restriction (e.g., codeword).

For example:

> (< Category, US City Description >, < City, Monterey >,
> < Population, 25000 >, <Classification, SECRET >)

The above would be a valid record in the database. If the database contained additional security attributes these would also have to be included (examples will be

given in Chapter VI). Rule 1 ensures the database is effectively partioned into disjoint clusters of records with the same set of security attributes. If one or more security attributes were absent from a record, that record could be assigned to more than one cluster.

For example, if C2 did not have a security attribute for classification, R3 could be in C1 and C2 concurrently and the sets defined by the clusters are no longer disjoint. In Chapter VI an example will be provided using more than one security attribute: classification and codeword. Again, if a record only had a value for the classification and not codeword it could be assigned to two clusters. If the classification were SECRET, the record could be assigned to the SECRET with-no-codeword cluster and the SECRET with-codeword cluster at the same time.

> RULE 2: Every security attribute in the database must have the covering property.

The covering property for security attributes states that all possible security attribute values are known by the system and describe the value of that attribute for any record in the database. For example, the security attribute *classification* has only four values (as defined for our database): *TOP SECRET, SECRET, CONFIDENTIAL* and *UNCLASSIFIED*. Each record in the base-data must be able to be described by one of those classifications

Again, this covering property is required for all directory keywords in ABDM to ensure the database is partitioned into disjoint sets, i.e., to maintain the characteristics of equivalence classes.

> RULE 3: The descriptors for a given security attribute must have mutually exclusive attribute values.

This is again the same rule that applies to ABDM, and is necessary to support partitioning. The classification value *TOP SECRET* can not overlap in any way with the value *SECRET*. A record is either one or the other (here of course we are describing the highest classification and not access rights).

Given that we are satisfied the database is divided into disjoint compartments, the remaining issues concern controlling access to the CDT and in effecting the original partioning.

This follows since we know that, e.g., C2 (record 3) does not contain data at a higher classification than SECRET. If we can ensure that only persons with a minimum of a SECRET clearance accesses C2, then we have successfully controlled the access.

Note that at this point I am only discussing the issue of read access.

Some assumptions:

- A user with a TOP SECRET clearance should be able to read all records that have classification attribute values of: UNCLASSIFIED, CONFIDENTIAL, SECRET, and TOP SECRET. However, a user with a SECRET clearance should only be able to read at SECRET and below. More specifically, a user should be able to read all records for which his access is equivalent to or dominates.
- In order to prevent the pass-through and access precision problems with the meta-data, the clusters in a CDT referring to records the user can not access should not be brought into main memory.
- Our model assumes we have a trusted operating system acting as the guard for entry into the computer system and for restricting secondary storage retrieval requests

The procedure is as follows: When the user logs in, the system checks the password. We will assume the password is associated with the highest classification that user is allowed to access. At that time, the system will bring in from secondary storage the corresponding CDT(s).

Referring to Figure 5.1, if a user with a SECRET clearance logged onto the system, two CDT tables would be brought into main memory. The SECRET CDT table would contain entries for C2 and C3, and the CONFIDENTIAL CDT table would contain C4 (an UNCLASSIFIED CDT table would be brought in for all users). The maintenance of these tables is done by the database system and security administrator.

There will potentially be a separate CDT table for every possible combination of security attributes. For example, if in fact we did have a codeword attribute, the SECRET CDT would be divided into two separate CDTs: one with the codeword and one without (if each had records). Therefore, when the user logged on to the system and did not have a codeword clearance the system would only access the CDT(s) without the codeword. This of course implies that the "guard" to our system recognizes this difference but this is assumed.

> RULE 4: A separate CDT is maintained for each different set of security attribute descriptors.

After this is done, each query from the user can be processed exactly as ABDM does normally. If the query matches a descriptor id set that the user is not cleared for it will not be accessed because the set does not appear in his CDT and only the CDTs the user is cleared for will be searched. Chapter VI provides a detailed example of this access control method.

This methodology supports two additional, optional safeguard checks to ensure the system has formed the CDT correctly. At this stage before the record retrieval there can be another verification that the clearance level of the user is greater-than or equal-to the classification attribute contained in the descriptor id set. The second check can be

performed after the record is brought into main memory before release to the user. The clearance level can again be compared to classification attribute in much the same way that a non-directory keyword is checked by ABDM.

