University of Wollongong
Research Online

University of Wollongong Thesis Collection

University of Wollongong Thesis Collections

1996

# Access control in object-oriented databases

Ahmad Baraani-Dastjerdi
*University of Wollongong*

Recommended Citation
Baraani-Dastjerdi, Ahmad, Access control in object-oriented databases, Doctor of Philosophy thesis, Department of Computer Science, University of Wollongong, 1996. http://ro.uow.edu.au/theses/1290

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au

**NOTE**

This online version of the thesis may have different page formatting and pagination from the paper copy held in the University of Wollongong Library.

**UNIVERSITY OF WOLLONGONG**

**COPYRIGHT WARNING**

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site. You are reminded of the following:

Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

UNIVERSITY OF WOLLONGONG

# ACCESS CONTROL IN OBJECT-ORIENTED DATABASES

A thesis submitted in fulfilment of the
requirements for the award of the degree

**Doctor of Philosophy**

from

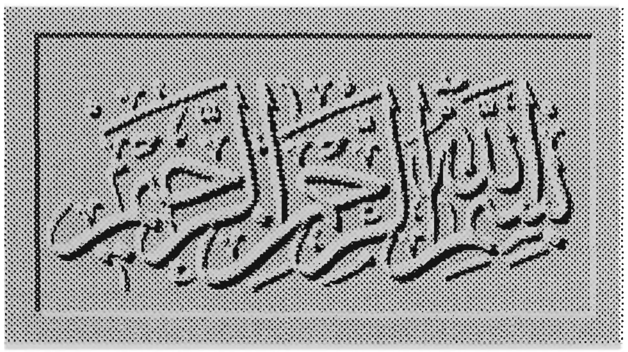
UNIVERSITY OF WOLLONGONG

by

**Ahmad Baraani-Dastjerdi, M.Sc**

Department of Computer Science

1996



In the name of God, the Beneficient, the Merciful

© Copyright 1996

by

Ahmad Baraani-Dastjerdi

All Rights Reserved

*Dedicated to*

*my family*

# Certificate of Originality

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of a university or other institute of higher learning, except where due acknowledgment is made in the text.

Ahmad Baraani-Dastjerdi

# Abstract

In a multi-user environment with a large shared database, it is necessary that the security of data in the database is considered. To enforce security of data in a database, we start with an *access control model.* The model defines *which users have what privileges to which information.* There are three different types of access control policies: *discretionary access control (DAC), mandatory access control (MAC),* and *role-based access control (RBAC).* A *discretionary access control* specifies users' privileges to different system resources, including their ability to transfer their privileges to other users. In a *mandatory access control,* the access of data by users is based on authorized security clearance levels. MAC policies are of concern in *multi-level databases,* which are databases that contain information of different security levels. *A role-based access control* manages access to data based on a user's responsibility within an organization. Each role has an associated collection of privileges. This collection is automatically transferred to a subject who plays the role

Most of the current access control models in database systems are developed for relational databases. Since the object-oriented database (OODB) model differs substantially from the relational model, results obtained for relational databases as well as models proposed for relational databases are not necessarily applicable to OODB systems. Amongst other issues, inheritance and the encapsulation of methods in the database information pose challenges in designing new authorization models for OODB systems. It is therefore necessary to extend the research on secure databases to include the O-O model. This thesis presents a study of security in OODB systems. Access control protection forms a substantial component of this work.

Principles from the O-O model are used to express rules for computing implicit privileges from explicit ones. It requires an efficient mechanism which evaluates implicit rights each time an access is requested. A cryptographic mechanism which is based on unique and secure access keys for each entity (object or class) is proposed. The proposal ensures that access keys for implicit authorizations were derived from related entities by applying pseudo-random and SIFF functions during query processing. The security of the system is based on the difficulty of predicting the output of pseudo-random functions and finding extra collisions for SIFF functions. Both are known to be computationally difficult.

Another major requirement for the access control model is the implementation of content-dependent authorization. The content-dependent authorization incorporates the value of attributes in the access control model. The accessible data are determined by checking the requested attributes. A content-dependent access control model based on views is proposed. Rules for computing an implicit authorization from the explicit ones are also formulated.

Finally, a new design approach for a secure multi-level OODB system based on views is proposed. The central idea is to provide the user with a *multi-level view* derived from a single-level secure OODB system. Hence the database operations performed on the multi-level views are decomposed into a set of operations on the single-level objects. They can then be implemented on any conventional mandatory security kernel.

# Publications From This Thesis

1. A. Baraani-Dastjerdi, J. Pieprzyk, and R. Safavi-Naini, "A Multi-level View Model for Secure Object-Oriented Databases," *Accepted for publication by Data & Knowledge Engineering*, 1996.

2. A. Baraani-Dastjerdi, J. R. Getta, J. Pieprzyk, and R. Safavi-Naini, "A Cryptographic Solution to Discretionary Access Control in Structurally Object-Oriented Databases," in *Proceedings of the 6th Australian Database Conference (ADC'95)* (R. Sacks and J. Zobel, eds.), vol. 17(2), (Adelaide, Australia), pp. 36–45, Australian Computer Science Communications, Jan. 1995.

3. A. Baraani-Dastjerdi, J. Pieprzyk, R. Safavi-Naini, and J. R. Getta, "A Cryptographic Mechanism for Object-Instance-Based Authorization in Object-Oriented Database Systems," in *Proceedings of The 14th International Conference on Object-Oriented & Entity Relationship Modeling (OOER'95)* (M. P. Papazoglou, ed.), vol. 1021 of *Lecture Notes in Computer Science*, (Queensland, Australia), pp. 44–54, Springer-Verlag, Dec. 1995.

4. A. Baraani-Dastjerdi, J. Pieprzyk, R. Safavi-Naini, and J. R. Getta, "A Model of Authorization for Object-Oriented Databases based on Object Views," in *Proceedings of The 4th International Conference on Deductive and Object-Oriented Databases* (T. Ling, A. Mendelzon, and L. Vieille, eds.), vol. 1013 of *Lecture Notes in Computer Science*, (Singapore), pp. 503–520, Springer-Verlag, Dec. 1995.

5. A. Baraani-Dastjerdi, J. Pieprzyk, and R. Safavi-Naini, "Modeling A Multi-level Secure Object-Oriented Database Using Views." Pre-proceedings of the Australian Conference on Information Security and Privacy, The University of Wollongong, NSW, Australia, May 24-26, 1996 (accepted).

# Acknowledgments

First, I would like to express my deepest gratitude to my supervisor Associate Professor Josef Pieprzyk for his studious guidance and support throughout my entire period of this study. The next two people whom I would like to thank are Doctor Reihaneh Safavi-Naini and Doctor Janusz R. Getta who acted as my *de facto* supervisors in the absence of A/Professor Pieprzyk. They have been very generous with their time in helping me with various aspects of my thesis. I held fruitful discussion with them at different stages of this research.

I would also like to express my appreciation of the support and assistance provided by the members of the Centre for Computer Security Research in the Department of Computer Science. In particular, I would like to thank Professor Jennifer Seberry for her helpful suggestions and encouragement. I am thankful to the administrative and support staff in the department and the Centre for their help, in particular Ms. Margot Hall.

I am greatly indebted to my wife and my children who have been a source of encouragement and support during the difficult times in this work. Their patience and understanding during the long hours of my work allowed me to finish this study. I sincerely hope that this experience and achievement will bring positive happenings to our lives in the future. Other people that I must thank are my family for their love, encouragement and support in my studies throughout my school and university years.

Finally, I thankfully acknowledge the financial support provided by the government of the Islamic Republic of Iran. I would not have been in this program without their support.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Introduction

The increasing development of information technology in the last decade has led to the widespread use of computer systems in various public and private organizations such as banks, universities, manufacturing or service companies, hospitals, libraries, etc. This means that organizations maintain more and more computerized information in databases, and they increasingly depend on that information for their correct functioning. Hence, information needs protection against unauthorized access and any possible threats which might be launched by insiders and outsiders, either malicious or accidental. This explains the need for secure databases which are of concern to the *database security.* The database security aims at preserving the *secrecy, integrity,* and *availability* of the information stored in it. **Secrecy** ensures that database information is readable to authorized users only. It ensures the **confidentiality** of the information. **Integrity** of information covers methods and techniques which are used to protect information against illegal or accidental modification. **Availability** of information ensures that authorized users can have access to information whenever they need them [65].

In order to obtain security in a database, Denning [65] lists four kinds of controls: *access control, information flow control, cryptographic control,* and *inference control.* The *access control* ensures that any users access to the system is authorized according to access rules given by the security policies. The *information flow control* makes sure that the protected information "contained" in some objects does not flow explicitly (through copy) or implicitly to other protected objects of lower security levels. It also regulates how the information is accessible (irrespective where it is stored). The *cryptographic control* makes data unintelligent to anybody except someone who knows the correct cryptographic key. The *inference control* protects information against its disclosure via different ways of deduction.

Research effort regarding secure databases has mainly focused on relational databases. Since the object-oriented database (OODB) model differs substantially from the relational model, results obtained for relational databases as well as models proposed for relational databases are not necessarily applicable to OODB systems. Amongst other issues, inheritance and the encapsulation of methods in the database information pose challenges in a secure OODB system. It is therefore necessary to extend the research on secure databases to include the O-O model. Some works have been done so far to address some issues concern of O-O model see for example [4, 29, 30, 32, 78, 80, 97, 130, 169].

The access control model for OODB systems which supports different granularity such as class-based, object-based, instance-variable-based, and/or content-based authorization, still needs more investigation. There is also a need for the design of a more efficient mechanism to derive implicit privileges during query processing.

## 1.2 Goals and Outline of the Thesis

The primary goal of this work is to present the results of my investigation and study of data security for object-oriented database (OODB) systems, and to design a more efficient mechanism to enforce access control and new security models for open problems such as instance-based, and content-based authorizations in OODB systems. The thesis is arranged as follows.

Chapter 2 deals with the basic concepts of OODB model. The key concepts of O-O data model (such as types, classes, objects, complex objects, aggregation, inheritance, encapsulation, and methods) are discussed. Database functionalities such as persistence, secondary-storage management, security, authorization, concurrency, and recovery are described. The view concept is also introduced in this chapter.

Chapter 3 provides a brief overview of the database security. Some basic facts, methods, and terminology are discussed. Security threats, requirements, and problems that arise in the pursuit of meeting these requirements for security in OODB systems are illustrated. The chapter also shows how much effort has been put into the design of access control models. The two major access control approaches, mandatory access control (MAC) and discretionary access control (DAC), are illustrated. Access matrix and Bell-LaPadula models as the most widely used for access control are presented. Then, several proposals for discretionary and mandatory security models for the protection of conventional databases and OODB systems are presented and some drawbacks of the solutions are pointed out.

Chapter 4 describes a cryptographic mechanism for the discretionary access control in OODB systems. The solution applies pseudo-random functions, sibling intractable function families (SIFF), and an authorization class (instead of access control lists). Pseudo-random functions and SIFF are used in such a way that unique and secure access keys for each entity (object or class) can be derived for the objects which are in relationship or by users who are members of the proper group. The security of the system is based on the difficulty of predicting the output of pseudo-random functions and finding extra collisions for SIFF functions, both of which are known to be computationally difficult.

In Chapter 5, we describe a solution for content-based authorization in OODB systems employing views. The chapter provides discretionary security requirements for authorization systems and presents rules for computing an implicit authorization from the explicitly defined one along with the three authorization dimensions, users, access privileges, and views.

Chapter 6 shows how to use the view concept to implement a multi-level security policy on the top of a single-level OODB system. The chapter describes the multi-level view, the content and context-classification, and the dynamic classification and ways of their handling in context of the view model. Finally the aggregation and inference problems are investigated in the view context.

Chapter 7 concludes the thesis and discusses some possible directions for the future work.

Appendix A describes a method of the authentication of data in database systems based on the use of pseudo-random functions and the SIFF suggested by Hardjono, Zheng and Seberry [101].

In Appendix B, a sketch of the implementation of SIFF function is presented.

Parts of this work have been published as references [17, 18, 20, 21, 19]. The papers represent a joint collaboration with my supervisor A/Prof. J. Pieprzyk, and my co-supervisors, Dr. R. Safavi-Naini and Dr. J. R. Getta. The problem definition and the preliminary versions of all four papers were the result of my own works. Further modifications of some technical points and improvements of their presentations were handled through fruitful discussions with A/Prof. Pieprzyk, Dr. Safavi-Naini, and Dr. Getta. Hence it would be fair to say that I did at least 85 per cent of the whole works.

# Chapter 2

# Object-Oriented Database System Concepts

## 2.1 Introduction

Since the theme of this thesis is access control methods in the object-oriented database (OODB) systems, we need an object-oriented (O-O) framework. This is required by the fact that there is no standard O-O data model. Although ODMG'93 [11, 135] and others are striving to achieve an international agreement towards acceptable standards, a prospect of working out standards for the O-O data model still looks distant. In this chapter, we first present some O-O concepts which, in our view, can be incorporated within any O-O data model. It is not our intention to specify all details of O-O data model. We are going to present concepts and properties which are used further in our work. The model discussed here has been widely accepted (see for example [7, 10, 11, 12, 23, 14, 28, 45, 47, 131, 135, 126]) in the O-O modeling world. We discuss key concepts for an O-O data model such as types/classes, objects, complex objects, aggregation, inheritance, encapsulation, and methods.

In recent years there has been considerable efforts in the research and development of O-O databases (ORION [124], ONTOS [164], $O_2$ [74], Objectivity/DB [159], and VERSANT [218]). Despite these efforts, the term OODB system is not well defined. The ambiguity in the term OODB system is largely a result of different emphasis on either the database or O-O programming language side. Historically, the notion of an OODB system has its roots in O-O programming languages. In fact, some O-O languages and their implementations have been extended to incorporate database functionality such as persistence, secondary-storage management, security, authorization, concurrency, and versions. Hence, we discuss database functionalities which every OODB system must have, and present a list of features for which OODB systems may differ and may be used as a basis for comparison with different systems.

Finally, the concept of a view is discussed because we believe that the use of views can help in solving some security problems of access control system in OODB systems. It is possible to define views with different granularity such as class-based, object-based, instance-variable-based, and/or content-based authorization. In Chapters 5 and 6, we present new models based on views for this purpose.

## 2.2 Object-Oriented Data Model-Concepts

Unlike relational database systems whose model was clearly defined by Codd [52], the OODB systems do not have any widely accepted model. There are, however, some basic O-O concepts that are generally accepted (see [7, 10, 11, 12, 14, 16, 23, 24, 28, 45, 47, 131, 135, 126]).

The basic concepts of an O-O data model include:

- *Objects* and *identities.* Each real-world entity is modeled as an object with a unique identifier.

- *Classes* and *types.* A collection of similar objects forms a class. All objects of one class contain the same properties (or instance-variables). Each object is an instance of some class.

- *Complex (or composite) objects.* The internal structure of an object is defined by class properties and their domains. The domains of properties can be complex or simple. In the case of complex domain, the value of a property can be an object or a set of objects.

- *Class hierarchies* and *inheritance.* Classes are organized into a hierarchical structure. There are two types of hierarchies which are orthogonal to each other: *class-composition hierarchy* and *class-inheritance hierarchy* [126]. The class-composition hierarchy captures the *is-part-of* relationship between a parent class and its component classes, whereas a class-inheritance hierarchy represents the *is-a* relationship between a superclass and its subclasses. Properties are inherited by the subclasses of the class in the hierarchy.

- *Encapsulation.* Each object contains and defines both methods (or procedures) and the interface that can be used to access and modify the object by other objects. The interface of an object consists of the set of operations which can be invoked on the object. Only the methods implementing operations for the object are allowed to manipulate the state of the object, i.e., users can only access the values of properties through methods.

- *Polymorphism* and *binding.* Different methods can be associated with a single operation name, leaving the system to determine which method should be used in order to execute a given operation. The overloading of operations, early binding at compilation time, and late binding at runtime are allowed.

Now, each of these concepts will be described in more detail.

### 2.2.1 Objects and Identity

In O-O systems, all real world entities are modeled as *objects.* Every object encapsulates *a state* and *a behavior.* The state of an object is implemented by *properties* (or instance-variables), and the behavior of an object is encapsulated in *methods* that operate on the state of the object. As a result, an object is something that owns "resources" and provides some services based on these resources. These resources are simple data elements (properties), and services are methods (or behaviors).

Each object is associated with a unique identifier called *object identifier* (OID) and may also be given a name.[1] The identity of an object exists independently of the values of the object properties. The system should guarantee that the OID is never reused and identifies only a single object during its lifetime.

[1]The idea of unique system-generated identifiers for objects is necessary to eliminate the shortcomings of primary keys and value-based models. The use of unique identifiers for objects has been a feature of semantic data models for a number of years under the name of *surrogates* [53]. For the detailed discussion of problems of identifying an entity on the value of an attribute that is called primary key and advantage of using OIDs over key, the reader is directed to [28, 47].

Object identity generates at least two notions of equality for objects. The first is *identity equality:* if they have the same OIDs. The second is *value equality:* two objects are equal if they have the same value. Hence, two objects are different if they have different OIDs even when their properties have the same values.

**Definition 2.1** *An object o is a quadruple: object identifier, its name, its class, and its state that is o=(OID, oname, class, state) where OID is the unique object identifier, oname is the name of the object given by the creator, class specifies the class of which the object is an instance, and state represents the value of the object [24, 131, 155].* □

## 2.2.2 Classes and Types

The primary schema level concepts of database models are types and classes. In the relational model, tables are considered to be types (the definition of table attributes includes the type definition while rows of a table are instances of the type). In O-O data models, this issue is ambiguous, i.e., some O-O systems support only types (all systems from the Smalltalk family and derived from LISP such as Gemstone [43], Orion [124], G-Base [156], etc.); others support only classes (all systems from the C++ family, like Objectstore [128], O2 [74], etc.), and some support both.

A *type* defines the common features of a set of objects with the same characteristics. It corresponds to the notion of an abstract data type [24] and has two roles: (i) to denote structure and (ii) to identify extensions, i.e., domains of elements. In general, in type-based system, types are mainly used at compilation time to check the correctness of the program.

A *class* is a set of objects which have exactly the same internal structure and therefore the same properties and the same methods. The class defines the implementation of a set of objects, while a type describes how such objects can be used.

Often the concepts *type* and *class* are used interchangeably. But, when both are used in the same system, the type refers to the specification of the interface of a set of objects, whereas the class refers to the implementation [11, 134]. A class can implement several types. If a class implements a type, it automatically implements all the supertypes of that type.

**Definition 2.2** *A class c is a triple: class name, its structure, and its methods, i.e., c=(cname, class-struct, method-list) where cname is the name of the class, class-struct is its structure, and method-list is the list of methods that are used to access and modify instances of the class [24, 155].* □

The *class-struct* defines the properties of the instances of *cname* and their domains. The *method-list* contains a roster of methods that can access instances of *cname* and manipulate them. A method consists of two components: *signature* and *body*. The *signature* specifies the name of the method, the names and class of input arguments, and the class of output values as well as any exceptional conditions [28, 135]. Therefore the signature is the specification of the operation implemented by the method. The *body* specifies code (or procedure) to manipulate the values of input arguments, and to generate output.

## 2.2.3 Complex (or Composite) Objects

An object is described by properties (or instance-variables) which make up the descriptive part of the object. Objects with their descriptive parts only are called *objects*. An object may have internal structure that is composed of low level components such as relationships, or other objects that together form structural part of the object. Objects with internal structures are called *complex (or composite) objects.*

A complex object can be constructed using: tuples, sets, bags, lists, arrays, etc. Any system must have at least set, list, and tuple as a constructor. Any constructor should be applicable to any object. Note that the support of complex objects also requires that appropriate operators are provided to deal with such objects. For example, it must be possible to retrieve, copy, or delete an entire complex object and to maintain referential integrity among related objects.

**Definition 2.3** *A complex (composite) object is comp* $= [P_i : \rho_i(T_i), \ldots, P_k : \rho_k(T_k)]$*, where* $P_i$ *is a name of property;* $T_i$ *is a type name of the respective property and* $\rho_i$ *is an optional type constructor, e.g. set-of, collection-of, array-of, ordered list, etc. The set of type names includes the names of atomic data types like integer, real, string, etc. as well as the names of classes that have been defined earlier.* □

The domain of a property of a composite class $c$ may be another class $c'$. $c'$ can be in turn defined in terms of other classes. The set of classes in the schema is then organized in a *composition (or aggregation) hierarchy* [126]. The composition hierarchy captures the *is-part-of* relationship among objects. The class $c$ is a parent of $c'$ or $c'$ is a descendent of $c$. For example, in Figure 2.1, $c_1$ represent computers in the real world. Computers have central processing units (CPU), memory units, and I/O units. These facts are modeled in the database by a hierarchical structure where $c_1$ is the root and the classes $c_2$, $c_3$, $c_4$ create the first level of nodes. This structure can be extended to include classes $c_5, \ldots,$ $c_8$ which are children of $c_2$, $c_3$, $c_4$. The hierarchy forms a composite class in which the classes $c_2, c_3, \ldots, c_8$ are parts of the class $c_1$ (or they are descendants of class $c_1$), and

$c_1$=[CPU:$c_2$, Memory-unit:set-of ($c_3$), I/O-unit:$c_4$],
$c_2$=[ALU:$c_6$, Control-unit:$c_5$], ...

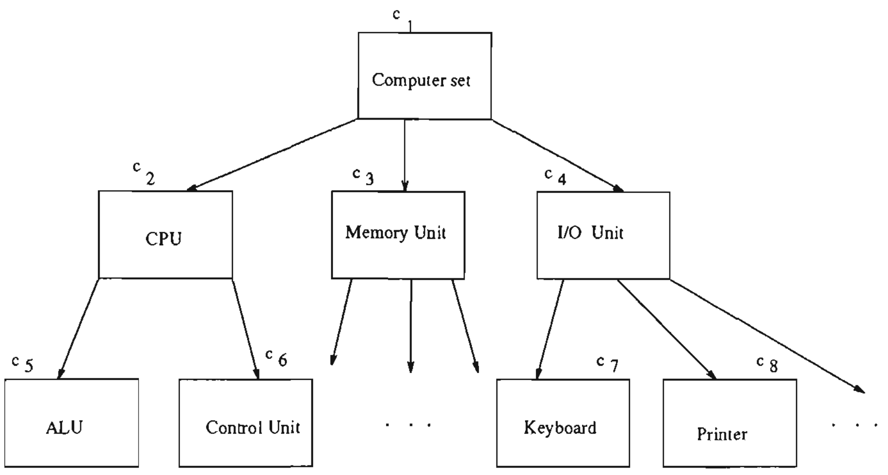


Figure 2.1: Computer set class.

It is worth noting that classes in a composition hierarchy can be defined recursively, so this is not a hierarchy in the strict sense of the word.

## 2.2.4 Inheritance and Class Hierarchy

The concept of inheritance is a mechanism of reusability. Bancilhon [14] points out that the inheritance concept is the most powerful concept of O-O data model. One of the most important advantages of inheritance is that it helps to separate shared specifications and implementations in applications. With inheritance, a class called a *subclass* can be defined on the basis of the definition of another class called a *superclass.* The subclass inherits the properties, and methods of its superclass. In addition, a subclass can have its own specific properties, and methods.