The format of the RETRIEVE command is not changed from the command in the original ABDM as described in Section IV. The system is tasked with bringing in all the CDT tables which contain clusters of records that the user is cleared to see. Therefore, security attribute does not need to be specified since all records which satisfy the directory attributes in the query will be provided to the user. These records may in fact span different classifications.

B. SYNOPSIS OF THE FUNDAMENTAL METHODOLOGY

To recap the methodology to this point:

- AT and DDIT remain the same for all users. Security attributes do not appear in the AT or DDIT. There is no need to do otherwise and both security and efficiency is supported (the user should not have access to tables that provide information on classifications they are restricted from).
- Each security attribute combination has a corresponding separate CDT listing all the clusters corresponding to that partition or, if there is no such CDT, this means there are no corresponding records in existence.
- The execution of the RETRIEVE operation would work now as the ABDM does, with the CDT chosen by the system based upon the clearance level of the user.
- Due to record clustering, only records the user is cleared to see will be brought into main memory.

C. OTHER ISSUES

Up to this point, we have outlined the fundamental control over database retrieval requests. ABDM, like other current database management systems, provides capabilities for records insertion, deletion, modification, and a logical join (RETRIEVE-COMMON in ABDM). These issues can be resolved through security policy and system design work

using the fundamental methodology already presented, and these implementation issues are not the primary thrust of this work. However, it does appear worthwhile to discuss at least record insertion and modification and outline how they might be handled.

1. Record Insertion

The basic *INSERT* request is, from the user's standpoint, organized in the exact manner of a RETRIEVE as described in Chapter IV with the difference being that the attribute values provided are used to create a new record.

INSERT(<CATEGORY,USCityDescription>, <CITY,Cumberland>,
<POPULATION,40000>)

The above will insert a record into the USCityDescription file for the city Cumberland with a population of 40000 (no record body is used in this example).

In order to satisfy the requirements for the security methodology, each record inserted will also require delineation of every security attribute used in the system. For example, if a basic classification scheme with codeword was used, the system would ensure the record was of the following format:

INSERT(<FILE,USCensus>, <CITY,Cumberland>,<POPULATION,40000>,
<CLASSIFICATION,S>, <CODEWORD1,Y>)

This example assumes there are only two security attributes in the system. If there were more these would also have to be entered to ensure we have the covering property.

In this example, the classification is SECRET and the material is codeword. In this case the codeword value is hidden. The association of *CODEWORD1* with the actual codeword used would be given to authorized users.

INSERT can only enter one record at a time into the database. The operation can result in significant work for the system, particularly if a *type C* attribute is used. However, this issue is one of implementation.

Under the security methodology, the basic system requirements of the INSERT will remain the same.

a. Step 1

For the example, the system would check to ensure that the user has the access/clearance necessary to enter the record of the security attributes given. A user with a SECRET clearance, for example, should not be able to enter TOP SECRET data.

As in the normal system operation, the non-security attribute-value pairs (keywords) of the record are identified. For each attribute-value pair, the attribute is used for matching the attributes in the AT. Locks are placed on the directory attributes found in the AT.

b. Step 2

With the attribute id's obtained from Step 1, the descriptors in the DDIT are searched to determine whether a descriptor *covers* the keyword of the record having the same attribute and the descriptor-id is used to form the descriptor-id group. The system must automatically add the security attributes to this descriptor-id group.

c. Step 3

Now a search is performed for the descriptor-id sets in the CDT(s) provided to the user based upon the user's access permission. If a descriptor-id set is found to be identical to the descriptor-id group, then the set is locked with a read/write-deny lock.

If there is no match, then a new entry is created in the appropriate CDT assuming that the descriptor-id group has enough descriptor ids to match an existing set or to create a new set. If the set does not have enough descriptor ids, this indicates the record lacks one or more directory keywords.

This step reveals more design issues for the security methodology. If the user is cleared to access SECRET and below codeword material, there will be separate CDTs for SECRET non-codeword and SECRET codeword clusters. The system could search through all CDTs looking for a match of the descriptor-id set, which includes the security attributes, and the access control would still be ensured. However, such global search is unnecessary since indexing could be used based upon the security attributes since the security attributes generate different CDTs.

For example, the system could automatically search the CDT containing SECRET codeword material by using a table lookup method and avoid searching the SECRET non-codeword CDT.

d. Step 4

The descriptor-id set of CDT which is matched or created is locked. The record id for the record is created and then placed in the locked entry for the cluster id in the CDT. The record id is transformed into a disk address and the record is

placed at the secondary storage at that address. The CDT is modified accordingly.

2. Record Update

The *UPDATE* operation in ABDM is very similar to the RETRIEVE operation: all six execution steps of the retrieval operation can be applied to update.

UPDATE((CATEGORY = USCityDescription) and (CITY = Monterey))
(POPULATION = POPULATION + 5000)

The above will modify all records in the file USCityDescription with the city of Monterey by increasing the population by 5000.

The primary complications concern security policy and *record migration.*

For security policy, the security administrator (or system designer using this methodology) must decide who has the authority to change a security attribute. This is a policy issue that has major ramifications. Should it only be the "owner" of the data that has the capability or should it only be the security administrator? This paper will not address such issues since they would be implementation specific.

Record migration occurs in ABDM when a record is reassigned to a different cluster (or new cluster) as a result of modification. The system as currently designed handles this migration automatically. If the security policy and system allows modification of security attributes, records could migrate based upon change of classification.

VI. IMPLEMENTATION EXAMPLE

The purpose of this section is to demonstrate the implementation of the methodology using the sample database of Appendix A.

A. THE META DATA

The following tables constitute the meta-data for this example *(all classifications are entered for example purposes only; all data has been created by author or extracted from open sources):*

Attribute	Attribute Type	DDIT Entry
RADIUS	A	D11
PLANE-Type	B	D21
COUNTRY	B	D31
FILE	B	D41

(a) Attribute Table (AT).

Id	Descriptor
D11	0 ≤ RADIUS ≤ 400
D12	401 ≤ RADIUS ≤ 600
D13	601 ≤ RADIUS ≤ 800
D14	801 ≤ RADIUS ≤ 1000
D15	1001 ≤ RADIUS ≤ 1200
D16	1201 ≤ RADIUS ≤ 2000
D21	PLANE-Type = Fighter
D22	PLANE-Type = Bomber
D23	PLANE-Type = Reconn
D31	COUNTRY = U.S.A.
D32	COUNTRY = U.S.S.R.
D41	FILE = Aircraft

(b) Descriptor-to-Descriptor-Id Table (DDIT).

Id	Desc-Id Set	Record-Id.
C1	{D13,D21,D31,D41,TS,CW1}	R1
C2	{D12,D21,D31,D41,TS,CW1}	R8
C3	{D14,D21,D32,D41,TS,CW1}	R20
C4	{D13,D21,D32,D41,TS,CW1}	R17
C5	{D12,D21,D32,D41,TS,CW1}	R30

(c)The TOP SECRET-with-codword1 Cluster-Definition Table (CDT)

Id	Desc-Id Set	Record-Id.
C6	{D13,D21,D31,D41,TS,NCW1}	R2,R13
C7	{D12,D21,D31,D41,TS,NCW1}	R9
C8	{D12,D21,D32,D41,TS,NCW1}	R21,R24,R27,R31

(d)The TOP SECRET-without-codeword1 CDT

Id	Desc-Id Set	Record-Id.
C9	{D13,D21,D31,D41,S,CW1}	R3
C10	{D12,D21,D31,D41,S,CW1}	R10
C11	{D13,D21,D32,D41,S,CW1}	R18
C12	{D12,D21,D32,D41,S,CW1}	R22,R25,R28,R32

(e) The SECRET-with-codeword1 CDT

Id	Desc-Id Set	Record-Id.
C13	{D13,D21,D31,D41,S,NCW1}	R4,R14
C14	{D12,D21,D31,D41,S,NCW1}	R11
C15	{D13,D21,D32,D41,S,NCW1}	R19
C16	{D12,D21,D32,D41,S,NCW1}	R23,R26,R29,R33

(f) The SECRET-without-codeword1 CDT

Id	Desc-Id Set	Record-Id.
C17	{D13,D21,D31,D41,C,CW1}	R6

(g) The CONFIDENTIAL-with-codeword1 CDT

Id	Desc-Id Set	Record-Id.
C18	{D13,D21,D31,D41,C,NCW1}	R5,R15
C19	{D12,D21,D31,D41,C,NCW1}	R12