For example, imagine that we must create two classes which contain information concerning students and employees of a university. The information about student consists of *identity number, name, age, address, sex, subject, start date, graduate date,* and *the greatest average point.* The information concerning employees are: *identity number, name, age, address, sex, rank, hire date, status, profession,* and *salary.* In the relational model, two relations must be defined, one for students *STUDENT(sidno, name, age, address, sex, subject, start-date, graduate-date, GAP)* and one for employees *EMPLOYEE(eidno, name, age, address, sex, rank, hire-date, status, profession, salary).* Using the inheritance concept, it is recognized that students and employees are human beings and they then have certain common features, and other features which differentiate them. First the class *PERSON* is introduced. This class has the common properties *idno, name, age, address,* and *sex.* The classes *STUDENT* and *EMPLOYEE* contain only the properties that are different for them. The resulting inheritance hierarchy is shown in Figure 2.2. The figure shows two arrows from classes *STUDENT*

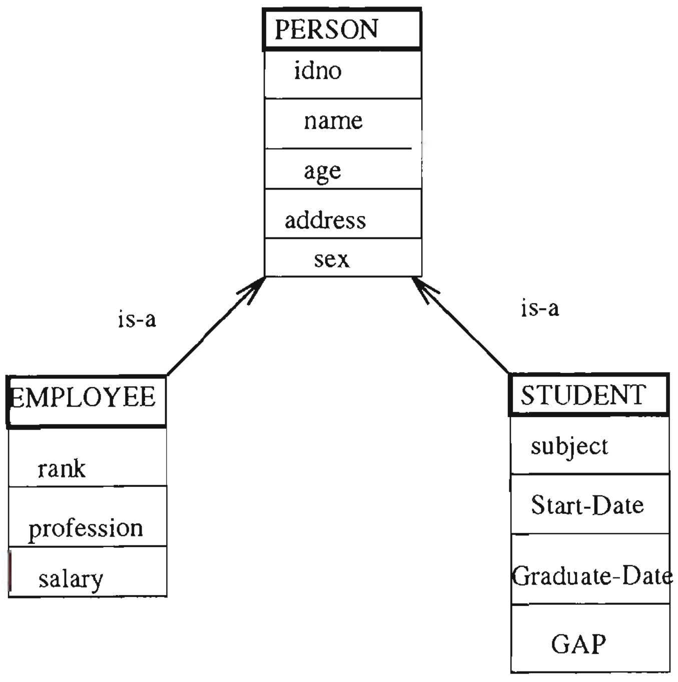


Figure 2.2: An example of inheritance hierarchy.

and *EMPLOYEE* to class *PERSON.* Both *STUDENT* and *EMPLOYEE* are subclasses of *PERSON* and vice versa *PERSON* is the superclass for *STUDENT* and

3 0009 03178495 7

*EMPLOYEE.* □

In certain systems, a class can have multiple superclasses. O-O data models which allow an object to have multiple types/classes are called *multiple inheritance* models. Other models that allow a single superclass only are called *single inheritance* models. Multiple and single inheritance are useful in reducing the number and complexity of classes required in a data schema. However, conflicts may arise if two or more superclasses have the same name for their properties but are from different domains. Generally, appropriate rules must be in place to solve such conflicts. Note that the need for multiple inheritance is rarer than the need for single inheritance [126]. Thus some O-O data models do not handle multiple inheritance. Also the implementation of the multiple inheritance is difficult.

The essential aspect of inheritance is the relationship between: superclass and its subclasses. The superclass, in turn, can be a subclass of other classes. Classes can be organized into the *inheritance (or generalization) hierarchy* , which is an orthogonal organization with respect to the *composition (or aggregation) hierarchy* [41, 126, 197]. The composition hierarchy captures the *is-part-of* relationship between a parent class and its component classes, whereas an inheritance hierarchy represents the *is-a* relationship between a superclass and its subclasses. The most significant difference of the inheritance hierarchy compared with the composition hierarchy is that the inheritance graph (representing the hierarchy) cannot be cyclic. The composition graph (representing the hierarchy) may have cycles. Inheritance hierarchy can be represented by a directed acyclic graph (or a lattice) [16]. Inheritance is often called subtyping, and subclassing.

In [10, 28, 47], four types of inheritance are identified: *specialization inheritance, substitution inheritance, classification (or inclusion) inheritance,* and *constraint inheritance.* The differences between the various inheritance concepts depend upon the significance of the class (or the type). For the sake of our discussion, assume that $c$ is a subclass of a class $c'$.

1. **Specialization inheritance.** Subclasses have more properties or methods than their superclasses. For example, *STUDENT* or *EMPLOYEE* are subclasses of *PERSON.*[2] Each of them has additional properties: *STUDENT* has *subject, Start-Date, Graduate-Date,* and *GAP* and *EMPLOYEE - rank,*

[2]See Figure 2.3 in Section 2.2.7 for the detail specification of *STUDENT*, *EMPLOYEE*, and *PERSON.*

*profession*, and *salary*. If the subclass does not override inherited properties, it may be useful to think of it as the concatenation of its own definition with the definition of its superclasses.

2. **Substitution inheritance.** The structure of $c$ and $c'$ is the same. The only difference is that $c'$ contains more methods. Any occurrence of $c'$ can be substituted by $c$.

3. **Classification (or inclusion) inheritance.** This inheritance happens if every object of class $c$ is also an object of class $c'$. This type of inheritance is based on structure and not on behavior. Subclasses just may be used as sets to classify objects, i.e., to define different extent. For example, we could classify an *EMPLOYEE* as a *TECHNICAL-STAFF*, *ACADEMIC-STAFF*, or *PROGRAMMERS* even if all three have the same structure and methods associated with them.

4. **Constraint inheritance.** Subclasses are defined using class predicates. A class $c$ is a subclass of a class $c'$ if it consists of all objects of class $c'$ that satisfy a given constraint. For example, *MANAGER* is a subclass of *EMPLOYEE* because managers don't have any more properties or methods than employees but they obey of more specific constraints, their profession are managers (*profession* = *"manager"*). As observed, each instance of *MANAGER* is an instance of *EMPLOYEE*. This type of inheritance is a subclass of the inclusion inheritance.

It is important to note that the different types of generalization are provided by the same subclass mechanism.

### 2.2.5 Encapsulation

The idea of encapsulation addresses the following:

1. the need to clearly distinguish between the structural definition of object (specification) and the implementation of operations;

2. the need of modularity;

3. the need to define a specific set of operations (or methods) for each complex object type; and

4. the protection of data part of an object from an unauthorized access [10].

Therefore, a number of O-O systems allow the definer of a new class not only to give the structure of that class, but also to determine the set of methods by which the user can access and manipulate objects of the class. We can say that the methods (or operations) are encapsulated with the specification of object. In other words, a behavioral description of object has been given, i.e., the structure and methods. The properties associated with an object are private, and only object methods may access or modify these data; the methods are publicly accessible.[3]

Some O-O systems apply strict encapsulation of object only, so publicly defined methods only are visible to users of the system. There are however cases in which encapsulation is not necessary. The usage of the system can be significantly simplified if strict encapsulation is not forced. For example, with ad-hoc queries the need for encapsulation is reduced since queries are very often expressed in terms of predicates on the values of the attributes. Therefore, some OODB systems allow direct access to properties and supply *system-defined* operations to read and modify the properties. In [28], the authors mentioned two advantages of direct access to the properties. They are:

- there is no need to develop a large number of generally conventional methods, and
- the efficiency of the applications increases as direct access is implemented using system-provided operations.

Obviously, the violation of encapsulation can cause problems. Un-authorized access to the values of properties may occur. Authorization systems can be used to control access to certain properties and methods.

A problem of considerable importance concerns whether code of methods of derived classes (subclasses) must have free access to the properties and methods defined in their superclasses. Where inheritance applies, the set of properties of

[3]There are two views of encapsulation: the programming language view, and the database view [10, 28, 145]. Encapsulation in programming language derives from the concept of abstract data types. In abstract data-type declarations, an object has the interface part and the implementation part. The interface part is the only visible part of the object and provides methods to access the object. The data structure is part of the implementation and is not visible. In databases, it is not clear whether the structure is part of the interface or not. Some databases allows data structure to be in the interface and publicly visible.

pp. 15-31 missing from original copies of this thesis

# Chapter 3

# Security In Databases

## 3.1 Introduction

Information is a critical resource in today's enterprises, whether they are military, industrial, commercial, educational, medical, etc. These organizations are now automating not only their basic operational functions, such as invoicing, payroll, and stock control, but also management-support functions such as sales forecasting, budgeting, and financial control. In order to support these functions, enterprises maintain more and more computerized information in databases, and they increasingly depend on that information for their correct functioning. The continued successful operation of an enterprise demands that:

1. *Confidential data is available only to authorized persons, so that privacy requirements are satisfied and the sensitive information is protected.*

2. *The data accurately reflects the state of the enterprise, that is, the data is protected against either malicious or accidental modification.*

These requirements are of the concern of *database security.* Database security comprises a set of measures, policies, and mechanisms to provide *secrecy, integrity,* and *availability* of data and to protect the system from possible attacks which might be launched by insiders and outsiders, either malicious or accidental [46, 176]. The aim of *secrecy* (or confidentiality) is to keep information unreadable for outsiders while making it available for *authorized* users. *Integrity* of information covers methods and techniques to protect information against illegal modification. *Availability* of information ensures that authorized users can have access to information whenever they require it [65]. To achieve security in the database environment, it is necessary first to identify threats so later a selection of proper security *policies* and *mechanisms* can be made. Security policies define *what* the security system is expected to do [163]. Security mechanisms define *how* the security system should achieve the security goals [25].

In order to obtain database protection, Denning [65] lists four kinds of control: *access control, information flow control, cryptographic control,* and *inference control.* An *access control* ensures that all direct accesses to the system are authorized according to access rules given by the security policies. The access control governs who can access objects. Once the access is granted, the involvement of the access control ends. Often, leakage of information happens not because of a defective access control, but as the result of lack of a proper policy about information flow. The *information flow control* makes sure that protected information "contained" in some objects does not flow explicitly (through copy) or implicitly into less protected objects, and regulates how the information is accessible (irrespective where it is stored). The *cryptographic control* makes data unintelligent to anybody except someone who knows the correct cryptographic key. The *inference control* protects information against its disclosure via different ways of deduction.

In this chapter, a survey of the access control models is presented. The emphasis is put on access control models in OODB systems. The survey discuss neither the application of cryptographic techniques in databases security nor inference control. For references, see [3, 46, 65, 103, 132, 166, 168, 191, 224].

Sections 3.2 and 3.3 state threats to database security and the security requirements for databases, respectively. Section 3.4 describes the concept of the access control, the policy choices, and classifies the access control policies. In Sections 3.5 and 3.6, discretionary access control in conventional databases and OODB systems are discussed. Sections 3.7, 3.8, and 3.9 are devoted to mandatory access control (MAC). The Bell-LaPadula security model, MAC models in conventional databases, and MAC models in OODB systems are presented. A brief discussion of role-based access control is presented in Section 3.10. Three types of architecture for multi-level databases are discussed in Section 3.11. Sections 3.12 and 3.13 provide a general concepts of inference control and cryptographic control. Section 3.14 summaries the chapter.

## 3.2 Threats to Database Security

Usually, a *threat* can be identified with a hostile agent who either accidentally or intentionally gains an unauthorized access to the protected database resources [46]. Threats to the database security may be physical or logical. *Physical threats* range from a forced disclosure of passwords, a theft, a destruction of physical storage devices to a power failure. The protection against these threats comprises a variety of different methods and techniques. The restriction of physical access to database storage facilities and the database backup and recovery are common protection methods.

*Logical threats* involve unauthorized logical (i.e. via software) access to information. They can result with: disclosure of confidential information, illegal modification of data, or destruction of database resource. The logical threats can be classified as follows [46, 67]:

- *Disclosure of information* which includes direct or indirect access (by inference) to protected information.
- *Illegal modification of data* which is caused by improper possibly accidental data handling or intentional modifications by an illegal user (these threats relate to all attacks to data integrity).
- *Denial of Service* which results from the monopolization of system resources in such a way that other users cannot access them. This involves all attacks on availability.

These threats can occur intentionally or accidentally. *Accidental threats* include: *natural disasters* such as earthquakes or water/fire damage, *errors or bugs in hardware or software*, or *human errors*. *Intentional threats* can be caused by *authorized users* who abuse their privileges and authorities, or by *hostile users* who execute some hidden codes within some legitimate functions in order to violate the security of the system. *Trojan Horses, viruses,* and *trapdoors* are examples of hidden codes. A Trojan Horse is hidden code that under a legitimate function collects information which is later used to break the security of the system. A virus is a code that is able to copy itself and damage permanently the environment where it resides. Trapdoors are code segments within programs that allow their owners to skip the protection mechanisms and to access data or system sources beyond their rights.

## 3.3 Database Security Requirements

To eliminate logical threats, it is necessary to define a proper security policy. The database is considered to be secure if a protection mechanism correctly enforces the security policy. Note that there can be many security mechanisms which realize the same security policy.

A security policy must have appropriate security features. These features should be implemented by a security mechanism. The following list gives a typical collection of security policies for databases [46, 71, 166].

- **Access control policy** ensures that all direct accesses to the system objects proceed according to the privileges and the access rules. Access control policies can be either *mandatory* or *discretionary*. A *discretionary access control policy* specifies users' privileges to different system resources, including their ability to transfer their privileges to other users. A *mandatory access control policy* restricts the access of users to system objects on the basis of their *security clearance* and *security classification* assigned to objects. Mandatory access control policies are of concern in multi-level databases.

- **Inference policy** specifies how to protect classified information from disclosure when the information is released indirectly in the form of statistical data.

- **User identification/authentication policy** indicates the requirements for correct identification of users. The user identification is the basis of every security mechanism. Users are allowed to access data after the identification as *authorized* users.

- **Accountability and audit policy** provides the requirements for the record keeping of all accesses to the database. It is an useful deterrent tool for data physical integrity, as well as it is useful for the analysis of the access profile.

- **Consistency policy** defines the states in which the database is considered valid or correct and includes *operational integrity, semantic integrity,* and *physical integrity of database.* Operational integrity aims to ensure the logical consistency of data in a database during concurrent transactions. Semantic integrity ensures the logical consistency of modified data by checking if data values are in the allowed range. Physical integrity of database aims to ensure that the database is immune to (or reconstructable after) physical threats such as power failures.

Most of the effort of the research in the database security has been spent on the first two issues.

It is crucial to be able to evaluate to what degree the security features have been incorporated into the mechanism. The evaluation of security (trusted) systems (mechanisms) is an integral part of the design process. Here is a list of the existing evaluation criteria:

- *Trusted Computer System Evaluation Criteria - TCSEC* (also known as the *Orange Book*), US Department of Defense [73],
- *Information Technology Security Evaluation Criteria - ITSEC* (also known as the *White Book*), Commission of the European Communities [54], and
- *Canadian Trusted Computer Product Evaluation Criteria - CTCPEC*), Canadian System Security Centre [44].

The evaluation of trusted systems (products) can be characterized by:

1. *Completeness.* This is measured by a list of possible threats against which the system is immune.
2. *Confidence.* The degree of trust that the system is immune against the specified threats.
3. *System flexibility.* The system should be able to implement different variants of security policies.
4. *Ease of use.* The system should not impose unnecessary or cumbersome restrictions.
5. *Tamperproof.* If a security mechanism itself is protected from unauthorized modification, then this mechanism is said to be tamperproof. This is an essential characteristic, since even if a security mechanism was proved to be correct, any later modifications to it could degrade its security.
6. *Low overhead.* The difference in performance of the system with and without its security mechanism should be as small as possible.

7. *Low operating cost.* This includes the cost of special hardware or software, the cost of security audits, salaries of security officers and others who are involved in performing security-related functions.

## 3.4 Access Control

There are two classes of resources in any computer system: *(active) subjects* and *(passive) objects.* The way a subject accesses an object is called the access privilege (or access mode). Access privileges allow subjects to either manipulate objects (read, write, execute, etc.) or modify the access control information (transfer ownership, grant and revoke privileges, etc.).

The correctness of access control heavily relies on the following:

- *the proper user identification, and*
- *the protection of access control mechanism.*

The access control may be based on different policies. The choice of a security policy is important because it influences the flexibility, usability, and performance of the system [82]. While working out a proper security policy, it is necessary to proceed with it according to "a good design guide" which includes the following principles [46, 82, 130]:

**Minimum vs maximum privilege principle**. According to the *minimum privilege principle*, subjects should be given the minimum set of privileges necessary for their activity (also called the least privilege). The opposite of this is the *maximum privilege principle* which is based on the principle of the maximum availability of data in a database. Subjects are given access to the largest range of system resources.

**Open vs closed system principle.** In an open system, all accesses that are not explicitly forbidden are allowed. While in a closed system, all accesses are allowed only if explicitly authorized. A closed system is inherently more secure. When security is an important objective, a closed system should be implemented.

**Centralized vs decentralized administration principle.** The principle addresses the issues who is responsible for the maintenance and management of privileges in the access control model. In centralized administration, a single authority (or group) controls all security aspects of the system, while in a decentralized system different authorities control different portions of the database.

The choice between centralized or decentralized administration has to be made. There are some advantages and disadvantages related to each choice (for detailed discussion of those, the reader is directed to [34]). There are, however, some intermediate choices such as: *delegation*, *owner-based administration*, and *co-operative administration.*

*Delegation* can be used in a centralized database system to avoid bottleneck and support local autonomy in a distributed database system. A central authority delegates their administrative rights over a subset of the database to the local authority. The central authority can nominate and dismiss local authorities.

*Owner-based administration* - the central authority passes all responsibilities and rights to the owners of objects. The central authority assigned a collection of system privileges to every user when is admitted to the database system.

*Co-operative administration* - some privileges are reserved to groups. To exercise the access, a single member of the groups needs to get permission from the rest of the group.

**Granularity principle** - a clear specification of size and structure of the smallest object to which access control can be handled. Five types of granularity can be distinguished [65, 82]:

1. *Name-dependent* (also called content-independent). All accessible objects are identified by their unique names.

2. *Content-dependent* (also called constraint-based). Accessible objects are determined by checking the requested attributes. This granularity can be very fine depending on the selection of the applicable attributes.

3. *Context-dependent.* The access is granted to the object whose structure is defined by a rule which checks not only names and attributes but also their structural relation (context).

4. *History-dependent.* This is a generalization of the context-dependent granularity in which the current access request is checked also against all previous access requests.

5. *Time-dependent.* The collection of accessible objects can vary in time ( for instance, a subject may be authorized to read from the class *EMPLOYEE* only between *8:00am* and *5:00pm*).

**Access privilege (or access mode) principle.** A basic collection of allowable access privileges includes: *read, write, delete, execute,* and *create.* They may be ordered. So if a user (subject) has a privilege of higher order, this implies that (s)he has also all privileges of lower order. Users can be assigned explicitly a collection of suitable privileges for each protected object - this is also called *positive authorizations.* On the other hand, in the *negative authorizations*, users are given a collection of privileges which are explicitly denied.

There are three different types of access control: *discretionary access control (DAC), mandatory access control (MAC), and role based access control (RBAC).* A system may employ DAC, MAC or their combination of both, or RBAC for protection.

A *discretionary access control* specifies privileges of subjects in accessing objects, and the rules whereby subjects can, at their discretion, grant and revoke their privileges to other subjects.

A *mandatory access control* identifies the rules whereby subjects can obtain direct or indirect access to classified data. The rules can also be used for sanitizing and reclassifying data. The MAC applies only to *multi-level databases,* which are databases that contain information of different security classification.

A *role-based access control* enforces the least privilege and the separation of duties. This is especially important in some database applications where subjects (users) can be assigned roles to perform specific tasks (defined by the role). Each role has an associated collection of privileges. This collection is automatically transferred to a subject who plays the role.

## 3.5 Discretionary Access Control Models

Discretionary Access Control (DAC) allows privileges to be granted to other subjects by the object owners. In discretionary security the way in which individual users (subjects) manipulate specific objects is determined explicitly through access rules. They are fundamental to operating systems (as a means of protecting files, memory segments, etc.) as well as database systems. DAC has been studied in the context of the *access matrix model.*

### 3.5.1 Access Matrix Model

The access matrix model was developed by Lampson in 1971 [129] and extended by Graham and Denning [93]. Later, Harrison, Ruzzo, and Ullman [106] developed a more general version of the model. They defined the safety problem and showed that it was undecidable. The access matrix model is defined for three components: *subjects* (active entities such as users, their processes, etc.), *objects* (passive entities such as files, records, classes, instances, views etc.), and the collection of *privileges* (read, write, delete, create, execute, etc.). Note that the class of objects contains all subjects. Having these three components, it is possible to define an access matrix $A$. Rows of $A$ are indexed by all subjects (their names) ($S$) and columns - by names of all objects ($O$). Each entry $A[s, o]$ contains a collection of privileges held by subject $s$ to object $o$. A representation of an access control matrix is shown in Table 3.1.

| | *Objects* | | | | |
|---|---|---|---|---|---|
| *Subjects* | $O_1$ | ... | $O_j$ | ... | $O_m$ |
| $S_1$ | $A[s_1, o_1]$ | | $A[s_1, o_j]$ | | $A[s_1, o_m]$ |
| . | | | | | |
| . | | | | | |
| $S_i$ | $A[s_i, o_1]$ | | $A[s_i, o_j]$ | | $A[s_i, o_m]$ |
| . | | | | | |
| . | | | | | |
| $S_n$ | $A[s_n, o_1]$ | | $A[s_n, o_j]$ | | $A[s_n, o_m]$ |

Table 3.1: Access matrix

The access control matrix can be used for protection in both operating systems and databases. In databases, access control matrix needs to be extended. Every entry $A[s, o]$ of the matrix contains (apart from privileges) a suitable condition which needs to be satisfied by the subject $s$ to access the object $o$. The condition can incorporate different types of access such as content-dependent, context-dependent, etc. ( for full list of access types, see Section 3.4). Fernandez [81], and Conway [56] showed how to generalize the access matrix model by using predicates and other components.

Observe that in general, an access control matrix $A$ is sparse. A direct storage of the matrix $A$ wastes a lot of memory. A simple solution is to store the matrix $A$ as a sequence of either rows (*Capability List*) or columns (*Access Control List*).

For each subject $s$, there is the unique capability list (row of $A$). Each element of the list indicates an object and the collection of privileges the subject has to the object. Note that the list does not contain empty entries. If there is no list entry for a given object, the subject cannot access it. Capability lists (CL) allow the system to identify quickly the collection of all accessible objects for a given subject. The opposite, i.e. finding the collection of all subjects who have access to a given object, is difficult and time consuming. Capability lists are used by the operating systems only [71].

Access control lists (ACL) are associated with their objects (columns of the access control matrix $A$). For a given object, the list consists of all non-empty entries of the column of $A$. This implementation allows for quick identification of subjects who can access an object. On the other hand, the recreation of the capability list for a subject from ACL is difficult. ACL are used in both operating systems and database systems [71].

Note that in both implementations the storage requirements increase with the growth of the number of subjects and objects. In particular, the maintenance of such lists is expensive in terms of time and consumed computing resources.

However, the access matrix provides a flexible model which can be used to analyze its security properties. It is known [46, 147] that the general safety problem is undecidable, i.e., there is no algorithm which can be used to verify the security of the access control matrix model. But it is still possible to restrict the model and design an algorithm which can be used to prove some security properties. Some work has been done to extend the access matrix model to make the safety problem decidable. This includes the *Schematic Protection Model* [9, 178, 179] and the *Typed Access Matrix Model* [181].

## 3.6 DAC Models in OODB Systems

In recent years there has been considerable efforts in the research and development of object-oriented databases. ORION [124], ONTOS [164], $O_2$ [74], Objectivity/DB [159], Iris [222], and VERSANT [218] are examples of such efforts. The driving forces behind these efforts are the advantages offered by O-O approach to database modeling. In particular it is possible to represent the composite (or complex) structure of objects and to simulate the behavior of objects through operations encapsulated within the classes. In order to fully exploit the benefits of the O-O paradigm, it is important to consider how the O-O data model impacts on access control models. To avoid confusion, we will use the term *entity* to refer to a passive item in protection system instead of object, and the term *object* has the meaning usually associated with the O-O environment. Note that the term *object* is usually used in literature for such a passive item. Let us consider the following important issues related to DAC in O-O environment [34, 46, 144, 199, 206].

- *Access privileges (or access modes).* One of attractive features of OODB systems is *encapsulation.* Encapsulation allows data to be stored as values of properties (or instance-variables) that are encapsulated in an object and available only through the methods defined for the object. In order to take advantage of the encapsulation feature, the access control model (or authorization model) should support privileges to execute these methods (instead of traditional privileges such as read, write, and create). Moreover since methods can call other methods, it would be appropriate also to consider methods as subjects.

- *Propagation of privileges (or authorizations).* OODB systems allow that classes are organized into a hierarchy of classes. There are two types of hierarchies which are orthogonal to each other: a *class-composition hierarchy* and a *class-inheritance hierarchy* [126]. The class-composition hierarchy captures the *is-part-of* relationship between a parent class and its component classes. Whereas the class-inheritance hierarchy represents the *is-a* relationship between a superclass and its subclasses. There is a question - how to assign privileges to objects in these hierarchies? In particular we would like to know whether a privilege the subject has to a class, is valid also to all descendants of the class for the class-composition hierarchy (*visibility from above* [130]). Or whether a privilege to a subclass is valid to the property values of the superclasses on the higher levels of the class-inheritance hierarchy (*visibility from below* [130]). Additionally, OODB systems must provide a policy for solving possible authorization conflicts between explicit privileges (privileges assigned directly) and implicit privileges (privileges derived through the hierarchies).

- *Authorization administration.* In most access control models when a user (subject) creates a new element, (s)he becomes its owner. The owner controls the distribution of privileges to it. So, the unit of ownership cannot be a class since classes often represent just templates for users to derive their own instances. Hence, there is a need to allow class instances to be units of ownerships. Being more specific, the access control policy must clearly spell out how the class owner can affect the privileges of other users especially in the context of instance-of, class-inheritance, and class-composition relationships.

- *Granularity of authorization.* It should be possible to control access to single object-instances, entire classes, and properties (instance-variables) [169]. For the *class-based authorization,* a subject can access a class and all its instances. For the *instance-based authorization,* instances of a class are units of authorizations. A subject may have authorization to a subset of the instances of a class. For *instance-variable-based authorization,* access control on instance-variables or properties is allowed. A subject can be limited to access only a part of an object. Moreover, the model of access control should take into account semantic constructions such as composite objects and versions.

- *Content-based authorization.* Authorization may be defined in terms of the ability to invoke a certain method on an object. The access control policy must specify how we can access the object content for checking the condition if a method which can access the required properties cannot be invoked?

Access control models for OODB systems are still being investigated [27]. Although there is an undeniable progress in the area, there are few OODB systems only (Orion [169] and Iris [4]) which provide access control models similar to those provided by the current relational databases.

Dittrich [75] divided OODB systems into three categories: *structurally, behaviorally,* and *fully* OODB systems. *Structurally* OODB systems can handle composite (or complex) object structures (i.e. objects that may result from the aggregation of other objects) using generic privileges. *Behaviorally* OODB systems provide an interface to deal with objects and methods on different levels of the inheritance hierarchy. *Fully* OODB systems combine the properties of structurally and behaviorally OODB systems.

Access control models for these categories of OODB systems were discussed in [4, 29, 30, 32, 78, 80, 97, 130, 169, 170]. The models promise to offer some solutions to the forthcoming issues. However, each of them addresses only some of the issues, therefore many problems remain open.

## 3.6.1 DAC in Structurally OODB Systems

In structurally OODB systems, the privileges are typically generic system-defined operations such as *read, write, delete,* and *read-definition.*

### DAMOKLES Access Control Model

Dittrich, Hartig, and Pfefferle [78] developed a DAC model for the DAMOKLES system [76]. DAMOKLES is a structurally OODB system for CASE and similar applications and its data model is called DODM (design object data model) [77]. Two types of objects are supported by the model: DODM objects, and DODM relationships. The access control allows a user $u$ to grant a privilege to other users if $u$ is the owner of an object. When a user creates a database object, an object or a relationship, (s)he becomes the owner of the created object. The ownership may be transferred (by the owner) to other users or user groups. However, at any time there is exactly one owner for every object. In other words, the ownership is transferable but is not transitive. In the model, each access must be explicitly authorized. The absence of appropriate privileges is interpreted as *access not allowed.* The model handles both composite objects and versioned objects as well.

The model applies the following access privileges:

- *Exist* - this privilege allows a user to read keys of objects.
- *Read* - permits a user to read object properties, objects components and relationships roles.
- *Write* - allows a user to modify objects properties and roles, to insert new components/create new versions, and to remove components/delete versions.
- *Delete* - enables a user to remove objects.

The privileges are partially ordered. If a user holds a privilege of a higher order, then they also hold all privileges of lower order. The assumed order is: *exist* < *read* < *write* < *delete.*

The triple $(s, o, r)$ specifies a single access rule. It means that a subject $s$ is assigned an access privilege $r$ to an object $o$. The subject $s$ is a pair $s = (u, p)$ where $u$ is a user or a group of users and $p$ is a program or program group. The pair $(u, p)$ can be read as "user $u$ while executing $p$". The object $o$ is a protection object (p-object). In DAMOKLES data model (DODM), every object (in the O-O sense) is further broken down into smaller access units called *protection objects (p-objects).* These p-objects are: the descriptive part $D$ (object's properties), the structural part $S$ (the components/composite objects), version part $V$ (the object's versions), and role part $\rho$ consisting of the relationship's roles that is participating objects. Note that the access is explicitly granted to the $D$ part of a p-object. However once granted, the access extends to all object's properties.

There are two types of authorization: *simple* and *complex.* In the simple authorization, a p-object $o$ is $D$, $S$, $V$, or $\rho$ part of a complex object. In the complex authorization, a privilege is given to an entire object including its components which belong to the same owner. Thus if an object actually contains components of various owners, a subject has to get permission from all of them to be able to work with entire object.

### LPGSF Access Control Model

Larrondo-Petrie, Guides, Song, and Fernandez [97, 130] developed an access control model (LPGSF model). The model is based on a set of policies that define authorization inheritance through class hierarchies. Negative authorizations can be used to override implied privileges. Predicates (content-dependent) with positive authorization and instances along the class hierarchy can also be applied. The model allows a decentralized administration of authorizations by users.

The access control model uses the following access entities as elementary protected objects: classes, instances of classes, and their properties (or instance-variables).

An access rule is a tuple $(s, o, r, [c])$. The subject $s$ is a user or a user group. $r$ is a privilege or a set of privileges. They can be positive or negative. The object $o$ is a class, an entire object $O$ or its components, or a set of its properties, i.e.,

$o = \{O.p_1, O.p2, \ldots\}$. Properties $p_i$ $(i = 1, 2, \ldots)$ must be either defined for the object $O$ or inherited by it. $c$ is a condition that must be satisfied for the object $o$ so that the subject $s$ can use the privilege $r$. The model enforces the following policies:

**Inheritance policy.** *A user who has access to a class, is allowed to have the same type of access to subclasses and to the properties inherited from the class satisfying the inherited conditions.*

**Visibility from below policy.** *The access to a class implies the access to the properties defined in the class as well as to properties inherited from the higher classes (this is applicable to the class-relevant values of these attributes only). If there is more than one ancestor (in the case of multiple inheritance), the access to the union of the inherited properties is granted. Note that properties defined in subclasses are not accessible by any of their superclasses.*

**Visibility from above policy.** *The access to an object of a composite object implies the access to all components of the object.*

**Overriding policy.** *An explicit positive authorization is a triple* $(s, o, +r)$. *An explicit negative authorization is a triple* $(s, o, -r)$. *An implicit positive authorization is a triple* $[s, o, +r]$. *An implicit negative authorization is a triple* $[s, o, -r]$ *where* $s$ *is a subject,* $o$ *is an object, and* $r$ *is an access privilege. We have the following order to solve possible authorization conflicts:* $(s, o, -r) > (s, o, +r) > [s, o, -r] > [s, o, +r]$.

### ORION Access Control Model

In 1991, Rabitti, Bertino, Kim, and Woelk [169] developed a DAC model which has been implemented for ORION [124]. Their model handles the following types of authorization: *implicit*, *explicit*, *positive*, *negative*, *strong*, and *weak*. Implicit authorizations are deduced from explicit authorizations stored in the system. Positive authorizations grant users access privileges to objects whereas negative authorizations (or lack of authorizations) deny access privileges to objects for users. A *strong authorization* cannot be overridden whereas weak authorizations can be overridden by a strong or other weak authorizations. Triples $(s, o, +r)$ and $(s, o, -r)$ denote a *strong positive authorization* and a *strong negative authorization*, respectively. Triples $[s, o, +r]$ and $[s, o, -r]$ are a *weak positive authorization* and a *weak negative authorization*, respectively. As usual, $s$ (subject) is a user or a group of users, $o$ is an entity protected by the system, and $r$ is an access privilege.

In the model, two access bases are defined. The *strong access base SAB* contains all explicit strong authorizations including both positive and negative. The *weak access base WAB* contains all explicit weak authorizations including both positive and negative.

A privilege $r$ is granted to an object $o$ for a subject $s$ ( or the access request $(s, o, r)$ is authorized) if the return value of the following function is true.

```
Function f(s, o, r)

  if there exists an explicit or implicit $(s, o, +r)$ in SAB
         then return True
         else if there exists an explicit or implicit $(s, o, -r)$ in SAB
            then return False
           else if there exists an explicit or implicit $[s, o, +r]$ in WAB
               then return True
               else if there exists an explicit or implicit
                     $[s, o, -r]$ in WAB
                  then return False

end
```

Note that the function works correctly if there exist at least a weak authorization for each possible privilege.

The cornerstone of the discussed access control model is the implicit authorization which can be propagated along each of the three dimensions (parameters) of access rules namely: subjects, entities, and access privileges. To reduce the number of subjects involved in explicit authorizations, users and groups of users are assigned roles. Roles can be arranged into a *role hierarchy* (see Figure 3.1).

The following two rules govern the propagation of authorization for roles.

**Implication Rule 1** *Explicit positive authorizations for roles result in implicit positive authorizations for all higher-level (upper) roles.* □

**Implication Rule 2** *Explicit negative authorizations for roles result in implicit negative authorizations for all lower-level roles.* □

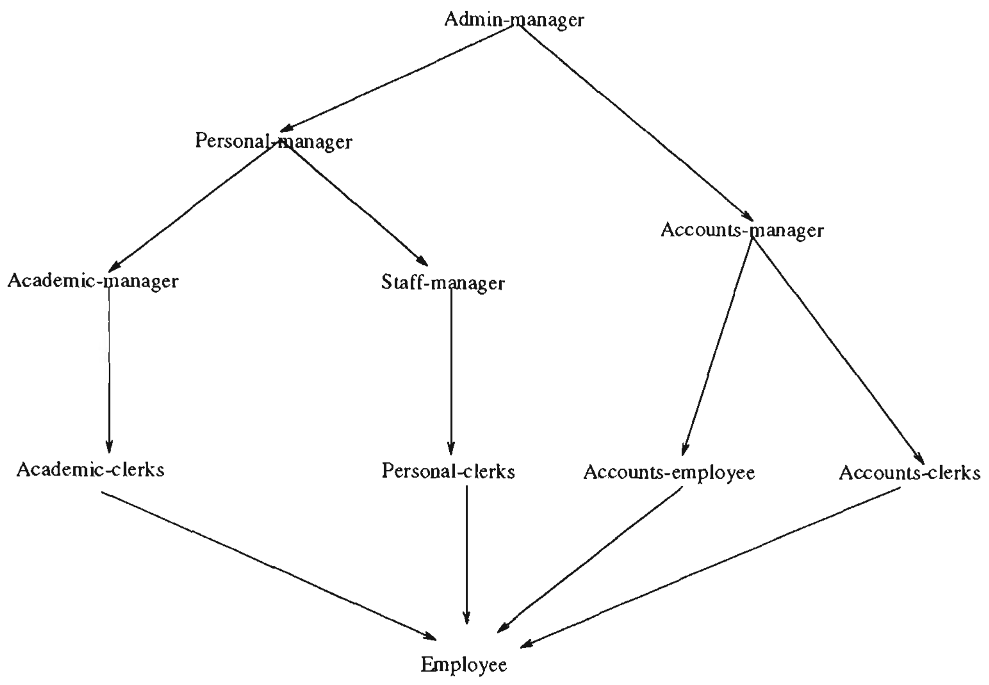


Figure 3.1: A sample of role hierarchy.

Access privileges also create a privilege hierarchy. An example is given in Figure 3.2. The next two rules specify how to generate authorizations along the privilege hierarchy.

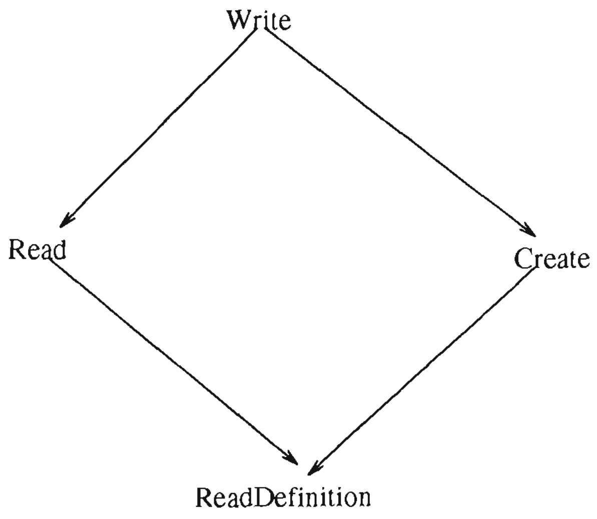


Figure 3.2: A sample privilege hierarchy.

**Implication Rule 3** *If a positive authorization is given for an access privilege in the privilege hierarchy, then this implies the right of all access privileges below it.* □

**Implication Rule 4** *If a negative authorization is given for an access privilege* *in the privilege hierarchy, then this also implies the negative authorization of all access privileges above it.* □

Entities are also organized as the *entity hierarchy.* The entity hierarchy is the instantiation of the *entity schema.* Figure 3.3 shows an example of the entity hierarchy and the entity schema.

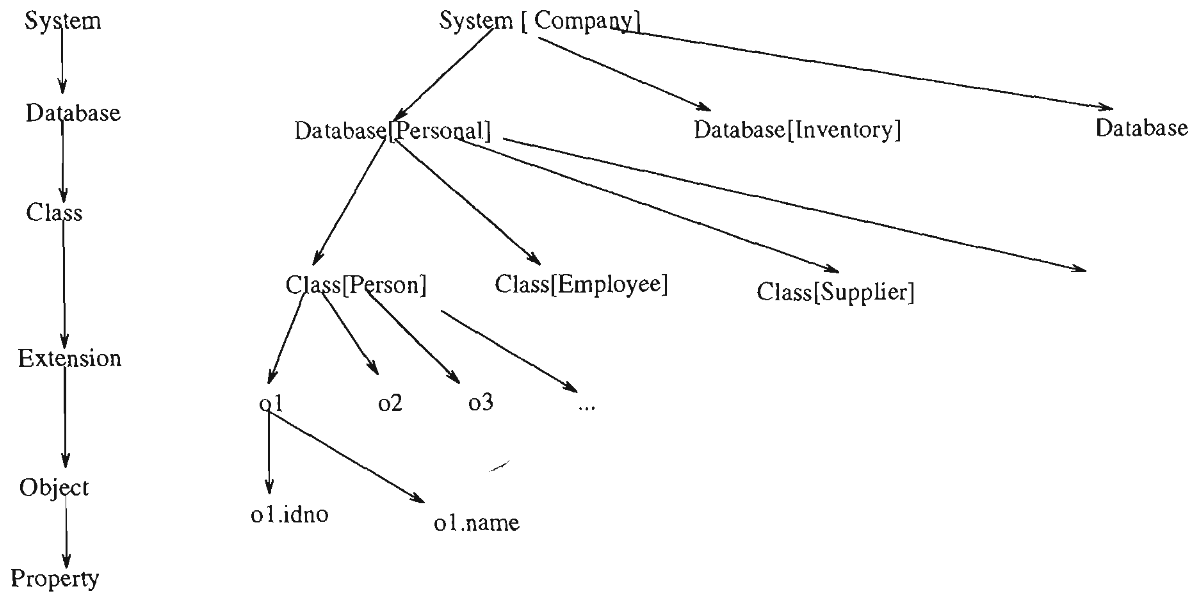


Figure 3.3: A sample Entity Schema and Entity Hierarchy.

The deduction of authorization in the entity hierarchy depends on not only the entity but on the mode of privilege involved, as well. For example, the *read* and the *write* access privileges for certain objects require the corresponding class definitions to be readable. This means that the direction of the implication is *upward* in the entity hierarchy. On the other hand, the *read* permission on the extension of a class implies the *read* permission on all the instances of that type. In this case, the direction of the implication is *downward.* Some privileges do not have any implications, such as creation (definition) of a new class. So all access privileges can be split into three classes: (1) *OpUp* contains those access privileges having upward implications, (2) *OpDown* includes those access privileges having downward implications, and (3) *OpNil* contains those privileges having no implications.

**Implication Rule 5** *Let $r \in OpUP$. The explicit right $(s,o,r)$ yields the implicit authorization $(s,o',r)$ for any entity $o'$ above $o$ in the entity hierarchy.* □

**Implication Rule 6** *Let $r \in OpUP$. The explicit authorization $(s,o,-r)$ yields the implicit right $(s,o',-r)$ for any entity $o'$ below $o$ in the entity hierarchy.* □

**Implication Rule 7** *Let $r \in OpDown$. The explicit authorization $(s, o, r)$ yields the implicit right $(s, o', r)$ for any entity $o'$ below $o$ in the entity hierarchy.* □

**Implication Rule 8** *Let $r \in OpDown$. The explicit right $(s, o, -r)$ yields the implicit authorization $(s, o', -r)$ for any entity $o'$ above $o$ in the entity hierarchy.* □

**Implication Rule 9** *Let $r \in OpNil$. The explicit authorization $(s, o, r)$ generates no implicit ones.* □

Note that the *read* right on methods forces the *execute* privilege. Any access which is performed during the execution of the method must be authorized independently. For example, if during the execution of a method invoked by a subject $s$, an attempt is made to update a particular property value $o$ of an instance, the triple $(s, o, write)$ needs to be verified. Other hierarchical structures (which are orthogonal to the entity hierarchy) are inheritance, composite, and version hierarchies.

There are two approaches to give privileges on instances of a subclass. In the first one, the creator of a class should have no implicit right on the instances of a subclass derived from the class. This approach encourages the reuse of existing classes without diminishing privacy. However if a query has the access scope which is a class and all its subclasses, it will only be evaluated for those classes for which the user has the read privilege. In the second approach, the creator of a class should have implicit rights on instances of subclasses. This means that a query (whose scope of access is a class and a class sub-hierarchy is rooted at the class) will be evaluated against the class and its subclasses. The first approach is used as the default while the second one is an option.

**Implication Rule 10** *A read (or write) privilege on a class of a inheritance hierarchy implies read (or write) rights on all classes in the inheritance hierarchy.* □

The following rule enforces consistency between the SAB and WAB for positive and negative authorization in multiple inheritance hierarchy.

**Implication Rule 11** *The ordering between the access privilege subclass-generator and other access privileges is: write>subclass-generator>read-definition. So a user with the privilege write on a class receives implicitly the subclass-generator* *right, i.e., the user can derive subclasses from the class. A user with the privilege subclass-generator on the class C implicitly receives the read-definition right, i.e., the user can read the class definition of C.* □

**Implication Rule 12** *Authorization on a composite class C implies the same right on all instances of C and on all objects that are components of the instances of C. Similarly, privilege on a composite object implies the same privilege on each component of the composite object.*□

To solve conflicts caused by the combination of strong/weak, explicit/implicit, and positive/negative authorizations through the composite hierarchy, the following order of authorizations is assumed:

$(s, o, +r) > (s, o, -r) > (s, o, \overline{+r}) > (s, o, \overline{-r}) > [s, o, +r] > [s, o, -r] > [s, o, \overline{+r}]$ $>[s, o, \overline{-r}]$, where $\bar{r}$ indicates an implicit privilege.

There are suggestions that the composite object should be considered as a unit of authorization, i.e., giving privilege to a composite object implies the same privilege to all components of the composite object. Two types of authorization: *partial and total* are defined. The next two rules specify how to enforce *partial and total* authorizations.

**Implication Rule 13** *If a user has total write (read) privileges on a component of a composite object, then (s)he has the same rights on all descendants, and partial write (read) rights on all top-level entities in the entity hierarchy.* □

**Implication Rule 14** *If a user has partial write (read) privileges on a component of a composite object, then (s)he has authorization on the object only, not its descendants, and the same rights on all top-level entities in the entity hierarchy.* □

There are two ways to bind an object with a versioned object: *static* and *dynamic*. In static binding, the first object directly references the second object. In dynamic binding, the first object references a *generic instance* of the second object. A generic instance maintains the history of the object versions derived so far. When a generic instance receives a message, the message is forwarded to one of the versions, which has been designed as the default version. Furthermore, on the basis of the types of operation that may be allowed on versions, two types of versions: *transient* and *working* can be distinguished. A transient version may be modified or deleted by the user who has created it whereas a working version may be deleted but not updated. A new version cannot be derived from a transient version. A transient version must first be *promoted* to a working version before a new version may be derived from it.

**Implication Rule 15** *An authorization on a set-of-generic-instances implies the same right on all generic instances of the class. The write privilege on a generic instances allows the user to modify the generic instance itself . It also implies authorization on the object versions described by the generic instance. The read right on a generic instance means that the user has the same right on all versions. The write privilege on the set-of-versions implies the same right on the versions described by the generic instance and also gives the right to create a new version from a working version of the instance.* □

The model does not address content-dependent and method-based authorizations. Bertino and Weigand [30] extended the model so it could handle content-dependent access rules. They introduced two different modes for authorization administration, *centralized* and *decentralized* administrations. In the decentralized administration, every user creating an instance of the class is considered to be the owner of the instance and, therefore, can grant and revoke other users' privileges to the instance. For the centralized administration, all instances of a class are considered to have the same owner who is called the *class administrator.* Instances of the class created by any user belong to the class administrator (see [30] for detailed discussion). The main problem of the model is how to efficiently evaluate conditions associated with authorizations. In particular, the processing of the conditions could require two references to the object (one to evaluate the conditions and the second to filter data that satisfy the user query).

### 3.6.2 DAC in Behaviorally OODB Systems

The models presented so far take into consideration many of the characteristics of O-O data models such as inheritance hierarchy, versions, and composite objects. However, they do not apply encapsulation of the O-O model. In a behaviorally OODB system, all interactions are conducted through messages that are parts of object interfaces. A subject invokes an initial method $m_1$ which in turn calls $m_2$, and so on. In an access control model for behaviorally OODB systems, we are concerned with how the subject gets permission for methods in the sequence of calls.

### Faatz-Spooner Access Control Model

Faatz and Spooner [80] describe a DAC model for O-O engineering databases. In the model, objects consist of the structural and operational parts. The structural part includes instance-variables that hold the data associated with the object. The operational part contains a set of methods or procedures that can be performed on the data in the structural part of the object. Only methods defined for an object can be used to manipulate the contents of the object instance-variables. These variables are not visible outside the object. The access control model applies object interfaces (object views) in order to restrict the number of messages an object accepts from other objects. Users are allowed to interact with the objects only by calling methods defined in their interface.

Although the above approach requires no extensions to the O-O paradigm, it provides only a partial solution to the access control. A complete solution has to handle access control to object classes with hierarchical structure. It also has to work properly in the context of the inheritance.

### Iris Access Control Model

In 1992, Ahad, Davis, Gower, Lyngback, Marynowski, and Onuegbe [4] developed a DAC model which has been implemented for Iris [222]. In the Iris data model, both instance-variables and methods are represented as functions. Instance-variables are defined as *stored functions*, and methods - as *derived functions.* Objects are encapsulated by a set of functions that users can call. The access control model is based on a concept of the function call control and the evaluation of calls.

The access entities are: *stored functions, derived functions, generic functions, guard functions*, and *proxy functions. Stored* and *derived* functions are instance-variables and methods, respectively. A *generic function* is the specification of a function which may have a set of associated *specific functions* that are defined for different types. When a generic function is called, a specific function based on the type of the argument object in the call is selected for execution. If a function has an associated guard function, the function can be executed only if the guard function returns *true*. *Proxy functions* provide different implementations of specific functions for different users. A function may have several associated proxy functions. When a user calls a function, the appropriate proxy is executed in place of the original function.