(h) The CONFIDENTIAL-without-codeword1 CDT

Id	Desc-Id Set	Record-Id.
C20	{D13,D21,D31,D41,U,NCW1}	R7,R16

(i) The UNCLASSIFIED-without-codeword1 CDT

Figure 6.1 The Meta-data for Security Example

B. DISCUSSION

The database used for this example uses Soviet and American fighter aircraft. A database of this organization might be developed by an organization that is interested in keeping statistics on aircraft which can be keyed by combat radius in nautical miles (the attribute RADIUS). Such an organization might be very appropriate to a carrier battle group operating near a Soviet littoral. The battle group commander could want to know which fighters have the range to reach his task force. In such a database, the record body of each record might contain where these fighters are based, etc., and this information would be classified according to source, sensitivity, and effects of disclosure. CW1 means the record contains codeword1 and NCW1 means it does not.

1. Database generation

A primary consideration with the security methodology proposed is database generation. That is, what are the requirements and overhead necessary to effect the partitioning desired?

This research was conducted in support of Department of Defense (DoD) security requirements. In DoD, classified data is caveated at a granularity where the material of different sensitivities is distinguishable when they are presented together in a document form. That is, in a TOP SECRET document, paragraphs containing only SECRET material are marked as SECRET within the document. This philosophy also applies to message traffic, where at a minimum, each paragraph is caveated.

Given this data environment, it is envisioned that a database administrator (or an ADP device) would insert data using the technique outlined in Section V by creating a separate record for each classification category. In our sample database,

information about the Soviet Su-27 Flanker resides in four records (30 through 33). One record is TOP SECRET and contains a codeword1, another is TOP SECRET and does not contain codeword1 material, one is SECRET with codeword1 and the last is SECRET without codeword1.

It is possible that originally the data from all four records were present together in a single message classified at the highest classification (TOP SECRET with codeword). In order to satisfy multi-level security requirements, the database administrator (or an ADP device) divided the text of the message into four pieces based upon the security requirements of each part.

When the proposed security method is implemented, the individuals with a TOP SECRET codword1 access will see all four records if they have queried the database specifying the necessary directory attributes--just as if they were reading the original message. However, a user with TOP SECRET but no codeword access would only see records 31 through 33.

2. Some observations

You will note that there are 33 records distributed among 17 clusters. One of the design concerns with ABDM and the security methodology is how to devise a DDIT that will maximize clustering in order to keep the size of the CDTs relatively small and still satisfy access precision requirements.

In the security part of the thesis I have not discussed non-directory attributes. In an actual implementation, non-directory keywords can play an important role. For example, in this database we may want to add a capability to further specify that we want only fighter aircraft with "look-down shoot-down" capabilities, but that because

of the reorganization necessary in the meta-data we want this to be a non-directory keyword.

In this case, ABDM will use the meta data without change; however, the database may bring in records that do no satisfy the non-directory keyword requirements. The check to determine whether the record does meet the requirements is done in the primary memory. Therefore we have less than absolute precision.

The issue of record-to-clustering ratios will have to be analyzed for every implementation and will require data analysis by the database administrator.

In the case of the example given, analysis was done a priori of the development of DDIT--of particular concern are type A attributes, the range attributes, since by choosing the ranges well a database administrator can maximize clustering and thus maximize the efficiency of searching the meta-data files.

Combat radius ranges were examined as given in "Jane's All the Worlds Aircraft" for the fighter aircraft from the countries to be used in the sample database (COUNTRY a directory attribute). It was evident that for Soviet fighters there was a definite group with a combat radius between roughly 400 and 600 nautical miles and another group between 600 and 800. As a result, C8, C12 and C16 all fit the descriptor requirements for four records. More American fighters in the 600 to 800 radius ranges could have also been included but were not in order to keep Appendix A reasonably short.

In summary, the type A attributes of ABDM are invaluable in designing a database to maximize meta-data (directory) look up. Such consideration is extremely important to a secure database since the partitioning is finer than in a normal database

and the size of the CDT will tend to be much larger because of the requirement for a separate CDT for every security attribute combination.