The access control is implemented using functions that are allowed to be called by a subject (user). An access rule is a triple $(u, f, t)$ where $u$ is the user name, $f$ is a function or a set of functions, and $t$ is the type of the argument to the function. Note that the access of $u$ is limited by the set of functions $f$ the user is allowed to evaluate. Functions $f$ can be either stored or derived. Access to derived functions can be *static* or *dynamic*. If a user $u$ has a dynamic access to the derived function $f$, $u$ must have a permission to call all the underlying functions. For a static access, the user $u$ dose not need to have call privileges[1] to the underlying functions. While creating a derived function, the user must specify whether the access to the underlying functions should be static or dynamic. Note that the creator of the derived function must have call privileges to all the underlying functions. The access control model supports time-dependent and content-dependent authorizations using guard and proxy functions, respectively. Derived functions can also be used to support content-dependent authorizations including conditions.

The concept of ownership is supported as well. A user who creates a function is its *owner*. The owner of a function automatically has a *call privilege* to the function. If the dynamic access is specified while the object is created, the owner can freely grant call privileges for other users. By contrast, for static access, the owner cannot grant a call privilege unless he has grant privilege to all the underlying functions. Access to a function can be granted and revoked to/from other users by users who have grant privilege over that function.

The model supports also the database administrator (DBA) concept. The system may have a DBA or a group DBA who has the privileges implied by the owners of functions. Moreover (s)he can grant call privilege to functions with static access to selected users.

[1]A user who has a call privilege on a function is authorized to evaluate the function.

## Bertino Access Control Model

An interesting example of another access control model is given by Bertino in [32]. The model uses methods such as a tool to control the access to objects. This is why it is also called the *Data-hiding* model. There are two types of methods: *private* and *public*. Private methods can be invoked by other methods only when they present in the list associated with the method. The list is called *invocation scope* of the method. On the other hand, a *public* method can be called by all users and methods.

The access control can have two levels: external or internal. For the external access control, a triple $(s, o, m)$ indicates that a user $s$ can call the method $m$ on the object $o$. The internal access control is content-dependent and is implemented as a part of the method. Users can have privileges to public methods only. For instance, if a user $u$ invokes a method $m_1$ on $o$ which in turn calls $m_2$, then $m_1$ has to be public and $u$ has to have execution right for $m_1$. If $m_2$ is public, the entry $(u, o, m_2)$ must exist. If $m_2$ is private, $m_1$ must belong to the invocation scope of $m_2$.

A user $u$ can grant the execution right on an object if $u$ is the owner of the object. An object may have several owners. Only the creator may grant/revoke ownership authorizations to/from other users. However, the creator may grant the creator privilege to a user $u$. When the creator does so, $u$ becomes the only creator of the object. The old creator looses all privileges on the object. Sometimes it is useful to enable a user to execute a method $m$ without giving him/her the execute rights on all public methods that are invoked by $m$. The model introduces the notion of *protection mode*. If the user $u$ grants the user $u'$ the right to execute method $m$ in the protection mode, then all invocations of public methods made by $m$ are checked for authorizations against $u$ (instead of $u'$) when $u'$ executes $m$.

The reader is directed to [32], for formal definition of the model and its detailed discussion. The main problem with the model is the lack of flexibility as content-dependent authorizations are parts of method implementations. Changes in authorizations would require a change in the specification of the methods. As pointed out in [206], the impact of inheritance hierarchy on the model needs more investigation.

## 3.7 Mandatory Access Control Models

Some databases contain sensitive or classified data. A record (tuple) may be composed with elements of different security level (or classification). The security of entity can be classified on $n$ levels (in practice $n = 4$) and can be further subdivided into compartments by category. For example, multi-level databases store data with different security classification (or security classes). The security class is a pair $(L, C)$ where $L$ represents a *security level* and $C$ denotes a *category.* The security level can be unclassified, confidential, secret, and top secret. Note that they are partially ordered by the the relation "$\geq$". The category can be the name of the application that the entity is associated with. In general, they have no ordering, but may be further subdivided. There are two hierarchies for both security classification of stored data and users of the database. Users are classified according to their clearance.

Unlike the discretionary access control, the mandatory access control (MAC) makes sure that the flow of information complies with a well-defined security policy. MAC enforces that users with their clearance can only access entities on "proper" security levels. The mandatory access control is also called the multi-level access control.

Bell and LaPadula in [26] introduced their MAC model. This is an extension of the access matrix model. The Bell-LaPadula model is based on two properties: the *simple security property* and the **-property.* According to the simple security property, a subject (or an active entity) is allowed to read information from an object (or a passive entity) if the clearance level of the subject dominates the security level of the entity. The *-property requires that a subject has write access to an object if the subject clearance level is dominated by the security level of the entity. Informally, a subject can *read-down* (simple security property) and can *write-up* (*-property). For example, consider two security classes $X = (L1, C1)$ and $Y = (L2, C2)$. $X$ *dominates* $Y$ (denoted by $X \geq Y$) if and only if $L1 \geq L2$ and $C1$ is a compartment of $C2$ or is equal to $C2$ ($C1 \sqsubseteq C2$). $X$ *strictly dominates* $Y$ (denoted by $X > Y$) if and only if $L1 > L2$ and $C1 \sqsubseteq C2$. The set of possible access privileges (or access modes) in the model is determined by the combinations of these properties. The privileges are: neither observe nor alter, observe only (READ-ONLY), alter only (APPEND), observe and alter (WRITE), execute a program (EXECUTE).

A generalization of the Bell-LaPadula MAC model was suggested by Denning [65, chapter 5 ] and called the information flow system. It is based on a lattice of security levels. Information in an entity is allowed to flow (either directly or indirectly) only to entities with higher security levels.

The application of the Bell-LaPadula model to the protection of database systems introduces new security requirements such as *entity integrity, referential integrity,* and *polyinstantiation integrity. Entity integrity* requires that no tuple can have null values for any primary key attribute. *Referential integrity* insists that no tuple in a relation can refer to a nonexistent tuple. *Polyinstantiation* means that one record can appear (be instantiated) many times, with different security levels. In order to deal with these new security requirements, extensions of the Bell-LaPdula model have been proposed for multi-level security databases (MLS/DBS). The relational data model has dominated much of the work on MLS/DBS [59, 69, 79, 99, 109, 112, 116, 117, 118, 141, 184, 188, 200, see for example]. MLS/DBS have been developed not only as prototypes but also as products [142, 208]. Security issues have also been investigated in other systems such as OODB systems [33, 38, 39, 114, 123, 149, 150, 210], entity relationship systems [87] and knowledge based systems [212] among others. A detailed report on the recent development in database security is given in [211].

## 3.8 MAC Models in RDBS

A multi-level relational database system (ML/RDBS) supports data with different security levels (or classifications) and users with different security clearances. The granularity of a multi-level system is the smallest unit of data which has its own security level. Sometimes, data may be classified at the *attribute level* (all the data associated with a particular attribute has the same security level); at the *tuple level* (every tuple has a single security level); at the *relation level* (all the data in the relation has the same security level); at the data element level; or combination of these (we have the following order: *relation level* $<$ *tuple level* $<$ *attribute level* $<$ *element level*). A relation with these security levels is called a multi-level relation. A multi-level relation $R$ is represented by a schema $R(A_1, C_1, \ldots, A_n, C_n)$, where each data attribute $A_i$ has a corresponding security level attribute $C_i$. Multi-level security affects the data model because not all data

seen differently by users with different clearances).

Now consider how *entity integrity*, *referential integrity*, and *polyinstantiation* are resolved in ML/RDBS.

For a tuple, all the primary key elements must have the same security level or be at least as low as the security levels of other elements in the tuple. Otherwise, a user with low clearance would see nulls for the primary key which violates *entity integrity*.

In a ML/RDB system, *referential integrity* means that a tuple of a low security level cannot reference a tuple of a high security level because the referenced tuple would appear to be nonexistent to users with a low clearance.

*Polyinstantiation* emerges in the form of: *polyinstantiated relations, polyinstantiated tuples,* and *polyinstantiated elements* [69]. A *polyinstantiated relation* occurs when two subjects with different views of the real world entity try to create a relation of the same name [210]. A *polyinstantiated tuple* happens when a user inserts a tuple that has the same primary key value as the existing but invisible tuples (because they have higher security levels). A *polyinstantiated element* is created if a user writes a new element in a tuple. The entry which corresponds to it is seen as a null before the write operation. The element is not null and it contains data with a higher security level. Thus a second tuple is added to the relation with the same primary key but a different security level [144].

Several mandatory access control models for RDBS have been proposed so far [68, 69, 88, 99, 110, 117, 141, 142, 148, 200]. The main difference among the models is the way they solve the integrity and polyinstantiation problems. Let us consider three such models whose security evaluation classification was aimed to be at Class A1[2] level. They are as follows.

- **SeaView**. The SeaView project has its roots in the Summer Study on Multi-level Data Management Security held by the Committee on Multi-level Data Management Security of the U.S. Air Force Studies Board [55]. The project was a three-year joint work by SRI International, Gemini Computers, and Oracle Corporation sponsored by the U.S. Air Force, Rome Laboratory. The SeaView was developed at SRI by Lunt, Denning, Schell, Shockley, Heckman, and Neumann and provided element-level labeling with Class A1 assurance for mandatory security. SeaView generates virtual

[2]In the evaluation criteria defined by US Department of Defense (DOD) [73], security systems is classified on four hierarchical division (D, C, B, A). Class A1 specifies a system whose security has been formally verified.

multi-level relations from physical *single-level* relations. The physical relations are stored in segments managed by an underlying mandatory security kernel. SeaView uses GEMSOS Trusted Computing Base (TCB) as the kernel [187]. GEMSOS enforces the mandatory security policy using a label-based mechanism. Whenever a subject attempts to retrieve a physical object, a label comparison is performed. SeaView supports element, tuple, and relation polyinstantiation. A multi-level query language called MSQL is used in SeaView to define and manipulate multi-level data. Furthermore, to deal with the polyinstantiation, MSQL provides the following functions: *highest-class*, *highest-tuple*, *most-recent-tuple*, and *most-recent.* The function *highest-class* returns the tuple with the highest security level for a potentially polyinstatiated element. The *highest-tuple* function returns the tuple with the highest tuple level from among a set of polyinstantiated tuples. The function *most-recent-tuple* retrieves the most recently updated or inserted tuple from among a set of polyinstantiated tuples. The *most-recent* function returns the tuple with the most recently updated or inserted value for a potentially polyinstantiated element. For a detailed discussion of the model, the reader is directed to [141, 142][46, Section 10 of Chapter 2].

- **Lock Data Views (LDV).** The LDV is a ML/RDB system that was developed by a group at Secure Computing Technology Corporation (SCTC). It is designed to run on SCTC's LOgical Coprocessor Kernel (LOCK) Trusted Computing Base (TCB) [37]. LDV's security policy builds on the security policy of LOCK. Its design is based on three assured pipelines for the query, update, and meta-data management operations. It allows an application to specify classification constraints for how incoming and outgoing data are to be labeled. LDV supports the tuple polyinstantiation. Tuples are polyinstantiated based on their maintenance levels. The maintenance level of a tuple is a security level at which the tuple was inserted into the database. However because of classification constraints, some of data elements that make up a tuple could be stored at security level above the tuple security level. TCB enforces the constraint that the level at which data is stored is the lowest security level at which the data can be retrieved. See [200, 201] for detailed discussion.

- **ASD-Views.** The ASD-Views is a prototype developed at TRW Defense System Group. The main aim for the ASD-Views project was to achieve a suitably-sized Trusted Computing Base (TCB) that would meet the criteria for evaluation of class B2 and above. The main feature of the project is that the ASD allows an interconnection between the mandatory and discretionary access control. Moreover, only one copy of the shared data is stored within the system. This copy is accessible to users at different security levels. The ASD allows to create a subset of rows (tuples) and columns (attributes) from a singular underlying base relation. Joins, aggregate functions and arithmetic expressions are excluded, as they do not support polyinstantiations [88, 110].

## 3.9 MAC Models in OODB Systems

A number of security models for OODB systems have been proposed ([33, 39, 114, 121, 122, 123, 143, 144, 149, 150, 160, 210, 215, as examples]). Nonetheless, the design of an O-O multi-level security model is still an active research topic and there is no dominant design. The main difference among various models is the way how they assign security levels to data stored in objects.

Some proposals consider *single-level objects.* That means that for every object, a unique security level is assigned and this level applies to all components of the object (properties and methods) [33, 114, 149, 210]. This approach is attractive for its simplicity and compatibility with the security kernel. Its most important advantage is that the security kernel is small enough so that it can be verified. Moreover there is no need to handle the multi-level update problem [210].

However in the real world, there are situations when it is necessary to classify instance-variables of an object at different security levels. That is, the security model has to support *multi-level objects.* There have also been proposals that introduce a finer grain of classification by assigning a security level to each instance variable of an object [122, 140, 160]. Unfortunately, these proposals require a trusted enforcement mechanism on the object layer and a complex security kernel.

In order to maintain the security kernel compatibility of the single-level object and to overcome the difficulties of the handling of multi-level objects, some researchers proposed to design a schema which manages various security constraints (*simple, content,* and *context* constraints) [114, 149, 210]. For example, if we want the *GAP* instance-variable of the class *STUDENT*[3] to be secret, we need to create a class *STUDENT_GAP* with security level SECRET to be a subclass of the class *STUDENT* (see Figure 3.4). The main drawback of this

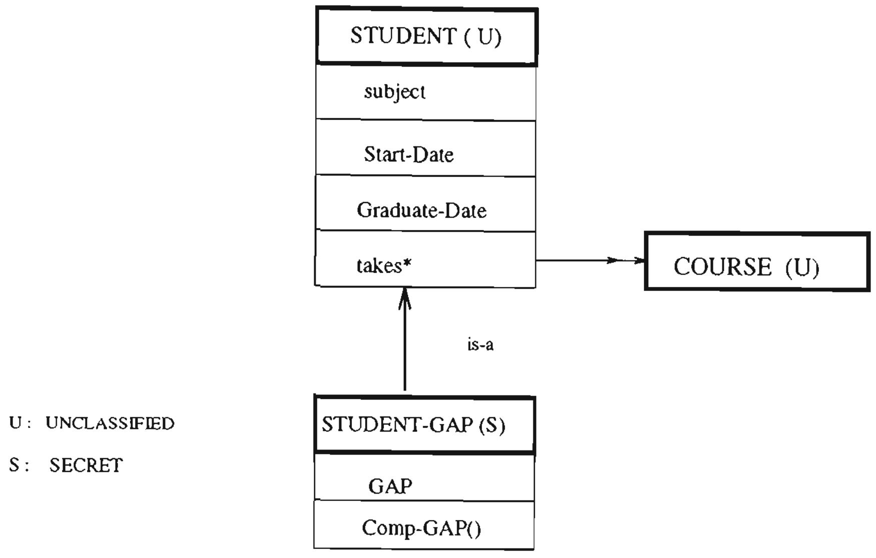


Figure 3.4: An example of representation of simple constraint.

approach is that it requires information to be replicated at different levels. Thus the problem of data consistency starts to play a crucial role.

An alternative approach for modeling multi-level entities through single-level objects were proposed independently by Bertino and Jajodia in [33], and Boulahia-Cuppens, Cuppens, Gabillon, and Yazdanian in [38]. Bertino's approach is based on the use of *composite objects.* Instead of replicating low security level data in higher security level objects, a reference to the object containing the low level data is inserted in the higher level object. The class *STUDENT_GAP* has a composite property say *Student_Specification* whose domain is the class *STUDENT* (see Figure 3.5). Note that in this approach properties of a multi-level entity are stored in different objects. Thus if a user wants to see the entity, more objects must be accessed.

Boulahia-Cuppens and his group used the decomposition of a multi-level entity into single-level entities. Data on each level are stored in a single-level database. Dynamic links are also created between the objects of these single-level databases for properties with low security levels. For instance a multi-level object *O*, which

[3]The specification of STUDENT is shown in Figure 2.3 in Section 2.2.7 of Chapter 2

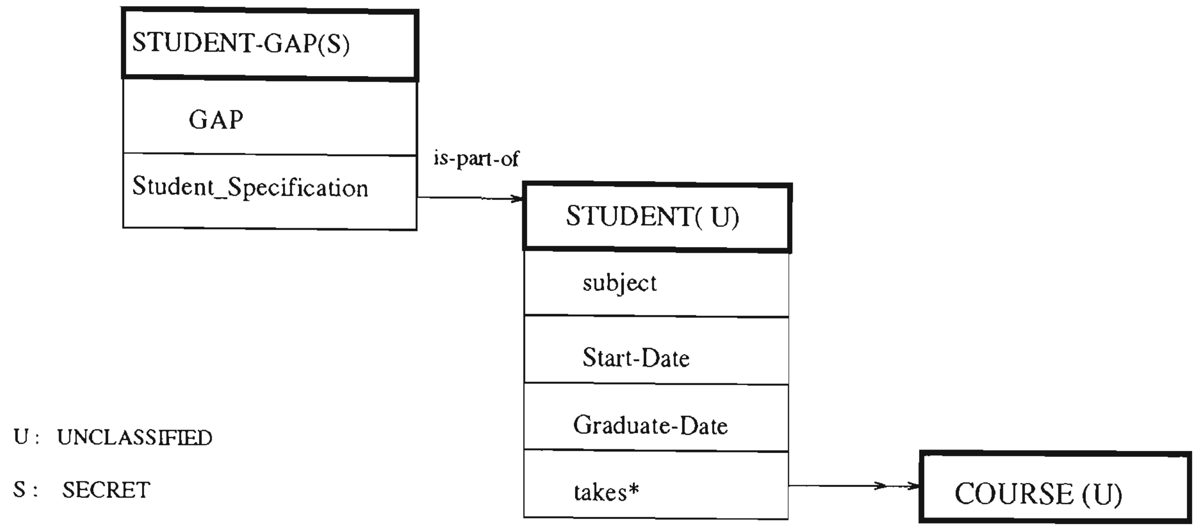


Figure 3.5: An example of representation of a multi-level entity through composite objects.

is an instance of a multi-level class *STUDENT*, will be decomposed to *U-O, C-O,* and *S-O* levels. Each of them is actually a single-level object corresponding to unclassified, confidential, and secret levels which are physically stored in single-level databases. In the *S-O* level there are pointers to the confidential properties stored at the *C-O* level and consequently in the *C-O* level, there are pointers to unclassified properties. This means that if an classified user updates the unclassified properties of the *U-O* level, this update will be automatically propagated to the instances at both *C-O* and *S-O* levels. Note that the values of the properties which point to low-level objects can be updated by the users cleared to access them. If this happens, the pointer to the low-level database is broken, and the value of the object at the low-level database is considered to be a cover story.

By employing a view model mechanism, we also propose a new design approach to model multi-level entities through single-level objects (see Chapter 6). The central idea behind our approach is to provide the user with a *multi-level view* derived from a single-level secure OODB system. Hence the database operations performed on the multi-level views are decomposed into a set of operations on single-level objects. The operations can be implemented with any conventional mandatory security kernel. The detailed discussion is presented in Chapter 6.

### 3.9.1 Security Models

Some of the proposed security models for OODB systems are discussed briefly. For detailed discussion, the reader is directed to the cited papers.

## Jajodia-Kogan Security Model

Jajodia and Kogan [114] proposed a security model for OODB systems that controls access by using the encapsulation characteristic of O-O systems. In O-O systems, communication between two objects can be done via the exchange of messages only. The model controls the information flow by filtering the messages transmitted between objects. That is why it is also called the *message filter* model. The model is defined in terms of *subjects* and *entities. Entities* are objects which may have dual roles of either entity or subject. An active object can act as a subject by sending messages. A security level is assigned to each object at the creation time, and is fixed for the life-time of an object. In the model, all objects are single-level ones. Every message exchanged between objects is done via the message filter. The message filter decides how to handle the message based on the security levels of the sender and receiver, and security level of information encountered in a chain of method executions. The list of possible actions that the message filter can undertake, includes: leaving the message unaltered, blocking the message, and enforcing restriction on the execution of the method invoked by the message.

## SORION Security Model

SORION is a security model proposed by Thuraisingham [210] to incorporate a secure access control into the ORION model [16, 124]. The SORION model is defined in terms of *subjects, entities,* and *access privileges.* A *subject* is any user of the system. Every user is assigned a security level (clearance). *Entities* in the system are *classes, objects, properties*, and *methods.* Object may be *primitive* objects or *composite* objects. The model allows the *read, write,* and *execute* access privileges. The access to an entity is controlled by checking security level of the subject (who wants to access the entity) and the security level of the entity which is computed according to the security rules of the system. For full detail and the list of security rules, the reader is referred to [210]. The model requires all objects to be single-level ones. However, as we have already observed, real-world entities are often multi-level. The representation of multi-level entities through single-level objects are then proposed to design the schema which handles various security constraints (*simple, content,* and *context* constraints).

### Millen-Lunt Security Model

Another security model that enforces a mandatory security policy in OODB systems is that proposed by Millen and Lunt [149]. Their security model is defined in terms of *subjects, entities,* and *access privileges.* A *subject* is an active entity that executes methods upon reception of messages and can also send messages. *Entities* are *classes,* and *objects. Access privileges* are *create, delete, addmessage, getvar,* and *setvar.* The model requires all objects to be single-level ones. Each request from a subject to execute a method or to write/read instance-variables is allowed only if it satisfies the security properties. A list of security properties that must be satisfied by the system is provided (see [149]).

### SODA Security Model

SODA, proposed by Keefe, Tsai, and Thurainsingham [121, 123], is a security model for OODB systems with multi-level entities. The model is based on the Smalltalk model [91]. The SODA model is defined in terms of *subjects*, and *entities.* A *subject* is any user of the system. *Entities* are objects or properties (instance-variables). Objects may be entities or subjects. Objects or properties (instance-variables) are assigned ranges of security levels. Subjects are assigned clearance levels. Every message that travels through the system carries with it a current security level and a clearance level of the subject. The current security level is the least upper bound of all security levels of information the message has read or has access to, and is adjusted whenever an object or property with higher security is accessed. Classification rules (based on the current security level and clearance level of a message and on an object or property security level) determine whether a method should be allowed to access an object or property.

## 3.9.2 Security Rule Requirements for a Secure OODB system

Olivier, Sebastiaan and Von Solms [160, 161] gave a taxonomy for secure OODB systems. They specified security issues and properties that are relevant to the modeling and implementation of a secure OODB system. The taxonomy identifies eight major design parameters every designer of a multi-level secure OODB system must consider. These eight parameters are grouped into three categories:

*labeling semantics, structural labeling,* and *dynamic labeling.* The first group (which consists of two first design parameters) answers the two following questions. *What is a protected entity? How is an entity protected?* The *structural labeling* talks about protected entities, the way security labels are assigned to them, and specifies the restrictions imposed on security labels by the relationship among the entities. The last group of parameters ensures that the secrecy is not compromised by activities in an OODB system. For detailed discussion, the reader is directed to [160, 161]. In the next section, we use this taxonomy and represent some security classification rules that must be satisfied for a secure OODB system. Note that the term labeling refers to the assignment of a security levels to an entity.

In any secure model, it must first be clarified what is exactly protected. It is possible to protect (control) the access to an entity (called *access protection* model) or to hide the fact of existence of an entity (called *existence protection* model). Note that in an *existence protection* model, since the existence of a high security-level entity is hidden from lower-clearance subjects it is possible that the entity is re-created by the subjects. The system must therefore support polyinstantaition.

### 3.9.3 Mandatory Classification Rules

In general, an OODB system consists of a non-homogeneous collection of entities (objects, methods, instance-variables, classes, class methods, class variables, etc.). These entities are related by relationships such as *encapsulation, instantiation, inheritance, composition, association,* and *data structure membership.*

- *Encapsulation* - the relationship that exists between an object and its facets.
- *Instantiation* - the relationship that exists between a class and its objects. Each object is an instance of a class.
- *Inheritance* - the relationship that exists between a class and its subclasses.
- *Composition* - the relationship between objects that are combined into a large object.
- *Data structure membership* - the relationship that exists between data structure (such as a list) and member of the data structure.

- *Association* - the relationship that exist for an object associated with other objects.

In this section, the influence of the structure of data on the classification of entities is considered. Denote by LUB the least upper bound, GLB the greatest lower bound. The function *Level*(e) displays the security level of an entity $e$. The function *class(e)* returns the class of an entity $e$. $c$ and $o$ denote a class and an object, respectively. Denote by *sup(e)* the set of superclasses of an entity $e$, and $< x, c >$ a facet $x$ of a class $c$ where $x$ can be a property or a method.

## Security Classification Rules

The first classification rule that we consider are imposed by the *encapsulation* of an object. An object encapsulates everything inside it. This implies that an encapsulated facet cannot be accessed by a subject who is not allowed to access the corresponding object.

**Classification Rule 1** *(Encapsulation property) The security level of a facet of an object (class) dominates the security level of the object (class)*

$$(\forall o(\forall x \in o))[Level(< x, o >) \geq Level(o)]$$

*or*

$$(\forall c(\forall x \in c))[Level(< x, c >) \geq Level(c)].\square$$

The second group of classification rules is imposed by *instantiation.*

**Classification Rule 2** *(Instantiation property) The security level of an instance must dominate the security level of its class either in an existence protection model or in an access protection model, i.e.,*

$$(\forall o)[Level(o) \geq Level(class(o))].\square$$

**Classification Rule 3** *(Facet property) The security level of a facet in an instance must dominate the security level of the facet in the class and must also dominate the security level of the instance itself. This is expressed by*

$(\forall o(\forall x \in o))[Level(< x, o >) = LUB(Level(< x, class(o) >), Level(o))]$ *(for existence protection) or*

$(\forall o(\forall x \in o))[Level(< x, o >) \geq LUB(Level(< x, class(o) >), Level(o))]$ *(for access protection).* $\square$

A class may have several subclasses. The following properties regulate the classification of subclasses and inherited facets.

**Classification Rule 4** *(Inheritance property) The security level of any subclass must dominate the security level of its superclass(es)*

$$(\forall c(\forall d \in sup(c)))[Level(c) \geq Level(d)].\square$$

**Classification Rule 5** *(Facet inheritance property) In the case of the access protection, if any class $c$ inherits a facet $x$ from a superclass $d \in sup(c)$, then*

$$Level(< x, c >) \geq LUB(Level(< x, d >), Level(c)).$$

*In the case of the existence protection, one of the following must be held:*

1. *if a class $c$ inherits a facet $< x, c >$, the security level of $< x, c >$ may only be different from its security level in the superclass when*

$$Level(< x, c >) = Level(c)$$

2. *if a facet is defined in the one only superclass of a given class $c$ (or the class $c$ has the one only superclass, or the O-O model allows single inheritance only), that facet may be inherited provided*

$$Level(< x, c >) \geq LUB(Level(c), Level(sup(c)))$$

3. *if a facet $x$ is redefined in $c$, then the security level of $< x, c >$ must be dominated by the security level of every liked-named facet (facet with the same name $x$) in any superclass of $c$ (whenever the security level of the facet dominates the security level of $c$) and*

$$(\forall d \in sup(c))[Level(c) \leq Level(< x, c >) \leq LUB(Level(< x, d >), Level(c))]$$

4. *if $x$ is inherited from a specific superclass $d' \in sup(c)$, then for every superclass $d \in sup(c)$ that has a facet $x$,*

$$LUB(Level(< x, d' >), Level(c)) \leq Level(< x, c >)$$

*and*

$$Level(< x, c >) \leq LUB(Level(< x, d >), Level(c))$$

5. *the security level of an inherited facet $< x, c >$ must be dominated by the security level of all like-named facets in superclasses of c, whenever the security level of the like-named facet dominates the security level of c.* □

The last point of the rule above indicates two strategies in the existence protection that can be used to resolve the conflict caused by multiple inheritance. The strategies are:

1. the like-named facet with the lowest security level is always inherited, or
2. the security level of the class is an upper bound for the security levels of all concerned facets in superclasses.

If an object $o$ is constructed from objects $o_1, o_2, \ldots, o_n$ by using array or set, then the following classification property must be satisfied.

**Classification Rule 6** *(Data structure membership property)*

*If array-like structure (or homogeneous structure) is used to construct o, then all members of the array must have the same security level in the existence protection, i.e.,*

$$Level(o) = Level(o_1) = Level(o_2) = \ldots = Level(o_n).$$

*If o is a set-object, the security level of o is the least upper bound of the security levels of the element objects,*

$$Level(o) \geq LUB(Level(o_1), Level(o_2), \ldots, Level(o_n)).\square$$

**Classification Rule 7** *(Association property) If an association R is defined between objects $o_1$ and $o_2$, then the security level of R must satisfy*

$$Level(R) \geq LUB(Level(o_1, o_2).\square$$

### 3.9.4 Dynamic Labeling

As long as an encapsulated object moves as a unit from one to another location in the system (probably as a parameter of a message), security of information will not be compromised. For example, suppose that SALARY object has *PrintSalary* method. A subject that obtains SALARY from EMPLOYEE object will still not be able to invoke the *PrintSalary* method if it is not authorized even though the subject accessed the SALARY. Encapsulation feature of O-O model, combined with security level labels, provides very natural mechanism of protection.

However if, in the example above, the salary was not encapsulated in a SALARY object, but rather stored as a real number, the value would have no natural protection once it leaves the EMPLOYEE object. A similar problem occurs when methods return values of instance-variables. Therefore, a secure system must impose some restrictions on the flow of authorizations and the flow of information to ensure that the security will not be compromised.

- *Authorization flow restrictions.* Messages act on behalf of a subject and therefore the clearance of a message depends on that of the subject. The system must specify how the clearance (or authorization) of a message is determined, and how the security level of a message is determined.

- *Information flow restrictions.* If some of the classified information contained in a message is stored in variables of the receiving object, it must be ensured that an unauthorized subject cannot now access the information in this object. The system should indicate any flow restrictions and any conventions for reclassification.

For full details of how the clearance of a message is determined, how clearances are combined, and possible strategies to control the flow of information, the reader is referred to [160, 161].

## 3.10 Role-Based Access Control Models

Like military agencies, civilian government agencies and commercial firms are very much concerned with the confidentiality of their information. This includes the protection of personal data, marketing plans, product announcements, formulas, manufacturing and development techniques. Many of these organizations have even greater concern with integrity of information [49]. Integrity is particularly relevant to such applications as funds transfer, clinical medicine, environmental research, air traffic control, avionics, etc.

Each organization has unique security requirements, which are often difficult to enforce using traditional access control policies such as MAC and DAC. In many organizations, the end users do not "own" the system objects. In these organizations, the corporation or agency is actual the "owner" of system objects as well as the programs that process them. The control is often based on the employees' position with in the organization hierarchy rather than their data ownership rights. Access control decisions are often determined by the roles individual users take on as a part of their jobs. The roles are usually specified by user responsibilities, and qualifications. For each user, *a role-based access control (RBAC)* policy determines the collection of roles the user is allowed to perform (undertake) within the organization. With each role, there is a associated set of functions (transactions) which are allowed to be executed by a holder of the role. The user cannot pass access permissions to other users [84]. This is the fundamental difference between RBAC and DAC.

A *role* can be thought of as a set of *transactions* that a user or set of users can perform within an organization. A transaction can be considered as a *transformation procedure (TP)* (a program or portion of a program) plus a set of data items accessed by the $TP$. Each role has an associated set of individual members. The set of roles and their association with transactions is defined by the system administrator. Moreover, the membership in a role is granted and revoked by the system administrator [84].

The usual grouping mechanism of DAC can be used to implement roles [182]. The difference between groups and roles is the same as the one between a security policy and a security mechanism. Two very important differences between groups and roles are as follows [183]:

1. Groups are essentially a discretionary mechanism whereas roles are non-discretionary. The ability to assign permissions to a group is usual discretionary. On the other hand, the allocation of permissions to a role, as well as determination of membership in a role, are both intended to be non-discretionary.

2. The nature of permissions allocated to a role is significantly different than the usual read, write, execute, etc.

A common challenge in the design of roles is to ensure *separation of duties.* This requires that for particular sets of transactions, no single individual is allowed to execute all transactions from the set. The separation of duties can be either static (being built directly into the role definitions) or dynamic (with access constraints based on the previous access history of the affected entities) [2].

The static separation requirements can be implemented simply by the assignment of individuals to roles and allocation of transactions to roles. A more difficult case is the dynamic separation of duty, where the compliance with requirements can only be determined during the system operation. For example, the dynamic policy will allow an individual to take on both the initiator and authority roles for payment, with an exception that no one could authorize payments they had initiated. The system must use both the role and the user ID to check access to transactions based on *audit* records of previous accesses. The *audit* records of previous accesses introduce the following problems. How long must audit records be retained? How are they managed in a distributed environment? Sandhu [180] discusses an automated separation of duties with the use of roles and *transactions control expressions (TCE)*.

For detailed discussion of the definition of RBAC, different possible approaches for role organization, and different proposed solutions for separation of duties, the reader is directed to [8, 49, 83, 84, 85, 90, 108, 152, 154, 165, 180, 183, 185, 186, 193, 206, 207, 214, 215].

## 3.11 Implementation Strategies

The 1982 Air Force Summer Study [55] suggested two architectures for building secure multi-level database management systems (DBMS): *Trusted Subject Architecture* and *Woods Hole architectures.* The *Woods Hole architectures* are the *Kernelized*, the *Replicated* architectures, and the *Integrity Lock.* In the *Trusted Subject Architecture* both a trusted DBMS and a trusted operating system (OS) are used, while in the *Woods Hole architectures*, an untrusted DBMS is employed with an additional trusted filter. Thus, the *Trusted Subject Architecture* requires either the development of a new DBMS from scratch, or the extension of the security features of an existing DBMS. In the following, we present a brief review of these architectures. For detailed discussion, the reader is directed to [46, chapter 4].

In the *Kernelized* DBMS, the multi-level database is partitioned into single-level databases which are stored separately. A trusted OS which is responsible for the physical access to data in the database is used to enforce the mandatory access control. The decomposition and recovery algorithms must be properly defined to guarantee the correctness and system efficiency. The decomposition algorithm partitions a multi-level database into several single-level databases [38, 69, 115, see for example]. The recovery algorithm is performed on single-level entities when they are retrieved to generate a multi-level entity containing only the data the user is cleared to access [119, as an example]. This architecture is used by *SeaView* [141, 142], the commercial DBMS *Oracle* [46], and in the most current secure databases [33, 38, 149, 210, as examples].

In the *replicated* architecture, there is a database at each security level which contains all data whose classification are less or equal to the database security level. Each database is associated with a separate DBMS. In this architecture, we need to replicate lower security-level data in all database containing higher security-level data. There are only a few research projects which are based on this architecture [58, 59, 146].

In the *Integrity Lock*, each data element is assigned a checksum that indicates its security level which initially is on either secret or confidential level. The *Integrity Lock* consists of three main components: the *Untrusted Front End (UTFE)*, the *Trusted Front End (TFE)*, and the untrusted DBMS. The TFE authenticates and verifies a user, updates tuples, and handles all projections and creations of new data entities. The UTFE handles parsing query, formatting output, and computations. The untrusted DBMS handles searches of the database, selects requested tuples, inserts data tuples into the database, reorganizes database, and manages the storage. Integrity Lock can provide security at the record, attribute, or data element level. This architecture is vulnerable to *Trojan Horse* attacks, and inference attacks (see [67, 114]). Denning [67] proposes a new approach called a *commutative filter*. The *commutative filter* is inserted between the users and the DBMS in order to assure the elimination of the inference attack provided that the DBMS is free of *Trojan Horses* that leak data. The implementation of the integrity lock design was done in the MISTRESS database management system with the Unix operating system [94].

## 3.12 Inference Control

The word "inference" means *"forming a conclusion from premises"*. Users of any database can draw inferences from the information they have obtained from the database and from some prior additional information (called supplementary knowledge) they have. The inference can lead to information disclosure if the user is able to deduce the information they are not authorized to access. This is the *inference problem* in the database security. Thus for database systems that can contain sensitive information about individuals or companies, (such as statistical databases or multi-level security databases), information flow controls may be inadequate. New solutions and mechanisms are required to deal with the inference problem.

### 3.12.1 Statistical Databases

A *statistical database* (SDB) is a database that is used for statistical queries (for example, averages, counts) on subsets of the database entities. *Inference* in an SDB allows to reveal confidential information of single entities by the analysis of statistical queries [46]. A lot of work (see [65, chapter 6], [46, chapter 5], [3]) has been done and many solutions are proposed to make SDB secure against inference. The solutions can apply different mechanisms to restrict the access and prevent against inference. Typically, these mechanisms are based on *query restriction, data perturbation (or masking)*, *output perturbation*, and *conceptual restriction.* For detailed discussion of these mechanisms, the reader is directed to [3, 46, 65].

### 3.12.2 Multi-level Databases

In multi-level security database systems (MLS/DBS), the inference problem arises whenever some low-level data $x$ can be used to derive partial or exact information about some other high-level data $y$. In some cases, even the learning of the existence of the information may be unacceptable. An inferential link that may allow information to flow from a high security class to a low security class is called an *inference channel* or a *covert channel.* Wiseman [224] identifies four aspects of the inference in multi-level security databases: *addressing inference channel, relationship inference channel, aggregation problem*, and *architectonic problem.*

The *addressing inference channel* arises through the data role in addressing rather than from it being accessed. In the access protection model, labeling does not prevent the data from being addressed, only prevents from being seen directly. A user can receive the unauthorized information by addressing the data using the query mechanisms and inferring its existence from the result [35, 68, 223].

The *relationship inference channel* arises whenever system objects are classified lower than the relationships between the objects. To protect a database against the relationship inference channel, the control mechanism should allow users with a low clearance to access information only if the relationship with a higher classification cannot be deducted [132, 168].

The *aggregation problem* arises when a user can form aggregates of related unclassified data that infers classified data. Here, the collection of low classified information has a higher security classification. So it often requires us to assign a higher security level to the collection [60, 98, 139].

The *architectonic problem* arises when the structural information of the database objects (database schema) is classified lower than the database objects. Users with a lower clearance can use the structure information of database together with authorized information to deduce information with higher security [198].

### Proposed Solutions

Recently some solutions have been proposed to handle the inference problem. The inference problem can be dealt during the database design [69, 138, 195, 198, 213], or during the query processing [120, 209, 212].

In the first approach, security constraints during the database design are handled in such a way that security violations via inference cannot occur. So many inference problems can be overcome through a good design. The SeaView [144, 141], ASD-Views [88], and SWORD [177] projects are examples of this approach. Other works have also been done to provide tools which allow data designers to analyze a database schema for potential inference problems and remove those. DISSECT [89, 168], Multilevel Semantic Net [213], IAT [111], and Database Inference Controller [42] are examples. There are some non-reference formal models which can be used to verify any good design against them as well. The works of [198], [204], [111], [36], and [132] are examples.

In the second approach, the query processor is augmented with a logic-based inference engine to handle inferences during query processing. The inference engine will attempt to prevent users from the disclosure of the protected information. Some researchers argue that inferences can be the most effectively handled and thus prevented during query processing because the most users build their supplementary knowledge from responses they receive by querying the database.

LDV [98] is an example of this approach.

## 3.13 Cryptographic Control

The information is protected against the disclosure by providing a reliable operating system and a secure access control mechanism for database system. Unfortunately, there is a great deal of evidence that even sufficiently complex operating and database management systems may have security holes or Trojan Horses which could be a potential threat to the information security. Better protection of information can be obtained by using several security measures simultaneously.

The use of cryptographic techniques to protect database systems represents an important security mechanism. Through these techniques, secrecy of information is assured by making data unreadable to anyone but authorized users with cryptographic keys [65]. In general, cryptographic techniques might be used: to provide user authentication, to maintain the integrity (data authenticity) and the secrecy of data, to protect information during transition and during processing from disclosure, to protect private data from an unauthorized access in the hierarchical access control, etc.

Application of cryptography has the potential to solve both the problem of data integrity and the problem of data confidentiality. However, encryption has some disadvantages particularly in databases. These are: inability of record searching (particularly in the case of pattern or partial-match and range queries as the structure of data is lost due to encryption), the necessity of key generation, data expansion, impossibility to compute statistical data from the encryption information, and the overhead related to encryption and decryption operations. The application of cryptography for database security has been a main topic in [6, 62, 66, 65, 92, 96, 103, 104, 105, 166, 178, 191, 216, see for example]. For detailed discussion of cryptography and its applications in database security, the reader is directed to [65, 103, 166].

## 3.14 Summary and Remarks

In this chapter, we have illustrated security requirements, threats, and discretionary and mandatory security models for the protection of conventional database systems and OODB systems. We have also discussed security issues and concerns that are being expressed for OODB systems. Several proposals for discretionary and mandatory security models for the protection of conventional and object-oriented databases have been presented and some of their drawbacks have been discussed. The models offer some solutions to the protection of OODB systems. However, each of them addresses only some of the security issues, therefore leaving many questions unanswered. The following topics need further work:

- Design a more efficient mechanism for the access control on the instance-level. When the number of objects grows, the support of protection matrix and the search at the granularity of object-instances is extremely hard and time consuming. Finding implicit authorizations will be even harder.
- Design a general discretionary access control model with flexible and practical solutions to problems such as content-dependent and context-dependent authorizations with positive and negative authorizations.
- Design an extended multi-level access control model which allows to use the *class, object instance,* or the *instance-variables* as the unit of authorization and labeling.
- Design a formal verification protocol to check the correctness of the access control model.

# Chapter 4

# A Cryptographic Mechanism for Discretionary Access Control in OODB Systems

## 4.1 Introduction

In an OODB system model classes, inheritance, and composite data structures allow to express rules for computing implicit authorizations from explicit ones. Hence, an access request to OODB objects may require to apply authorization rules on explicit privileges to derive implicit authorizations. An important question is whether implicit authorizations must be evaluated each time an access request is processed, or they should be computed and stored as redundant authorizations. If the implicit authorizations are stored, the size of protection matrix gets too big. As the result, the support of protection matrix and the process of access requests become inefficient. In this chapter, we are going to suggest a solution to this problem. The solution applies cryptographic hashing.

Inheritance (*inclusion* relation) and composite data structure (*is part of* relation) create hierarchical structures in OODB systems [219]. An interesting question is how to extend the known cryptographic solutions for hierarchical access control in OODB systems. Two solutions were published in the literature. One was based on the RSA cryptosystem [5] and the other applied one way hash functions [225].

The main drawback of the first solution is that it is only applicable to a fixed known hierarchy with no provision for possible changes to the hierarchy. Moreover, the integer values associated with the nodes of the hierarchical structure become extremely large when the number of nodes is large. We use the second solution, proposed by Zheng, Hardjono, and Pieprzyk [225], which is based on the sibling intractable function families (SIFF), and we show how to develop a cryptographic solution for discretionary access control (DAC) in OODB systems. The solution applies pseudo-random functions, SIFF, and an authorization class (instead of access control lists or protection matrix). The advantages of our approach are as follows.

1. Pseudo-random functions and SIFF are used to produce a pair of unique and secure *access keys* and *passwords* for each database object (instances or classes) and its owner. Access keys and passwords for implicit authorizations may also be derived from related database objects during query processing.

2. An authorization class (*AC*) is employed instead of access control lists (or protection matrix). AC stores the current state of authorizations and use SIFF to derive authorization-instance identifiers associated with users. This results in a system that is more efficient and practical. This is true because any alteration of the membership of user groups requires manipulation of *AC* only rather than checking all access control lists in the database. Moreover, because of data structure consistency, database system operation can be used to manipulate *AC*. Hence, an access request may be verified during query processing.

3. The security of the system relies on the indistinguishability of pseudo-random functions from the truly random one and the difficulty of finding collisions for SIFF, both of which are known to be difficult [225].

4. Operations such as grant, revoke, propagation of rights, and the required changes due to the alterations of the user groups and of the class structure are relatively easy to perform.

5. The existence of multiple owners of a database object (instances and classes) is possible.

## 4.2 Background

Denote by $\aleph$ the set of all positive integers, $n$ the security parameter, $\sum$ the alphabet $\{0, 1\}$ and $l(n)$, $k(n)$, and $m(n)$ polynomials in $n$ from $\aleph$ to $\aleph$.

### 4.2.1 Sibling Intractable Function Families (SIFF)

Zheng, Hardjono, and Pieprzyk [225] introduced the notion of sibling intractable function family (SIFF) which is a generalization of the concept of the universal one-way hash function defined by Naor and Yung [151]. A universal one-way hash function is a class of hash functions with the property that the number of functions that map any collection of $r$ distinct input strings to the same hash value is fixed. SIFF is the universal one-way hash function family with the additional property that given a set of colliding sequences, it is computationally infeasible to find another sequence that collides with the initial sets. This means that if a SIFF function $h$ maps the bit string $x_1, x_2, \ldots, x_i$ to the same string, it must be computationally infeasible to find another string $x'$ such that

$$h(x') = h(x_1) = \ldots = h(x_i).$$

In the following, we give the definition of SIFF [103, 225] which is used in this chapter. For detailed discussion, the reader is directed to [103, 225].

Let $x \in_R X$ denote an element $x$ which is randomly chosen from the set $X$ with the uniform probability. A sibling finder F is a probabilistic polynomial time algorithm that is given an input $X = \{x_1, x_2, \ldots, x_i\}$ and the description of $h$, where $x_i \in \sum^{l(n)}$ and $h$ is a hash function that maps $x_1, x_2, \ldots, x_i$ to the same string. The finder $F$ outputs either "?" ("I cannot find") or a string $x' \in \sum^{l(n)}$ such that $x' \notin X$ and $h(x') = h(x_1) = \ldots = h(x_i)$.

**Definition 4.1** *Let $k(n)$ be a polynomial with $k(n) \geq 1$ and $H = \{H_n | n \in \aleph\}$ be a family of functions that are computable in polynomial time and samplable. Moreover they have the collision accessibility property and map $l(n)$-bit input into $m(n)$-bit output strings $H_n = \{h | h : \sum^{l(n)} \rightarrow \sum^{m(n)}\}$. Assume $X = \{x_1, x_2, \ldots, x_i\}$ be a set of $i$ initial strings, where $1 \leq i \leq k(n)$. $H$ is a k-sibling intractable function family (or k-SIFF) if for all $1 \leq i \leq k(n)$, any sibling finder F, any polynomial $Q(n)$, and for all sufficiently large $n$,*

$$Pr\{F(X, h) \neq ?\} < \frac{1}{Q(n)},$$

*where $h$ is chosen randomly and uniformly from $H_n^X \subset H_n$. $H_n^X$ is the set of all functions that map $x_1, x_2, \ldots, x_i$ to the same strings in $\sum^{m(n)}$. The probability $Pr\{F(X,h) \neq ?\}$ is computed over $H_n^X$ and the sample space of all finite strings of coin flips that $F$ could have tossed [103, 225].* □

### 4.2.2 Sketch of Implementation

As mentioned in [225], SIFF can be constructed from any universal one-way hash function family (OWHF). Figure 4.1 illustrates a possible construction of a k-SIFF hash function.