C. A SAMPLE RETRIEVE OPERATION

Assume the user logs on to the system and is identified as having access to SECRET with codeword1 material (it is implied the user has access to classifications equal-to and below this level).

The user then types in:

RETRIEVE((FILE = Aircraft) and (350 < RADIUS < 601) and
(COUNTRY = U.S.S.R.) and (PLANE-Type = Fighter).

Steps 1 and 2 of the execution steps described in Chapter IV are unchanged: The query is processed against AT and the directory attributes are processed against DDIT.

With the completion of step 2, we have the descriptor-id groups that cover the query. These are:

{D11,D21,D32,D41}
{D12,D21,D32,D41}

At Step 3, the system then brings into the main memory only those CDTs this user is cleared for. We will ignore implementation issues such as the fact that in actual implementations all the tables would probably not be brought into the main memory at the same time. For this example, these are tables (refer to Figure 6.1) (e),(f),(g),(h) and (i). CDTs (a),(b), (c), and (d) are never brought into the main memory.

Step 3, the cluster determination step, then proceeds normally as in ABDM. Using the descriptor-id groups we search the CDTs for a corresponding descriptor-id set (cluster). In this case, we match with C12 and C16. Note there would be a match with

C5 and C8 if the user had access to TOP SECRET with codeword1 material. However, the CDTs giving the address to these records is never searched.

An optional security check is to confirm the user's security access is greater-than or equal to the security attributes specified in the CDTs.

Step 4, determining the record ids and addresses, is unchanged.

Step 5, selection of the database records, is also unchanged except, again, there is an option of checking the security attributes contained in the record in the main memory to again ensure the user is properly cleared. Note that it is also at this point that even if only directory attributes are used, we can not be assured of absolute precision with ABDM. If the query had asked for all Soviet fighter aircraft with combat radii between 415 and 601, the same clusters would have been selected and the records brought into the main memory. However, records 28 and 29 would not be given to the user because the combat radius of the Fishbed is 410. The security requirements are not affected by this lack of precision.

Step 6, processing the database records, is unchanged.

VII. CONCLUSIONS

A. SUMMARY OF CONTRIBUTION

This thesis has outlined a multilevel security mechanism that is conceptually both efficient and secure. It is efficient because it can operate with close to absolute precision and not pass through unwanted data. A critical point is that the method is secure for the very same reasons it is efficient.

The approach outlined is an integrated database system approach. Both the meta-data and base-data are organized and controlled to support partitioning of the data into disjoint sets of records (equivalence classes). ABDM supports this system approach.

The base-data are partitioned into sets of records called *clusters*, each of which has the same set of directory attribute descriptors. The security methodology has further specified that each cluster have the same set of security attribute descriptors. The security attributes are treated in much the same manner as directory attributes are.

The security methodology partitions the meta-data also. This is accomplished by using separate CDTs, one for each different aggregate of security attribute descriptors. CDTs are referenced in accordance with the user's access rights. This construction supports precision and efficiency objectives at the meta-data level in the same way it is supported for the base-data. Only those CDTs which address records the user is cleared for are brought into the main memory for processing.

At this conceptual level, this method appears superior to both the query modification and the view mechanism. Query modification requires a system overhead in

query processing not incurred by this model, and the view mechanism still suffers from the pass-through problem, thereby losing efficiency and providing a potential for security compromise in case of hardware failure.

B. REMAINING ISSUES AND FUTURE WORK

Two primary areas of work remain: a formal proof that the implementation of this methodology is secure; and the actual implementation of the conceptual model. A proof would entail construction of a formal semantic model of ABDM and the additions proposed in this paper. At later stages of development, the actual implementation design would have to be incorporated into this proof to evaluate its sufficiency. This area will not be discussed further.

There are fundamentally two categories of implementation depending on motivation: implementation which modifies ABDM in order to benchmark efficiency and serve as a medium for further research; and the second, which is a secure implementation and would be convergent with the formal proof already mentioned.