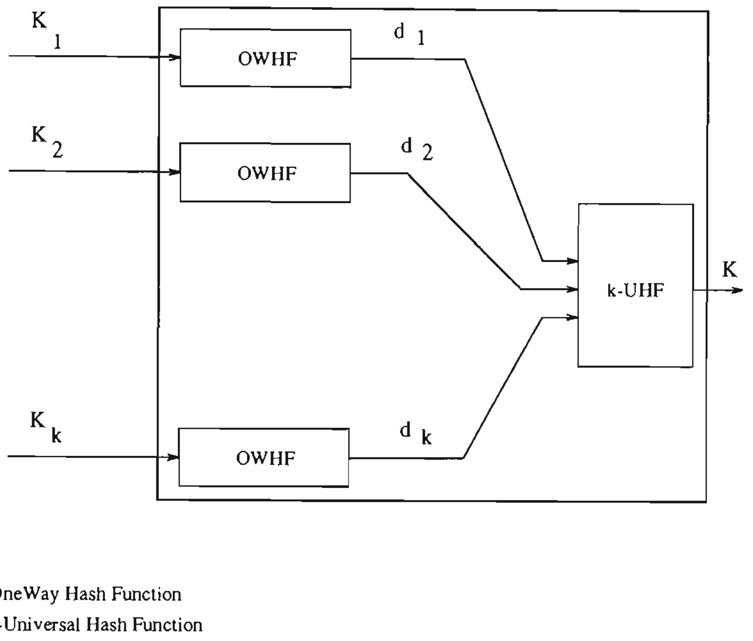


Figure 4.1: A sketch of Implementation of k-SIFF hash function.

The OWHF can be any one-way hash function such as MD4, MD5 [172, 173], or HAVAL [226] for which a fast hardware implementation is available. Note that the security of MD4, MD5, or HAVAL has not been proved formally, however no major weaknesses of these functions have been reported. As stated in [220, 225], a possible candidate for a k-universal hash function family (k-UHF) with the collision accessibility property are polynomials over finite fields.

## 4.3 Security Policy

The specification of access control may involve a range of policies choices. The choice of policies for security is important because it influences the flexibility, usability, and performance of the system [82]. In this chapter, our considerations are restricted to the DAC.

### 4.3.1 General Policies

In the authorization system for an OODB system, the granularity of the control, i.e., the smallest unit of authorization, may be a class, an object-instance, or a property (or instance variable) [169]. We choose the units of authorization to be classes and instances of classes. It means that one user may be granted access to a complete class, while another user may be granted access to its instance. We will use the term *entity* to refer to either a class or an instance of a class.

An authorization system allows a number of possible access privileges on the protected entities. We assume the following set of privileges in the authorization system: *read-definition, read, write, delete, execute*, and *create.* The privileges are partially ordered such that authorization to access privileges of higher order implies authorization to access privileges of lower order. The assumed order is:

*write* > *execute* > *read* >*read-definition*,

*create* > *execute* > *read* >*read-definition*, and

*delete* > *read* > *read-definition.*

It is worth noting that the *execute* privilege is used to call and execute methods associated with the class of an object, and implies *read* and *read-definition* rights. It means that the method can access the required definitions and value of instance-variables and objects, and the holder of the execute privilege can access the return result of the method as well. The state of the object will not change. In order to change the state of an object, the privilege *write* is required.

Our authorization system is chosen to be a closed system, i.e., each privilege must be explicitly authorized. Hence, the absence of appropriate authorizations is interpreted as *"access not allowed"*.

### 4.3.2 Administrative policies

Administrative policies determine who is allowed to grant and revoke authorizations to entities (classes or objects). There are two approaches: *centralized* and *decentralized* administrations. In centralized administration, the grant and revocation of authorizations are done by a special user or users called *database administrator* or *security officers.* The centralized administration is sometimes too restrictive. In decentralized administration, users are allowed to grant and revoke authorizations by applying *ownership* policy or other mechanisms. Here, we use the decentralized administration and allow each entity to have its owner. Users are grouped and each group has its sponsor who gives authorizations to members of the group. The database administrator (security officers) duties include admitting new users to the database system and revoking/replacing the ownership.

Each entity (object or class) has its owner. Whenever a user creates an entity, (s)he will be its owner and have the full authority over it. The owner of the entity grants and revokes privileges for other users. The owner authority is limited to the entity (s)he created. The owner has only implicit *read-definition, read*, and *execute* privileges to the entities which has relationships with the owner entity. The owner must get permission explicitly for other privileges such as *write*, *delete*, and *create*. The ownership can also be granted and revoked by the creator of the object.

It is worth noting three points. First, each class has its owner and the owner can be different from the owners of the class instances. Owners of classes have full authority over their classes, and have implicit *read-definition, read*, and *execute* authorization rights on relevant instances of classes. Second, an authorization on a class propagates to instances only when the grantor has the same rights or is their owner. Third, a user must have *create* privilege in order to create an object of a class.

A group is defined as a set of users or a collection of smaller groups. Groups are not necessarily disjoint. This means that a user may be a member of more than one group. Groups may be members of other groups provided they do not belong (directly or indirectly) to any of its members. The resulting group hierarchy has to be a directed acyclic graph. Figure 4.2 shows an example of a group hierarchy.

Each group has its sponsor who administers it. The sponsor can add new members to the group or remove members from the group. Any user who has the sponsorship privilege may create a new group and grant the sponsorship privilege to other users.

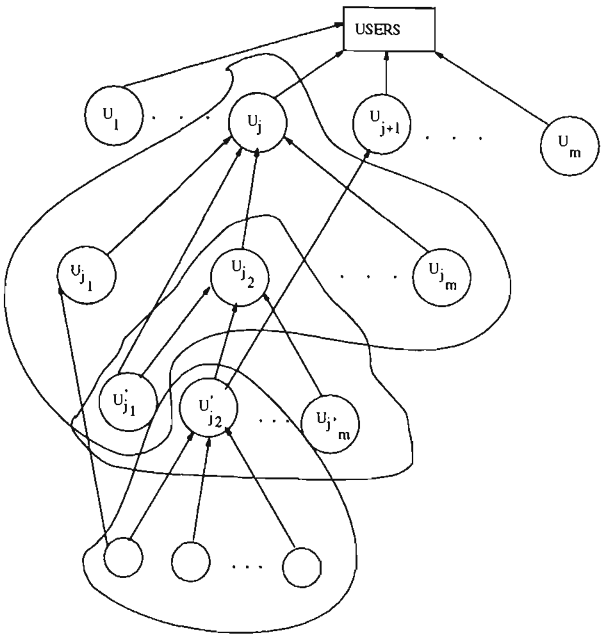


Figure 4.2: User groups Hierarchy.

### 4.3.3 Implicit policies

There are two different types of object hierarchies in OODB systems: *class-composition* and *class-inheritance hierarchy* [126]. To access the full information regarding an entity, a user requires to have the same authorizations along the hierarchies. There are two policies: *visibility from above* and *visibility from below* that define how an explicit authorization may propagate along the hierarchies [130].

In the object-composition hierarchy, the root corresponds to a complex object and other objects in the hierarchy define its internal structure. If users are authorized to access the root, they should also be authorized to access all information of the descendants of the root. This is called *visibility from above.*

The classes can also be organized in the inheritance (class/subclass) hierarchy. In this case, the access to a subclass implies the access to all objects of the superclasses in the inheritance hierarchy.

In order to indicate how privileges are propagated along the hierarchy, different types of authorization should be identified. Two possible types of authorization are: *partial* and *full* [169].

In the object-composition hierarchy, a user with the *full authorization* for a set of privileges (can be *read-definition, read* and/or *execute*) over an entity has the same rights to the components of the entity, i.e., the entities that form the structural part of the entity. In the case of the *partial authorization,* the access to an entity does not extend to its components.

In the inheritance hierarchy, when users have the *full authorization* for a set of privileges (the set can include *read-definition, read* and *execute*) over an object of a subclass, they have implicitly the same rights to the relevant objects of the superclasses. In the case of the *partial authorization*, a user can access the object only. However, users that are given authorizations to an object of a class, will not be authorized to access the objects of subclasses of that class unless they are authorized explicitly or are the owners of the objects of those subclasses.

Note that for other privileges such as *create, write*, and *delete*, the user must be explicitly authorized by the owner of the related objects, unless the two objects have the same owner.

## 4.4 Notations, Assumptions, and Definitions

### 4.4.1 Notations

- $O_i$ and $OID_i$ are the names of the i-th object and the i-th object identifier, respectively. $C_i$ is the name of the i-th class. $E_i$ denotes the i-th entity name (object or class) which can be either $O_i$ or $Ci$. $U_j$ denotes the j-th user login-name. $ACID_{j,i,k}$ is the authorization-instance identifier of the user $U_j$ for the object $E_i$ granted by $U_k$. We use an $n$-bit string to represent $OID_i$, $O_i$, $C_i$, $U_j$, and $ACID_{j,i,k}$.

- $PS_j$ denotes the login password of the user $U_j$. This password is chosen by $U_j$. It is long enough and is kept secret by $U_j$.

- $||$ and $\oplus$ denote concatenation and exclusive-or (XOR), respectively.

- TM and DBMS denote a tamper-proof module, and a database management system, respectively.

- $K_{db}$ is the database cryptographic key which is kept in a secure place and is only accessible to TM. $\{x\}_{K_{db}}$ stands for the ciphertext of $x$ generated using the key $K_{db}$.

### 4.4.2 Assumptions

1. $F = \{F_n | n \in \aleph\}$ is a pseudo-random function family, where $F_n = \{f_K | f_K : \sum^n \rightarrow \sum^n,\ K \in \sum^n\}$.

2. $H = \{H_n | n \in \aleph\}$, where $H_n = \{h | h : \sum^{2n} \rightarrow \sum^n\}$ is a k-SIFF mapping $2n$-bit inputs to $n$-bit output strings. $k$ is a parameter which is chosen in such a way that no database entity has more than $k$ relevant entities and no group has more than $k$ users.

3. Random $n$-bit strings $K^{rd}, K^r, K^w, K^e, K^d, K^c$ correspond to *read-definition*, *read*, *write*, *execute*, *delete*, and *create*, respectively. They are accessible to TM only.

### 4.4.3 Definitions

Each class and object in our system has the following specification.

**Definition 4.2** *A class C is represented by a tuple:*
*(CNAME, PNAME, "class-struct", "method-list", SECURITY-INFO).*
*CNAME is a unique name of C given by its creator. PNAME is the parent name of C. The "class-struct" is its structure, and "method-list" is the list of methods that can be executed by users if they have the execute privilege. SECURITY-INFO specifies class authorization information which is an aggregation of the CKEYS-LIST and H-FUNCTION. CKEYS-LIST is a pair of access keys* $(K_i^P, K_i^F)$ *corresponding to partial and full authorization. H-FUNCTION describes the hash function that must be used by the related classes to derive the access key* $K_i^F$. □

**Definition 4.3** *The class-struct is* $[P_i : \rho_i(T_i), \ldots, P_k : \rho_k(T_k)]$, *where* $P_i$ *is a name of property,* $T_i$ *is a type name of the respective property and* $\rho_i$ *is an optional type constructor, e.g. set-of, collection-of, array-of, ordered list, etc. The set of type names includes the names of atomic data types like integer, real, string, etc. as well as the names of classes that have been pre-defined.* □

**Definition 4.4** *An object O is a tuple:*
*(OID, ONAME, CNAME, "state", SECURITY-INFO). OID is the identifier of the object and created by DBMS. ONAME is the name of the object given by its creator. CNAME indicates the name of the class to which O belongs. "state" is*

*the associated state of the object. SECURITY-INFO specifies object authorization information which is an aggregation of the OKEYS-LIST and H-FUNCTION. OKEYS-LIST is a pair of access keys* $(K_i^P, K_i^F)$ *corresponding to partial and full authorization. H-FUNCTION indicates the hash function that must be used by the related classes or objects to derive the access key* $K_i^F$ *of the object* $O$. □

The definitions of superclass and ancestor are as follows.

**Definition 4.5** *Ancestor of class* $C_i$ *is any class* $C_k$ *such that either*
*(1)* $C_k = [\ldots, P_i : \rho_i(C_i), \ldots]$ *or*
*(2)* $C_k = [\ldots, P_j : \rho_j(C_j), \ldots]$ *and* $C_j$ *is ancestor of* $C_i$. □

**Definition 4.6** *Superclass of* $C_i$ *is any class* $C_k$ *such that either*
*(1)* $C_i = (C_i, C_k, \ldots)$ *or*
*(2)* $C_k = (C_k, C_j, \ldots)$ *and* $C_j$ *is a superclass of* $C_i$. □

In order to enforce DAC security requirements and to protect an entity against unauthorized accesses, the authorization system has to know the exact user privileges. This can be accomplished by storing the explicit privileges and necessary DAC information in the *authorization class (AC)*.

**Definition 4.7** *An Authorization Class (AC) is a tuple:*
*(GRANTEE, ENAME, GRANTOR, MEMBER-LIST, DAC-INFO).*
*GRANTEE indicates the user who is authorized to access the entity. ENAME specifies the entity which can be a class name or an object identifier. GRANTOR names the user who has authorized the GRANTEE to access the entity. MEMBER-LIST is the list of users who are the members of the group whose sponsor is the GRANTEE. DAC-INFO specifies DAC information and has the form (OP-RIGHTS, AUTH-TYPE, SPONSORSHIP, OWNERSHIP, H-FUNCTION, PSWORD). OP-RIGHTS indicates the list of privileges which the GRANTEE has on the entity (it could be read-definition, read, write, execute, delete, create, and all; the word "all" is used to indicate all possible access privileges). AUTH-TYPE (F or P) specifies full or partial authorization. SPONSORSHIP (YES or NO) indicates if the GRANTEE can be the sponsor of a group (or groups) (indicated by the GRANTOR), and is able to propagate their privileges to the group members. OWNERSHIP (YES or NO) specifies whether GRANTEE has ownership privilege. H-FUNCTION indicates the hash function that must be used to derive the grantor's password. PSWORD stores the user password.* □

Note that the values of the *DAC-INFO* and *SECURITY-INFO* are encrypted with the $K_{db}$ by TM.

## 4.5 Proposed Solution

Our main goal is to design a cryptographic mechanism for discretionary access control in OODB systems. Thus we will not consider other security issues such as authentication and secrecy of stored data. To enforce authentication and secrecy, the scheme proposed by Hardjono, Zheng and Seberry [101, 102] for database authentication based on SIFF can be applied. A discussion of their scheme is presented in Appendix A. We assume that the user authentication is done by the underlying operating system, and is secure. Also, we use a tamper-proof module (TM) to perform all necessary cryptographic operations, to generate needed cryptographic elements, and to verify the validity of access attempts. The security of TM relies on the security of underlying operating system and DBMS. TM can be an interface between the user and the database system, or between the database and physical layer, or a separate function in the database system. Figure 4.3 shows the position of TM when it is a separate function in DBMS.

DBMS provides essential authorization information such as the entity identifier and access privileges in plain form and user password, access key, and SIFF in encrypted form to TM. Then TM evaluates the request according to the algorithms described in this section and Section 4.6, and passes the result to DBMS.

To protect an entity against unauthorized accesses, the authorization system needs to know the users authorization rights. There are two possible approaches to accomplish this. In the first, all authorizations either explicit or implicit are stored. In the second, only explicit authorizations are stored, and implicit ones are derived each time when the access request is processed. The first approach is inefficient and time consuming when the number of object-instances is large. The second approach is even worse if we use access control lists to store explicit authorizations. Here, we show how to improve the second approach. To do so, we propose a cryptographic mechanism using SIFF to evaluate implicit authorizations from explicit authorizations stored in the Authorization Class (AC) each time for checking validity of access request which is relatively straight forward and efficient.

To allow access to an entity $i$ (object or class), we must be able to produce

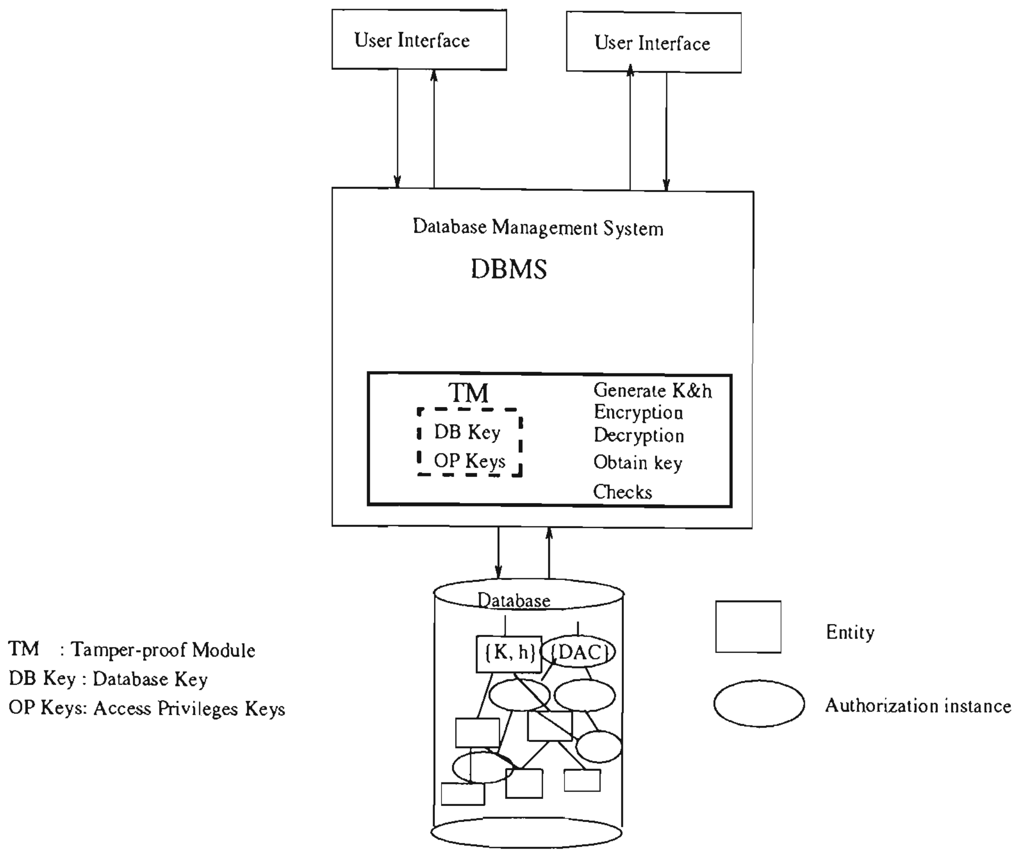


Figure 4.3: A possible implementation of TM.

access keys $K_i^P$ and $K_i^F$ for the entity. $K_i^P$ and $K_i^F$ correspond to the partial and the full authorization, respectively. $K_i^F$ can be derived from the access key of the related objects. The relationship can be either the inheritance *(is-a)* or the aggregation *(is-part-of)*. In the case of the inheritance, the access key $K_i^F$ can be derived from the access keys of the instances of subclasses of the entity $i$. Inheritance takes on four different forms specified in Chapter 2. Whereas in the case of the aggregation, the $K_i^F$ can be computed from the access keys of the objects of ancestors of the entity $i$. In other words, the access key $K_i^P$ guarantees partial explicit authorization access, and $K_i^F$ insures implicit authorization rights along the hierarchies: *inheritance*, and *composite*. Every time a user requests the access to a specific entity $i$ either $K_i^F$ or $K_i^P$ is computed and compared with the stored one by TM. If they match, the access is permitted otherwise denied.

Next, we discuss algorithms for the generation of access keys ( $K_i^P$, $K_i^F$), passwords, and SIFF associated with entities and users.

### 4.5.1 Creating

When a user $U_j$ (with login password $PS_j$) creates an entity $E_i$ by running *create command,* access keys for this entity are generated. Note that the entity can be either a class $C_i$ or an object $O_i$.

**Case 1. Partial authorization.**

**Step 1.** TM calculates the password $n_{j,i} = f_{PS_j}(U_j \oplus E_i)$ of the user $U_j$ for the entity $E_i$.
**Step 2.** TM selects at random the access key $K_i^P$ for the entity $E_i$ ($K_i^P \in_R \sum^n$) for partial authorization.
**Step 3.** TM selects at random a SIFF hash function $h_i^P \in_R H_n$ for partial authorization. The function has the following collisions:

$$h_i^P(n_{j,i}\|K^{rd}) = h_i^P(n_{j,i}\|K^r) = h_i^P(n_{j,i}\|K^e) = h_i^P(n_{j,i}\|K^w) = h_i^P(n_{j,i}\|K^d) = h_i^P(n_{j,i}\|K^c) = K_i^P \qquad (1)$$

TM also encrypts DAC-INFO, $\{(\text{“}all\text{”}, \text{“}F\text{”}, \text{“}yes\text{”}, \text{“}yes\text{”}, h_i^P, n_{j,i})\}_{K_{db}}$. The word *all* is used to indicated all possible access privileges.
**Step 4.** DBMS creates the object ($U_j, E_i, U_j$, MEMBER-LIST, DAC-INFO) which is an instance of the Authorization Class (AC).

**Case 2. Full authorization.**

Suppose that objects $O_{l_1}, O_{l_2}, \ldots, O_{l_p}$ with access keys $K_{l_1}^F, K_{l_2}^F, \ldots, K_{l_p}^F$ are related to the object $O_i$ (via either inheritance or aggregation).
**Step 1.** TM selects at random the access key $K_i^F$ for the object $O_i$ ($K_i^F \in_R \sum^n$) for full authorization.
**Step 2.** TM selects at random a SIFF hash function $h_i^F \in_R H_n$ for the full authorization. The function has the following collisions:

$$h_i^F(K_{l_1}^F\|K^{rd}) = h_i^F(K_{l_1}^F\|K^r) = h_i^F(K_{l_1}^F\|K^e) = h_i^F(K_{l_2}^F\|K^{rd}) = h_i^F(K_{l_2}^F\|K^r) = h_i^F(K_{l_2}^F\|K^e) = \ldots = h_i^F(K_{l_p}^F\|K^{rd}) = h_i^F(K_{l_p}^F\|K^r) = h_i^F(K_{l_p}^F\|K^e) = K_i^F \qquad (2)$$

Clearly, users who have access to related objects $O_{l_s}$ ($1 \leq s \leq p$), can also access the object $O_i$. The access to the object $O_i$ is granted only if TM can regenerate $K_i^F$ from a pair (a related object keys $O_{l_s}$ and a suitable privilege key ($K^{rd}, K^r$, and $K^e$)). Note that in the case of inheritance, $O_{l_s}$ ($1 \leq s \leq p$) are instances of

subclasses of the object $O_i$. Whereas in the case of aggregation, $O_{l_s}$ $(1 \leq s \leq p)$ are objects of ancestors of the object $O_i$.
**Step 3.** DBMS appends the hash function $\{h_i^F\}$ to the *H-FUNCTION* of the object $O_i$.

Note the following two points. In the case of the full authorization of a class, the associated access key $K_i^P$ is computed first, then the full authorization keys associated with the related to the class objects are derived from its instances. If the owner of an entity is replaced by a new one or if the login password of the owner has been changed, then in both cases the process prescribed above must be done again.

### 4.5.2 Authorization Administration

To be complete, an authorization system must include mechanism for the authorization handling. Here, we present algorithms to perform the *grant, revoke,* and *ownership transfer* operations.

#### Granting

Suppose that the grantor $U_j$ has the password $n_{j,i}$ for the entity $E_i$. Assume $U_j$ wants to give access to $E_i$ for $m$ grantees $U_{l_1}, U_{l_2}, \ldots, U_{l_m}$ (with login passwords $PS_{l_1}, PS_{l_2}, \ldots, PS_{l_m}$). If $U_j$ runs the *grant command,* the following steps will be completed. $U_j$ can be the owner of $E_i$ or the sponsor of a group.
**Step 1.** TM calculates the password $n_{l_s,i,j} = f_{PS_{l_s}}(E_i \oplus U_j)$ of the grantee $U_{l_s}$ for the entity $E_i$, $s = 1, \ldots, m$.
**Step 2.** TM selects at random a SIFF hash function $h_{j,i} \in_R H_n$ such that

$$h_{j,i}(n_{l_1,i,j}) = h_{j,i}(n_{l_2,i,j}) = \ldots = h_{j,i}(n_{l_m,i,j}) = ACID_{j,i,k}$$

This step ensures that all grantees $U_{l_1}, U_{l_2}, \ldots, U_{l_m}$ (the grantees are member of the group whose sponsor is the grantor $U_j$) can directly compute the $ACID_{j,i,k}$ of the $U_j$ and access the authorization-instance related to the $U_j$ for the entity $E_i$ granted by $U_k$.
**Step 3.** TM encrypts the DAC-INFO, $\{($"access privileges", "P/F", "yes/no", "yes/no", $h_{j,i}, n_{l_s,i,j})\}_{K_{db}}$.
**Step 4.** DBMS creates the object ($U_{l_s}, E_i, U_j$, MEMBER-LIST, DAC-INFO) as an instances of the class AC ($ACID_{l_s,i,j}$) for $s = 1, \ldots, m$.

**Step 5.** DBMS updates the MEMBER-LIST of the authorization-instance related to the grantor $U_j$.

### Revoking

If a grantor $U_k$ wants to revoke $U_j$ authorization over the entity $E_i$, the following steps have to be completed.
**Step 1.** DBMS deletes the associated authorization-instance $ACID_{j,i,k}$ from AC.
**Step 2.** TM selects a new SIFF with one less collision for the group whose sponsor is $U_k$ (the user $U_j$ does not belong to the group anymore).
**Step 3.** TM replaces the old SIFF in the authorization-instance associated with users in the MEMBER-LIST of the sponsor with the new one.
**Step 4.** DBMS updates the MEMBER-LIST associated with $U_k$.

Section 4.8 discusses in detail the impact of the group updating on the authorization system.

### Ownership Transfer

An entity (class or object) can have several owners who may act independently. Suppose that the creator of $E_i$ is $U_j$ and $U_j$ wants to grant the ownership of $E_i$ to users $U_r$ and $U_s$ by executing *transfer-own* command. The following steps must be completed.
**Step 1.** TM computes passwords $n_{r,i} = f_{PS_r}(U_r \oplus E_i)$ and $n_{s,i} = f_{PS_s}(U_s \oplus E_i)$ for new owners.
**Step 2.** TM selects a new SIFF hash function with the following collisions:
$h_i^P(n_{j,i}||K^{rd}) = h_i^P(n_{j,i}||K^r) = h_i^P(n_{j,i}||K^e) = h_i^P(n_{j,i}||K^w) = h_i^P(n_{j,i}||K^d) = h_i^P(n_{j,i}||K^c) = h_i^P(n_{r,i}||K^{rd}) = h_i^P(n_{r,i}||K^r) = h_i^P(n_{r,i}||K^e) = h_i^P(n_{r,i}||K^w) = h_i^P(n_{r,i}||K^d) = h_i^P(n_{r,i}||K^c) = h_i^P(n_{s,i}||K^{rd}) = h_i^P(n_{s,i}||K^r) = h_i^P(n_{s,i}||K^e) = h_i^P(n_{s,i}||K^w) = h_i^P(n_{s,i}||K^d) = h_i^P(n_{s,i}||K^c) = K_i^P$
**Step 3.** DBMS updates the instance in the AC for $U_j$ and creates new instances for $U_r$ and $U_s$.

Note that if the creator of $E_i$ wants to revoke the ownership of $E_i$ from the user $U_s$ by executing *revoke-own* command. It is sufficient that TM selects a new SIFF hash function with the following collisions:
$h_i^P(n_{j,i}||K^{rd}) = h_i^P(n_{j,i}||K^r) = h_i^P(n_{j,i}||K^e) = h_i^P(n_{j,i}||K^w) = h_i^P(n_{j,i}||K^d) = h_i^P(n_{j,i}||K^c) = h_i^P(n_{r,i}||K^{rd}) = h_i^P(n_{r,i}||K^r) = h_i^P(n_{r,i}||K^e) = h_i^P(n_{r,i}||K^w) =$

$h_i^P(n_{r,i}||K^d) = h_i^P(n_{r,i}||K^c) = K_i^P$
As the result, all privileges granted by $U_s$ to other users will be deleted as well.

If OWNERSHIP is on and both GRANTEE and GRANTOR are $U_j$, then $U_j$ is considered the creator of the $E_i$.

## 4.6 Validation of Access Requests

The processing of a user query starts by checking if the user has appropriate privileges to the entities specified in the query. This is done by the authorization system.

In OODB systems, the hierarchical structure of an entity may, or may not, be included in the evaluation of the query and hence there are two forms of query: *simple queries* and *hierarchical queries.* A *simple query* has the following form:

- **retrieve** Target-clause [**from** Entry-clause] [**where** Qualification-clause];

  Target-clause denotes target entity names to be retrieved. Entry-clause (**from**) denotes sets of entities through which the target entity can be accessed. If the target entity is an object of a complex object, the Entry-clause may denote any of the ancestors of the target entity. In the case of inheritance hierarchy, the Entry-clause may denote any instance of subclasses of the target entity. It is worth noting that if a user does not have an explicit right to the target entity then it is essential for the Entry-clause to be specified. Qualification-clause (**where**) specifies Boolean combination of predicates that must be satisfied by the retrieved objects.

For a *hierarchical query,* the scope of the query also includes the hierarchical structure of the target object. This is specified by putting "*" immediately after the name of the object. A *hierarchical query* has the following form.

- **retrieve** Target-clause* [**from** Entry-clause] [**where** Qualification-clause];

  The syntax of the query is similar to a simple query. "*" indicates that the hierarchy must be included in the evaluation of the query, i.e., the value of all properties (or objects) of the entity specified in the Target-clause and its relevant entities must be retrieved.

### 4.6.1 Access Validation

The access validation of a query is done in two phases. First, the authority of the user who issues the query is checked, i.e., it must be checked whether the user has proper authorization rights to entities which are requested. This is the *user validation phase.* Second, the specified privileges to the entity retrieved by the query are forced. This is the *access validation phase.*

Without loss of generality, we assume that the user $U_l$ issues the query:

**retrieve** $E_j$* **from** $E_i$;

**Phase 1. User validation.**

**Step 1.** Retrieve the authorization instance related to the user $U_l$ for the entities $E_j$ or $E_i$.
**retrieve** AC **where**
$(GRANTEE = U_l$ **and** $ENAME = E_j$ **and** PSWORD $= f_{PS_l}(E_j \oplus GRANTOR))$
**or**
$(GRANTEE = U_l$ **and** $ENAME = E_i$ **and** PSWORD $= f_{PS_l}(E_i \oplus GRANTOR))$;
The verification that
(PSWORD $= f_{PS_l}(E_i \oplus GRANTOR)$), or (PSWORD $= f_{PS_l}(E_j \oplus GRANTOR)$) is done by TM. If there is no such instance of the Authorization Class AC then the system rejects the request. Suppose that the verification has been successful and the extracted instance is
$(U_l, E_k, U_{l'}$, MEMBER-LIST,
$\{($"access privileges", "F", "yes/no", "yes/no", $h_{l',k}, n_{l,l',k})\}_{K_{db}})$ where $k$ is either $j$ or $i$.
**Step 2.** Derive the password of the owner of the entity $E_k$, say $U_w$, i.e.,
**while**($OWNERSHIP \neq$ "yes")**do retrieve** $h_{l',k}(PSWORD)$;
If $OWNERSHIP$ is not on, TM obtains $ACID_{l',k,w} = h_{l',k}(PSWORD)$. Then DBMS retrieves the $ACID_{l',k,w}$ instance of AC. Suppose that the derived password and SIFF for the owner $U_w$ of $E_k$ are $n_{w,k}$ and $h_k^P$, respectively ($k$ is either $j$ or $i$). Then we enter the access validation phase.

**Phase 2. Access validation.**

Assume that

$h^{P^C}$ is SIFF hash function associated with a class,

$h^{P^O}$ and $h^{F^O}$ are SIFF hash functions used for an object (partial and full),

$K^{P^C}$ and $K^{F^C}$ are access keys used for a class (partial and full), and

$K^{P^O}$ and $K^{F^O}$ are access keys associated with an object (partial and full).

The validation proceeds as follows.

**Step 1.** One of the following retrieve statements is executed (this depends on the type of the entity (object or class)).

- If $E_j$ is an object $O_j$, then
  **retrieve** $O_j$ **where**
  $(h_j^{P^O}(n_{w,j}\|K^r) = K_j^{P^O})$ **or**
  $(h_i^{P^O}(n_{w,i}\|K^r) = K_i^{P^O}$ **and** $AUTH\text{-}TYPE$="F" **and** $h_j^{F^O}(K_i^{F^O}\|K^r)=K_j^{F^O})$;
- If $E_j$ is a class $C_j$, then
  **do**(for all object $O_s$ is in $C_j$)
  **retrieve** $O_s$ **where**
  $(h_j^{P^C}(n_{w,j}\|K^r) = K_j^{P^C})$ **or**
  $(h_i^{P^C}(n_{w,i}\|K^r) = K_i^{P^C}$ **and** $AUTH\text{-}TYPE$="F" **and** $h_s^{F^O}(K_i^{F^O}\|K^r)=K_s^{F^O})$;

**Step 2.** Retrieve the objects which are in relation with the entity $E_j$ (via either inheritance or aggregation). Let $O_s$ denote such an object.
**retrieve** $O_s$ **where**
$(h_j^{P^O}(n_{w,j}\|K^r) = K_j^{P^O}$ **and** $AUTH\text{-}TYPE$ ="F" **and** $h_s^{F^O}(K_j^{F^O}\|K^r)=K_s^{F^O})$
**or**
$(h_j^{P^C}(n_{w,j}\|K^r) = K_j^{P^C}$ **and** $AUTH\text{-}TYPE$ ="F" **and** $h_s^{F^O}(K_j^{F^O}\|K^r)=K_s^{F^O})$;
After all $O_s$ have been retrieved, the process finishes. In the above steps, all checks are done by TM.

Note that the access key of instances of descendants (in the case of the composite object) or the access key of instances of superclasses (in case of the inheritance hierarchy) can only be derived from the access key of the entity. This ensures that the access to the instances which are not related to the entity will never occur. Furthermore, in the case of the partial authorization, the request for indirect access will fail because the checks in Steps 1 and 2 are not satisfied, .

## 4.7 Object Restructuring

In an OODB system, objects or relationships might be deleted, added, or modified. In this section, we consider the impact of these changes on our authorization system.

## 4.7.1 Deletion of Objects

Objects in OODB systems can be deleted indirectly by altering database schema or directly by using the delete privileges. For the indirect deletion, all objects of database system has to be reorganized according to the new schema. For direct deletion, we consider three possibilities: deletion from a leaf node, deletion from an intermediate node, and deletion of a relationship. Deletion of an object from a leaf node requires the authorization-instances corresponding to the deleted object to be deleted from AC. Deletion of an intermediate object which is a part of the composite object causes that the descendants of the deleted object become the descendants of the parent of the deleted object. This requires the generation of a new SIFF function that satisfies Equation (2) of Section 4.5.1 and replaces the old SIFF function. If the deleted object is an instance of a subclass in the inheritance hierarchy, it is logical to delete all objects of the lower subclasses of the deleted object. Otherwise the objects of the subclass of the deleted object become the objects of the superclass of the deleted object. New SIFF hash function which satisfy Equation (2) of Section 4.5.1 must be produced for all objects of the superclasses of the deleted object and replace the old ones. In both cases, it is also necessary that the authorization-instances which correspond to the deleted object have to be deleted from AC.

In an OODB system, there are three types of relationship: (i) *aggregation* relationship; (ii) *generalization* (or *is-a*) relationship, and (iii) *association* relationship such as *teaches*, *is-taught-by*, *supplies*, *is-supplied-by*, etc. [174]. Note that the modification of the data model may cause deletion of relationships *aggregation* and *generalization.* The deletion of the *association* relationship may occur if the object is not associated with any objects.

Deletion of a *aggregation* relation may affect the composite object in two ways. First, the deletion causes that a part of the component has been removed. In this case, AC must be updated if the deleted part does not exist in the database schema any more. Otherwise, due to the changes in the structure of the deleted part, new SIFF functions for all objects of the deleted part must be regenerated. AC is left intact. Second, the deletion may cause the changes in the hierarchy of ancestors. In this case, new SIFF functions for all the objects of the descendants must be reproduced.

The deletion of a *generalization* relationship may affect the hierarchy of objects in two ways. First, deletion causes the superclasses of the low-level classes to change. This requires that new SIFF functions for all objects of the superclasses be reproduced. Second, the deletion causes that the hierarchy of the object is removed. Hence the authorization-instance of the deleted object must be deleted from AC. If the changes affects both the subclasses and the superclasses, new SIFF for objects of the associated superclasses must be reproduced. AC is left intact. Finally if a relationship with one or more objects is removed. New SIFF functions with one less collision must be selected. SIFF functions are replaced by the new ones.

### 4.7.2 Addition of Objects

An object might be added to the database as an instance of an existing class, or as a new object of a new class. In the first case (a new object of the old class), it is sufficient to complete the process described in Section 4.5.1. In the second case (a new object of the new class), we can distinguish the following three possibilities: (i) a new class is added to a leaf, (ii) a new class is added to an intermediate level, and (iii) a new relationship is created.

If a class is added as a new leaf, then a process similar to the one described in Section 4.5.1 must be completed for all objects of the new class. Moreover, since the subclasses of the superclasses have changed, new SIFF functions for all object instances of superclasses must be regenerated.

If a class is added to an intermediate node, first, the process described in Section 4.5.1 must be completed for objects of the new classes. Next, new SIFF functions must be regenerated (see Step 2 of Phase 2 of Section 4.5.1).

In the case of the addition of new relationships, a new SIFF function as described in Step 2 of Phase 2 of Section 4.5.1, must be regenerated for all objects of its descendants or superclasses.

## 4.8 Grouping and Group Updating

When users have the grant authorization (specified by *SPONSORSHIP*), they can create user groups and become the sponsors of them. They can give the privileges to the members of the group by running the *grant command* (see Section 4.5.2). Members of a group with the grant option (i.e., if *SPONSORSHIP* is on) can propagate privileges to other users. They have a user group hierarchy similar to the one shown in Figure 4.2. An important issue in the group hierarchy is the group updating. We can distinguish three possible cases of group modification:

1. a member of a group, who is the sponsor of the group, is deleted,
2. a new user or a group is added, and
3. a member of the group (or the sponsor of the group) is replaced by another one.

The impact of above mentioned modifications on the group organization and the updates they necessitate, are discussed below.

### 4.8.1 Deletion of Memberships

Deletion of memberships, in a group structure, is done by revoking the users authorizations by the sponsor of the group that (s)he is a member of.

The deleted user is just a member of the group. It is required that the associated authorization-instance is removed from AC, and a new SIFF function, with one less collision is selected. The new SIFF function replaces the old one in the authorization-instance associated with users in the MEMBER-LIST of the sponsor. The MEMBER-LIST must be updated too.

The deleted user is the sponsor of a group. In this case, the authorization-instance of the user and entries associated with the users (all members of the group) must be deleted from AC. Note that even if the entries associated with the users who are granted access by the deleted sponsor are not deleted, the access to the entity by these users will be denied immediately after the deletion of the sponsor. The same process, described before, must be done for the group in which the deleted user is a member (a new SIFF function for the rest members of the group must be regenerated).

### 4.8.2 Addition of New Memberships

A user who has the sponsorship privilege can grant his/her privileges to other users who can be either an individual or a group sponsor. In any case, it is required that the process described in Section 4.5.2 be completed.

### 4.8.3 Replacing

A member in a group can be replaced by a new member, or the login password of a member in the group can be changed. The member can be the sponsor of a group or a normal user. It is required that a new password for the member is selected and then the associated authorization-instance of the member is updated. A new SIFF for the group which the user is a member, is regenerated. If the replaced user was the sponsor of the group, a new SIFF function would also be regenerated for the group.

## 4.9 Security of the Authorization System

The authorization system is considered secure if it is computationally infeasible for an insider/outsider to gain unauthorized access.

**Proposition 4.1** *Assume that the tamper-proof module (TM) is secure and is run by DBMS only, and the computational power of an outsider is polynomially bounded. If the authorization system can be broken, then either the SIFF scheme, or the pseudo-random functions, or the user authentication scheme, or the cryptosystem used for encryption is insecure.*

Sketch of the Proof:

The proposed authorization system is secure if it is computationally difficult or simply "impossible" for an intruder to discover necessary information required for request validity check in Step 1 of Phase 2 in Section 4.6 (this information is secure).

More precisely, an intruder can gain the access to a protected entity in the system if (s)he can provide the required secret information. The intruder has the following possible options for the attack.

1. The intruder generates valid access key and an associated SIFF function, and an access privilege key for entity $j$, $K_j^P, h_j^P$, and $K^{op}$. This means that the intruder is able to predicate the output of the pseudo-random function and to find collisions for the SIFF function. This contradicts the assumption that the pseudo-random function and SIFF are secure.

2. The intruder can guess or disclose the login password of the owner or one of the grantees of entity $j$, say $U_l$, so the intruder can compute $n_{l,j} = f_{PS_l}(U_l \oplus E_j)$, and access the authorization-instance associated with $U_l$. This means that the user authentication system is insecure.

3. The intruder modifies his/her own privileges or someone else's. For example, the intruder modifies a partial authorization to a full authorization, changes ownership, etc. If this happens, the ability of the intruder is equivalent to breaking the cryptosystem which is used for DAC-INFO encryption. This contradicts our assumptions.

4. The inside intruder who is authorized to access a component of the object hierarchy, accesses high-level objects. This means that the intruder has succeeded to invert the SIFF function which contradicts to the one-wayness of the hash function.

## 4.10 Complexity of The System

As discussed before, there are two hierarchies in an O-O data model: *inheritance* and *composite hierarchy.* DBMS evaluates the authorizations along the object hierarchies. The most efficient way of the evaluation is to employ the proposed hierarchical access control. There are two different approaches to the solution of hierarchical access control problem. The first is based on RSA cryptosystem. The second uses one-way hash functions.

In 1982, Akl and Taylor [5] were the first who proposed a solution to the *hierarchical access control problem.* Their solution is based on the RSA cryptosystem [171]. There are several problems with this scheme. The scheme can work only with rigid hierarchical structures and cannot be used in OODB systems where the database schema may evolve. Each node stores two integers. First integer is a prime assigned to the node and the second integer is a product of primes associated with other nodes that are not descendants of the given node. The average length of the second component is large and hence expensive in terms of the storage. Moreover, the entire system must be predefined by the trusted central authority, and there is no way to expand or modify according to changes of the hierarchy. Some other solutions to overcome these problems were proposed in [48, 104]. A common drawback of these solutions is that they are based on the difficulty of breaking the RSA cryptosystem, and make heavy use of the underlying algebraic properties of the crypto-function.

The SIFF solution has several attractive features. The SIFF construction is based on the assumption of the existence of a one-way function. So the SIFF can be based on fast hashing algorithms such as MD4 ( MD5 or HAVAL) instead of very slow RSA system. Each node in the hierarchical structure needs to keep only one key of the length $n$ ($n$ = 128 bits), and hence the required storage is low. Moreover, expansion of SIFF according to the change of the hierarchy is straightforward and easy.

Let us give a brief comparison of the time and space complexity of the SIFF construction with the RSA one. Let $k$ be the number of objects in the hierarchical structure. Let $n$ be the length of keys. Assume that MD5 and polynomials of degree $k$ over finite fields $GF(2^n)$ ($n = 128$) are used to construct the SIFF hash function. The system based on the SIFF requires $O(\log k)$ modular multiplications of 128 bits long for key derivation (see Appendix B). Note that because computation time of the pseudo-random function and MD5 is negligible, it is ignored. If the RSA approach is used for authorization derivation and generation, the time complexity of the system will be $O((k+1)\log k)$ modular multiplications of $n$ bit long integer; $n$ must be at least 512 bits long, that is four times more. If the computation time of integers associated with objects is added up, then the time complexity will be much higher. For example, if a typical size of objects in the hierarchical structure is $k = 2^{10}$ then RSA would consume 1025.10 = 10250 modular multiplications of 512 bits while SIFF would take 10 modular multiplications of 128 bits.

Let $m$ be the entire number of objects in the database. Each object in the proposed system holds two keys and a hash function. Hence, the proposed SIFF approach requires $O(3mn)$ space in total. Since $n$ bit strings are compressed by MD5, then it is sufficient that the length $n$ be chosen close to 128 bits long. In the RSA approach, each object holds a key. However the public parameters such as integers must be stored too, then the total space is $O(3nm \log k)$. Note that $n$ must be at least 512.

Finally, in order to increase the efficiency of the system and to benefit from the order of access privileges, the lower access privilege can be inferred from higher access privileges without the necessity of their storage. The access privilege keys $(K^{rd}, K^{r}, \ldots, K^{c})$ can be chosen such that lower keys are computable from higher keys. This results in a shorter list of privileges. Also the SIFF function has a smaller number of collisions. For example, Equation (1) in Section 4.5.1 can be simplified as:

$$h_i^P(n_{j,i}\|K^w) = h_i^P(n_{j,i}\|K^c) = h_i^P(n_{j,i}\|K^d) = K_i^P.$$

## 4.11 Conclusion and Remarks

This chapter proposes a cryptographic mechanism for discretionary access controls in OODB systems. The mechanism is based on unique and secure access keys for each entity (object or class). Owners and user groups are identified by their unique passwords. Pseudo-random functions and SIFF are applied in such a way that access keys can be derived by the objects which have relationship with or by the user who are members of the group. We use an authorization class (AC) to store security information, the AC information is used during query processing to evaluate access request and enforce the security policy. The security of the system is based on the difficulty of predicting the output of pseudo-random functions and finding extra collisions for SIFF functions, both of which are known to be computationally difficult.

Object-instances based authorization system presents a finer granularity than class-based authorization system and enables the control to be imposed for individual objects. Note that as the numbers of users and objects in the system grow, the number of instances in the Authorization Class AC will increase and the security enforcement and manipulation of the object structure become resource expensive. To alleviate this problem, view mechanism can be used to define views which contain objects with same owners or grantees. Views can be applied as the units of authorization in the system.

# Chapter 5

# An Authorization Model Based on Views

## 5.1 Introduction

A *view* or a *virtual class* can be used for protection by allowing subsets of data to be seen/manipulated by users with required privileges. Views can be used to provide the desired level of granularity if a powerful enough query language is used for their definition. They can also provide an object content-dependent authorization.

Several authorization models for OODB systems, which support discretionary or mandatory access control, have been defined [17, 78, 80, 114, 123, 130, 144, 149, 169, 160, 210](see Chapter 3 for detailed discussion). However, none of them support content-based access control on instances of a class. Recently, several authorization models based on methods, which support content-based authorization, have been proposed [4, 32]. This approach has the drawback that method specification depends on authorizations. Therefore a change of the authorizations requires a change in the specification of the methods. Bertino and Weigand [30] used a constraint language to provide content-dependent authorizations for the authorization model [169]. The main problem of this solution is how to efficiently evaluate conditions associated with authorizations in the model implementation. In particular, enforcing the conditions expressed in the authorizations by filtering the data prior to user access requires a double access to the object (one to evaluate the conditions and the other to evaluate the user query). A way to provide content-based authorizations is to define views containing conditions on the values of specific instance-variables of the classes. In relational database systems, queries are modified by adding the conditions expressed in the authorization to the user query (this is known as the query modification mechanism). The view based authorization ensures that the protection requirements are satisfied. At the same time the access control is not overloaded. Moreover different levels of granularity (such as class-based, object-based, method-based, and instance-variable based authorizations) can be provided by suitable view definitions which can be adjusted to the user needs.

The purpose of this chapter is to discuss views in OODB systems and their application to enforce discretionary access control policies. In particular, we are going to address the problem of time and content-dependent authorizations. To improve the flexibility of the view model and consequently guarantee the operational security of the system, *parameterized views* are introduced. Such views extend the view model given by Bertino [31] with parameters which are fixed at the time of a view evaluation. Four types of inheritance between base classes (or views) and derived views are considered: *specialization inheritance, constraint inheritance, strict constraint inheritance,* and *proper specialization inheritance.*

This chapter is organized as follows. Section 5.2 provides an overview of the view model proposed by Bertino in [31]. Section 5.3 discusses how view hierarchies can be inferred from class hierarchies and view definitions. In Section 5.4, the *access view* as a tool for the access control is introduced and security requirements for the discretionary access control are given. Section 5.5 formulates implicit authorization rules for each domain of authorization (user, view, privilege). Section 5.6 presents the way how the validity of an access request can be checked. Finally, Section 5.7 summarizes the chapter.