The former, implementation without concern of proof, is a matter of ABDM system level modification in at least the following areas:

1. Support of multiple CDTs;

2. Addition of system level look-up tables to address CDTs;

3. Modification of CDT to incorporate entry of security attribute value for each security attribute used by the system;

4. System check of password that provides associated access levels that can be equated to CDT address table entries (2 above);

5. Additional check of security values in descriptor id set upon match with cluster, and additional check of security attribute values in record upon retrieval (optional);

6. Modification of ABDM's three interactive dialogues used in the creation of a new database. Decisions will have to be made as to whether security attributes are entered into the templates (first dialogue) and modifications made to ensure the system treats these differently from non-security attributes (e.g., they do not appear in AT or DDIT tables). In the second dialogue, the descriptors for the security attributes will have to defined and protected. In the third dialogue, creation of multiple records will have to be controlled so that Rule 1 of Chapter V is enforced, i.e., that every record in the database must contain all the security attributes used by the system and a value for each attribute.

7. ABDM will have to be modified to incorporate security decisions concerning update of the security attribute values in records.

APPENDIX A - THE SAMPLE DATABASE

Most data extracted from "JANE'S ALL THE WORLDS AIRCRAFT"
(Some data are fabricated for the purpose of making the example)
ALL SECURITY CLASSIFICATIONS ARE FOR ILLUSTRATION ONLY

1. (< FILE, aircraft >, < PLANE-Type, fighter >, < COUNTRY, U.S.A. >, < RADIUS, 800 >, < CLASSIFICATON, TS >, < CODEWORD1, Y >, {Text: The McDonnel Douglas F-15C Eagle are based in...targeting...})

2. (< FILE, aircraft >, < PLANE-Type, fighter >, < COUNTRY, U.S.A. >, < RADIUS, 800 >, < CLASSIFICATON, TS >, < CODEWORD1, N >, {Text: ...})

3. (< FILE, aircraft >, < PLANE-Type, fighter >, < COUNTRY, U.S.A. >, < RADIUS, 800 >, < CLASSIFICATION, S >, < CODEWORD1, Y >, {Text:...})

4.(< FILE, aircraft >, < PLANE-Type, fighter >, < COUNTRY, U.S.A. >, < RADIUS, 800 >, < CLASSIFICATION, S >, < CODEWORD1, N >, {Text :...})

5. (< FILE, aircraft >, < PLANE-Type, fighter >, < COUNTRY, U.S.A. >, < RADIUS, 800 >, < CLASSIFICATION, C >, < CODEWORD1, N >, { Text: ...})

6. (< FILE, aircraft >, < PLANE-Type, fighter >, < COUNTRY, U.S.A. >, < RADIUS, 800 >, < CLASSIFICATION, C >, < CODEWORD1, Y >, { Text: ...})

7. (< FILE, aircraft >, < PLANE-Type, fighter >, < COUNTRY, U.S.A. >, < RADIUS, 800 >, < CLASSIFICATION, U >, < CODEWORD1, N >, { Text: ...})

8. (< FILE, aircraft >, < PLANE-Type, fighter >, < COUNTRY, U.S.A. >, < RADIUS, 575 >, < CLASSIFICATION, TS >, < CODEWORD1, Y >, { Text: The McDonnell Douglas F/A-18A Hornet...})

9. Same as Record 8 with CLASSIFICATION TS and no codeword1 (in actual implementation text would be different).

10. Same as Record 8 with CLASSIFICATION S and codeword1 present.

11. Same as Record 8 with CLASSIFICATION S and no codeword1.

12. Same as Record 8 with CLASSIFICATION C and no codeword1.

13. (< FILE, aircraft >, < PLANE-Type, fighter >, < COUNTRY, U.S.A. >, < RADIUS, 725 >, < CLASSIFICATION, TS >, < CODEWORD1, N >, { Text: The Grumman F-14A Tomcat...})

14. Same as Record 13 with CLASSIFICATION S and no codeword1.

15. Same as Record 13 with CLASSIFICATION C and no codeword1.

16. Same as Record 13 with CLASSIFICATION U and no codeword1.

17. (< FILE, aircraft >, < PLANE-Type, fighter >, < COUNTRY, U.S.S.R. >, < RADIUS, 610 >, < CLASSIFICATION, TS >, < CODEWORD1, Y >, { Text: The MiG-25A Foxbat...})

18. Same as Record 17 with CLASSIFICATION S and codeword1 present.

19. Same as Record 17 with CLASSIFICATION S and no codeword1.

20. (< FILE, aircraft >, < PLANE-Type, fighter >, < COUNTRY, U.S.S.R. >, < RADIUS, 810 >, < CLASSIFICATION, TS >, < CODEWORD1, Y >, { Text: The MiG-31 Foxhound...})

21. (< FILE, aircraft >, < PLANE-Type, fighter >, < COUNTRY, U.S.S.R. >, < RADIUS, 430 >, < CLASSIFICATION, TS >, < CODEWORD1, N >, { Text: The Mig-29 Fulcrum...})

22. Same as record 21 with CLASSIFICATION S and codeword1 present.

23. Same as record 21 with CLASSIFICATION S and no codeword1 present.

24. (< FILE, aircraft >, < PLANE-Type, fighter >, < COUNTRY, U.S.S.R. >, < RADIUS, 475 >, < CLASSIFICATION, TS >, < CODEWORD1, N >, { Text: The MiG-23 Flogger...})

25. Same as record 24 with CLASSIFICATION S and codeword1 present.

26. Same as record 24 with CLASSIFICATION S and no codeword1 present.

27. (< FILE, aircraft >, < PLANE-Type, fighter >, < COUNTRY, U.S.S.R. >, < RADIUS, 410 >, < CLASSIFICATION, TS >, < CODEWORD1, N >, { Text: The MiG-21 Fishbed...})

28. Same as record 27 with CLASSIFICATION S and codeword1 present.

29. Same as record 27 with CLASSIFICATION S and no codeword1 present.

30. (< FILE, aircraft >, < PLANE-Type, fighter >, < COUNTRY, U.S.S.R. >, < RADIUS, 595 >, < CLASSIFICATION, TS >, < CODEWORD1, Y >, { Text: The Sukhoi Su-27 Flanker...})

31. Same as record 30 with CLASSIFICATION TS and no codeword1 present.

32. Same as record 30 with CLASSIFICATION S and codeword1 present.

33. Same as record 30 with CLASSIFICATION S and no codeword1 present.

LIST OF REFERENCES

[1] Hsiao, D. K. and Harary, F., "A Formal System for Information Retrieval from Files," *Communications of the ACM Vol. 13, No. 2*, (February 1970).

[2] Kerr, D. S., Hsiao, D. K., and Madnick, S. E., *Computer Security* (Academic Press, 1979).

[3] Wong, E. and Chiang, T. C., "Canonical Structure in Attribute Based File Organization," *Communications of the ACM Vol. 14 No. 9*, (September 1971).

[4] Wood, C., Fernandez, E. B., and Summers, R. C., "Data Base Security: Requirements, Policies, and Models," *IBM Systems Journal Vol. 19, No. 2*, (1980).

[5] "Department of Defense Trusted Computer System Evaluation Criteria," Department of Defense Computer Security Center, CSC-STD-001-83, 15 August 1983.

[6] Hsiao, D. K. and McGhee, R. B., "Database Computers as Inference Engines," Technical report, NPS52-87-046, The Naval Postgraduate School, Monterey, California, October 1987.

[7] Hsiao, D. K., "Database Security," Technical report, NPS52-87-048 , The Naval Postgraduate School, Monterey, California, November 1987.

[8] Hopcroft, J. E. and Ullman, J. D., in *Introduction to Automata Theory, Languages, and Computation*, (Addison-Wesley, 1979), 1st edition .

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Chief of Naval Operations Director, Information Systems (OP-945) Navy Department Washington, D.C. 20350-2000	1
4.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	2
5.	Curriculum Officer, Code 37 Computer Technology Naval Postgraduate School Monterey, California 93943-5000	1
6.	Professor David K. Hsiao, Code 52Hq Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	2
7.	Professor Steven A. Demurjian Computer Science & Engineering Department The University of Connecticut 260 Glenbrook Road Storrs, Connecticut 06268	1
8.	Commander, Naval Security Group Command G30 3801 Nebraska Avenue N.W. Washington, D.C. 20390-5210	2

9.	Department of Defense Computer Security Center Fort George G. Meade, Maryland 20755	2
10.	Lieutenant Gregory S. Hoppenstand 7769A Nelson Loop Fort George G. Meade, Maryland 20755	2
11.	Captain J. Leonard, USN 15729 Edgewood Drive Dumfries, Virginia 22026	1

END

DATE

FILMED

8-88

DTIC