## 5.2 View Model

In the relational data model, a view is defined as a *virtual relation* derived by a query on one or more stored relations. The relational operations *join, select,* and *project* may be used to define a view. Views can be used in (almost) any query where relations can also be applied. Furthermore, privileges may be granted and revoked on views as on ordinary stored relations. This feature allows the content-based authorization on stored relations. Therefore, views can be used for both: *data protection* and *user convenience*.

Views in an OODB system model have been considered in a number of papers [1, 31, 63, 107]. However, there is no consensus about the form of the view model for OODB systems. Until such standard becomes available, the choice of a view model is to some extend arbitrary. Here, we consider the view model which was developed by Bertino [31] and describe how security properties might be incorporated into such model. Note that while the authorization model is developed in the context of the view model proposed in [31], the essential ideas are widely applicable.

In [31], Bertino provides a view model for OODB systems that extends typical view models of relational databases. In Bertino's model, as in relational systems, a view is defined by a query on one or more classes, called *base class(es)*. To make the view model suitable for usage in OODB systems, several additional features are provided. Views with additional properties that are not derived from the base classes are introduced (**additional-properties**). The view model is not restricted to a structural model but also has behavioral features, i.e., a view may have methods (**methods**). A method can be derived from the base class(es), with the same or different name, or it can be created for the view. In order to re-use existing view definitions in the specification of a new view, a new view may be defined as a subview of other views (these views are called **superviews**). For detailed discussion of the view model and its usage for user convenience, the reader is referred to the Section 2.4 of Chapter 2.

## 5.3 Inferring the View Hierarchy

From Bertino's view model [31], two types of view hierarchies may be constructed. The first is the *composite (or is-part-of) hierarchy*. An instance variable specification (provided with the keyword **additional-properties**) consists of the variable name and the domain where the domain can be a class or a view. Then an instance of a view may be the aggregation of a set of objects, each of which belongs to some class or view. Such an aggregate view is sometime called a *composite object*, and we call it a *composite view*.

The second view hierarchy is the *inheritance (or is-a) hierarchy*. A view hierarchy can be defined for a view (using **superviews**), as well as being derived from the view definitions which is called implicit (inferring) view hierarchy. In this section, we develop and formalize the idea of derived view hierarchies.

As pointed out in [1], a view can be constructed in two distinct ways: *top-down* or *bottom-up*. In the *top-down* approach, large classes are divided into smaller ones via *specialization* (a similar operation in relational systems is the selection operation). In the *bottom-up* approach, small classes are combined to form larger classes via *generalization* (the analogous operation in relational systems is the union operation). The following two examples illustrate the construction of a view using the *top-down* or *bottom-up* approach. Note that for examples presented in this chapter, the university database schema shown in the Section 2.2.7 of Chapter 2 is used.

**Example 5.1** *Construct a view which returns the information of postgraduate students who are from Australia. It is required that name, idno, subject, Graduate_Date, and country are visible.* □

One possible declaration of the view is as follows:

> **create-view** Australian_Graduate_Students
> **select** F.idno, F.name, F.Subject, F.Graduate_Date, F.country
> **from** F:FOREIGN
> **where** F.status = "graduate" **and** F.country="Australia";

This example shows how to construct a view using the *top-down* approach. It illustrates the application of view mechanism that restricts the set of visible object instances of a class (this is analogous to selection operation in the relation systems). The view *Australian_Graduate_Students* contains a subset of object instances of the class *FOREIGN*. It also illustrates the possible usage of view mechanism that restricts properties which are visible. This is similar to the projection operation in the relational systems.

**Example 5.2** *Define three separate views such that the first two views contain software supporters and programmers, respectively and the third one contains both.* □

One possible way to define such views is as follows:

**create-view** Software_Supporters
**select** E
**from** E: EMPLOYEE
**where** E.profession= "software_supporter";

**create-view** Programmers
**select** E
**from** E: EMPLOYEE
**where** E.profession=
"programmer";

The views *Software_Supporters* and *Programmers* contain object instances of the class *EMPLOYEE* that are software supporters and programmers, respectively. They are examples of the *top-down* approach without projection operation. The next view contains object instances of both views. It is an example of the *bottom-up* approach.

**create-view** Technical_Staff
**select** P, S
**from** P:Programmers, S:Software_Supporters;

In general, a view defined using *top-down* or *bottom-up* approach can be categorized as follows.

**T. Top-down Approach.** Suppose that a view $V$ is derived from a class (or a view) $C$. Then the view $V$:

*T1. inherits all properties of $C$, and the set of instances of $V$ is a subset of instances of $C$;*

*T2. has more properties and/or methods than $C$;*

*T3. has fewer properties and more methods (or vice versa) than $C$; and*

*T4. has fewer properties and methods than $C$.*

**B. Bottom-up Approach.** Suppose that the view $V$ is defined over classes (or views) $C_1, \ldots, C_n$. Then one of the following cases may happen.

*B1. $V$ inherits all common properties and methods of the base classes (or views). It contains all common instances of $C_1, \ldots, C_k$, and selected instances of $C_{k+1}, \ldots, C_n$. Note that if $C_1, \ldots, C_k$ are not required to have the same properties, then the view $V$ may have fewer properties and/or methods than the base classes (or views).*

*B2. V inherits all properties and methods of the base classes (or views) if $C_1, \ldots, C_k$ have the same properties. V includes all instances of $C_1, \ldots, C_k$, and selected instances of $C_{k+1}, \ldots, C_n$.*

*B3. V contains all instances of $C_1, \ldots, C_k$, and selected instances of $C_{k+1}, \ldots, C_n$. V has more properties and/or methods than base classes (or views).*

Based on the above discussion, we can distinguish four types of inheritance relationship among classes (or views) and derived views.

*1.* **Specialization inheritance.**

*Views inherit all the properties of the base classes (or views), and they also have some additional properties and/or methods. Examples are cases T2, and B3. In the case T2, the view V is a specialization of C, or in other words V is a subclass (or subview) of C. In the case B3, each $C_i$ is a subview (or specialization) of V for $1 \leq i \leq k$. Moreover, if a class (or view) D is a superclass (superview) of base class(es) (or views) then D is also a superclass (superview) of V.*

*2.* **Constraint inheritance.** *Views consist of all instances of base classes (or views) that satisfy given constraints (cases T1, and B2). In other words, every instance of the view is also an instance of the base classes (or vice versa). Hence, the view is a subview of base class (or vice versa). For example in case T1, V is a subview of C as every instance of V is an instance of C. In case B2, $C_i$ ($1 \leq i \leq k$) is a subview of V since every instance $C_i$ ($1 \leq i \leq k$) is an instance of V.*

*3.* **Strict constraint inheritance.** *Views not only consist of all instances of the base classes (or view) that satisfy given constraint, but there are fewer properties and methods than in the base classes (or views). In other words, the view not only filters out the instances of the base classes (or views) but also projects the properties (see cases T4 and B1). V is a strict subview of C (case T4). In the case B1, V is a strict subview of $C_i$ for $1 \leq i \leq k$.*

*4.* **Proper specialization inheritance.** *Derived views filter out the instances of the base classes and project the properties. They also contain new additional properties or methods (see the case T3). Views consist of* *all instances of the base classes (or views) that satisfy the given constraint. They do not inherit all properties or methods of the base classes (this contradicts the principle that a subclass must inherit all properties and methods of superclasses). They have new additional properties or methods. Therefore, derived views cannot be specialization of the base classes because they do not inherit all properties of base classes. Moreover, they cannot be strict subviews of base classes since they have additional properties and/or methods. Hence, the view $V$ is a proper specialization of $C$.*

## 5.4 Authorization System

As described in the previous section, view definitions use predicates on the values of properties of a class, and combine the object values of several classes to access objects that meet the specified conditions. If authorization is granted on a view, the access is restricted to the object instances that are filtered by the view. Hence, views provide a powerful and flexible mechanism for authorization based on database contents. They can be used to provide the desired levels of authorization granularity such as class, object, instance-variable, or method-based. Examples 5.3-5.6 show how a view model can be applied to provide different levels of granularity.

**Example 5.3** *Define a view Students_Info which contains all information related to all students (class-based authorization).*□

```
create-view Students_Info
select S
from S: STUDENT;
```

**Example 5.4** *Define a view which contains information related to a particular student, say student with id number 9272860 (object-based authorization).* □

```
create-view Student_9272860
select S
from S: STUDENT
where S.idno=9272860;
```

**Example 5.5** *Define a view which contains name, subject, and Start_Date related to all students (instance-variable based authorization).* □

**create-view** Students_Name_Subject
**select** S.name, S.subject, S.Start_Date
**from** S: STUDENT;

**Example 5.6** *Define a view which contains the method associated with class STUDENT, Comp-GAP(). The method outputs the greatest average point (method-based authorization).* □

The method *Comp-GAP()* computes the associated value with the instance variable *GAP* related to each student. Then, it is possible to define a view *Computing_GAP* as follows:

**create-view** Computing_GAP
**methods** Comp-GAP()
**from** STUDENT;

The view *Computing_GAP* can access only the method *Comp-GAP()*. If a user has *execute* right on the view *Computing_GAP,* (s)he can execute the method associated with the view. Therefore, a view model provides the facility which creates different interfaces for a class and views can be used as units of authorizations.

Furthermore, views provide a useful mechanism for efficient access control with less storage in an authorization system based on object instances. This is true because view mechanism provides a facility to group objects which share the same access control list. Hence, the maintenance of authorizations becomes easier.

The access rule in a database with views, can be formally defined as follows.

**Definition 5.1** *An access rule is a triple* $(u, v, r)$ *where*
$u \in U$, *U is the set of users/roles in the system;*
$v \in V$, *V is the set of views defined in the system; and*
$r \in R$, *R is the set of privileges.* □

The triple $(u, v, r)$ means that a user/role $u$ has an access right $r$ on a view $v$.

Before we discuss the use of views in object-oriented databases to enforce the discretionary access control, we need to answer the following questions: "who is a user?", "what is the unit of authorization?", "what is the set of privileges?", "who can grant/revoke authorizations?", "which security requirements must be considered?", "how to enforce the security requirements?", etc.

To answer these and other questions, we first introduce the notion of *access view*. Later, we use access views to give answers to the above questions. We also consider the impact of the hierarchical structure of each domain of access rule (user, view, privilege) on the authorization system.

For the rest of the chapter, we assume that the proposed authorization system uses *positive authorization* policy which requires explicit specification of allowable access privileges. The lack of authorization for a user to a view implies no access to the view. We denote by $u$ a user/role identity, $v$ a view, $r$ a privilege, $t$ an authorization type, $op$ an access privilege (access mode), $V$ the set of views, $U$ the set of users, and $R$ the set of privileges.

## 5.4.1 Access Views

In order to enforce security requirements and to protect an object against an unauthorized access, the authorization system has to know the exact users privileges. This can be accomplished by the creation of access views. An *access view* is an extended view that in addition to the view specification has an authorization list for the discretionary access control.

**Definition 5.2** *An access view is a view that includes the view specification plus a discretionary access control list. The format of the access view is as follows:*

> **access-view** *Viewname[parameters]*
> – *view specification as described in Section 2.4 of Chapter 2*
> **Auth-Spec** *authorization_information*

*Viewname is the name of the access view, and must be different from the names of other access views and classes.* **Auth-Spec** *specifies authorization information which is an aggregation of the* **ownid** *and* **ACL**. **ownid** *denotes the identity of the view owner and* **ACL** *indicates the access control list associated with the view.* □

To increase the flexibility of the view model and to provide possibility to define time-dependent authorizations, we allow *parameterized access view*. Parameters are bounded to the actual values at the time when the view is evaluated. Suppose that we want every employee to be able to access their own personal information. One way to do so is to define a view (see Example 5.4) for each employee. When the number of employees are large, this implementation of access control is inefficient. A better way is first to define a view and later to create a role (or a group) which specifies the collection of all employees who are authorized to access the view.

**Example 5.7** *Let a user $u_1$ create a view Personal_Info which consists of idno, name, address, age, and spouse. The user $u_1$ gives privileges (t, read) to a role (or a group) Employee.* □

> **access-view** Personal_Info(Current_idno)
> **select** P.idno, P.name, P.address, P.age, P.spouse
> **from** P:PERSON*
> **where** P.idno = Current_idno
> **Auth-Spec.ownid** $u_1$
> **Auth-Spec.ACL** $\{(Employee, (t, read))\}$;

The instance of the above view is based on the *Current_idno* which gives the current identity number of an employee who uses it. If employees are allowed to access their information only during working hours (9 am to 5 pm), the following modification of the above view is sufficient. *Time* gives the current system time.

> **access-view** Personal_Info(Current_idno, Time)
> **select** P.idno, P.name, P.address, P.age, P.spouse
> **from** P:PERSON*
> **where** P.idno = Current_idno **and** Time $\geq 9$ **and** Time $\leq 17$
> **Auth-Spec.ownid** $u_1$
> **Auth-Spec.ACL** $\{(Employee, (t, read))\}$;

The authorization specifications can be read by users who have access to the view but they can only be modified by the owner of the view, or users who have the grant or revoke authorization.

In general, the meta-data of a database system in the view model consist of two classes of data. Conceptual data are mapped to the stored data. In the case of object-oriented databases, these are called classes which are mapped to the stored objects. Each object is an instance of a class. Views are declared (or created) based on the classes or pre-defined views. Therefore, there can be two classes of users: those who are allowed to access/manipulate classes, and those who are authorized to access objects through views only.

**Authorization Rule 1** *There are two classes of users: security officers and normal users. Security officers supervise the entire database activities and are responsible for creating views based on classes. Furthermore, they create classes and perform schema changes. Normal users can only access data through access views.* □

Users may want to create subviews or superviews. They can create new views based on pre-defined views if they have *create-view* or *ownership* privileges on the base views.

**Authorization Rule 2** *A user can create an access view over other views if the user owns the views, or has the create-view authorization on the base views.* □

When a user creates a view, (s)he becomes its owner. This is specified by the **ownid**. The owner has the full authorization to the view and determines who can access it and how.

We assume that there are two sets of privileges: *view privileges,* and *view-instance privileges. View privileges* are used to manipulate the view definitions and typically are: *delete-view, create-view, modify-view, grant,* and *revoke. View-instance privileges* are used to access and manipulate the instances of a view. A typical collection of privileges is: *read-definition, read, write,* and *delete* (see Section 5.5.2).

**Authorization Rule 3** *The creator of a view is its owner. The owner of the view specifies other user privileges for all view privileges but the owner can grant other users only view-instance privileges that are authorized on based views.* □

**Authorization Rule 4** *The ownership is transferable to other users. However, at any time a view has a unique owner.* □

The authorization system must check if users have suitable authorizations to perform the requested operations on views. Hence for each view, a unique **ACL** is defined. It lists who and how users can access the view.

**Definition 5.3** *The access control list (ACL) of a view $v$ is a list of pairs $(u, r)$ where $u$ is the identity of a user(or role) and $r$ is a privilege granted to the view $v$.* □

Views (consequently access views) can have additional properties whose types can be pre-defined views or classes (composite views). Also we may have a query whose scope of access can be a view (class) and its subviews (subclasses). In such cases, it is essential to determine whether an authorization to the root of a view is to be extended to all its constituents.

Rabitti, Bertino, Kim, and Woelk [169] distinguished two types of authorization for composite objects and class-hierarchy object access. They called them *full* and *partial* authorizations.

The *full authorization* implies the same privileges on each component of the composite object or on each class of the hierarchy. However, the *partial authorization* does not extend privileges to the descendants of composite (or class-hierarchy) objects. We here employ the same concept.

**Definition 5.4** *A privilege $r$ has a form of $(t, op)$ where $t$ denotes the type of authorization, and $op$ indicates the access privilege such as read-definition, read, write, delete, execute, create-view, modify-view, delete-view, grant, and revoke.* □

An access rule $(u, v, (t, op))$ states that a user $u$ is authorized to execute an access privilege $op$ of type $t$ on a view $v$.

There are three types of hierarchy: *user-role, view,* and *composite hierarchy.* For every hierarchy, it is required to determine whether privileges to the root node of hierarchy should be extended to its descendants. Consider the authorization type $t$ and how it can be extended. It is reasonable to assume that *full $(F)$* or *partial $(P)$,* can be applied to all three types of hierarchy. This approach is simple but is not flexible. Alternatively, we can determine authorization type $t$ for each hierarchy independently. In this case, the authorization type $t$ will become a triple such that each element determines the associated authorization type for the three possible hierarchies. We have chosen the second approach.

**Definition 5.5** *An authorization type $t$ is a triple $(t_1, t_2, t_3)$ where $t_1$ indicates the type of authorization for user-role hierarchy, $t_2$ - for view-hierarchy, and $t_3$ - for composite hierarchy. $t_i$ $(i = 1, 2, 3)$ can be either $F$ (full) or $P$ (partial).* □

We assume that partial $(P)$ authorization for each type of hierarchy will become effective unless full $(F)$ authorization is explicitly specified, i.e., the partial authorization is the default option.

**Example 5.8** *Let a user $u_1$ create a view Adult. Assume that $u_1$ gives the following authorizations ((P, F, F), read), write, and ((, F, P), read) to users $u_2, u_3$, and $u_4$, respectively.* □

The access view *Adult* may look like the following:

> **access-view** Adult
> **select** P
> **from** P:PERSON*
> **where** P.age $\geq$ 21 **and** P.age $\leq$ 95
> **Auth-Spec.ownid** $u_1$
> **Auth-Spec.ACL** { *($u_2$, ((P, F, F), read)), ($u_3$, ((P, P, P), write)),*
> *($u_4$,((P, F, P),read))* };

The view *Adult* is declared on the composite class PERSON (the components of the class PERSON are classes NAME and ADDRESS and the component of the class ADDRESS is the class TEL). Therefore, the *full read* authorization for the user $u_2$ implies that $u_2$ can not only read the view *Adult* and consequently the class PERSON but can also read the components of the class PERSON (ADDRESS, NAME, and TEL). The users $u_3$ and $u_4$ can only access the view *Adult* and consequently the class PERSON. Moreover as indicated by "*" in the view definition, the view population might be all hierarchical instances of the class PERSON. So the full authorization implies the same right on all instances of the view subclasses (or subviews). As the result, the view population includes all hierarchical instances. Otherwise, it would just include the object instances of the root node of the hierarchy only. For instance, users $u_2$ and $u_4$ can read all constrained instances of the class PERSON including students, employees, suppliers, etc. However, the user $u_3$ can only access constrained instances of the class PERSON.

## 5.5 Implication Rules

In this section, we look at the impact of hierarchical structure (order) of domains of the access rule $(u, v, r)$ on the authorization system. We employ these hierarchical structures to provide deductive rules which improve the efficiency of the authorization system. By applying deductive rules, implicit authorizations can be derived from explicit ones .

### 5.5.1 Authorization Users

To reduce the number of explicit authorizations in the system, we can group users according to their roles. The roles are usually organized in some hierarchical structure [13, 169]. Therefore an explicit inclusion of the same authorization for more special (or higher-level) roles can be removed, because the higher-level roles will inherit authorization from those given to more general (or lower-level) roles. This is a result of the principles of generalization and specialization. In many situations, a natural hierarchy of roles exists (see Figure 5.1). A node of the graph

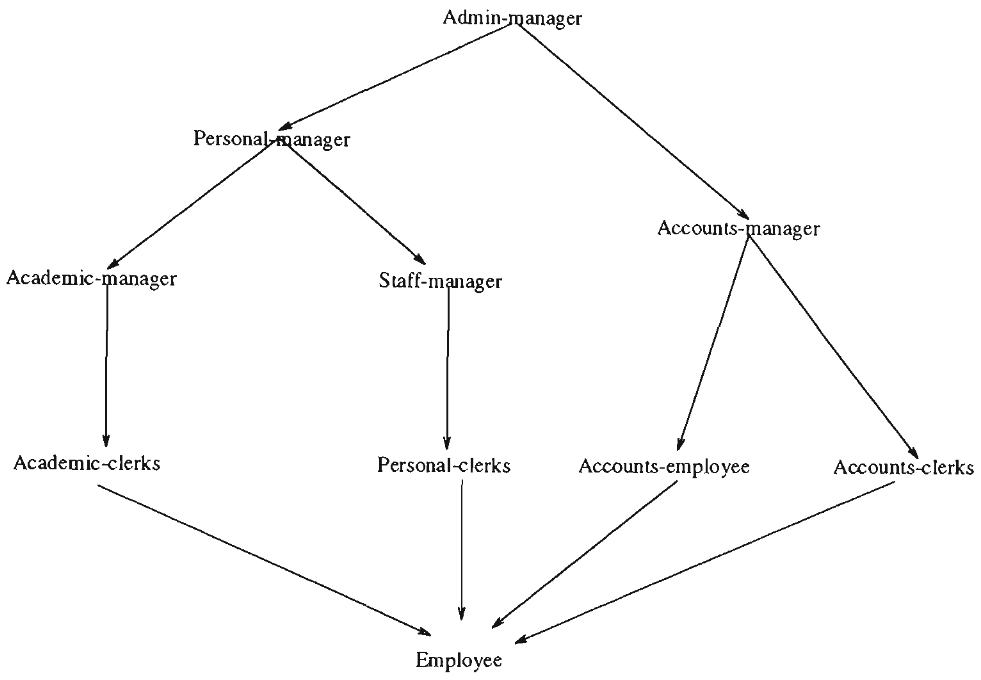


Figure 5.1: A sample of user role hierarchy.

represents a role, and a directed arc from one role to another indicates that the authorizations for the higher-level role subsume the authorizations for the lower-level role. For example, the privileges for the role *Personal-manager* subsume the privileges of the roles *Academic-manager, Staff-manager, Academic_clerks, Personal_clerks,* and *Employee* (see [169] for formal definition of role hierarchy). Consider Example 5.7, having an explicit authorization rule *(Employee, ((P, P, F), read))* implies that all members of the role *Employee* can read *Personal_Info.* It is clear that all higher-level roles must be allowed to read the view. If the explicit authorization is the only way to allow users to access data, then the *read* privilege must be replicated for all higher-level roles. The existence of many such authorizations may lead to inefficiency. Therefore, this is a good idea to allow higher-level roles to have all rights associated with lower-level roles. To derive implicit authorization for roles, we can use the following rule.

**Implication Rule 1** *An explicit authorization for a role results in implicit authorizations for all higher-level roles.* □

In our example from Figure 5.1, all higher-level roles, *Academic-clerks, Personal-clerks, Accounts-clerks, Accounts-employee, Academic_manager,* etc., have *read* privilege on the *Personal-Info* implicitly, by the above rule.

Because a role contains a set of users, it is possible to give privilege on a view $v$ to a role, rather than giving it individually to all members of the role.

**Definition 5.6** *The first element $u$ in access rule $(u, v, r)$ is either a user identity or a role identity. If the role identity is determined, all users of the role $u$ will have the privilege $r$ on the view $v$. Otherwise the specific user has the privilege.* □

Suppose the following view that contains information about casual employees payments is defined. The required information are *idno, name, rank, Hourly_Wage, Weekly_Hours,* and *Weekly_Wage.*

> **access-view** Casual_Payment(Current_idno)
> **select** C.idno, C.name, C.rank, C.Hourly_Wage, C.Weekly_Hours, Weekly_Wage
> **from** C:CASUAL
> **where** C.idno = Current_idno
> **methods(numeric** Weekly_Wage())
> **Auth-Spec.ownid** $u_1$
> **Auth-Spec.ACL** $\{(Account\_employee, ((P, P, P), read))\}$;

where the method *Weekly_Wage()* computes the weekly payment of an employee.

*read* privilege for the role *Account_employee* means that members of roles *Account_employee, Account_manager,* and *Admin_manager* can access the information (see Figure 5.1). Suppose we want that employees be allowed to access their own information. If we grant *read* privilege to the role *Employee* (in fact the tuple *(Employee, ((P,P,P),read))* is added to **Auth-Spec.ACL**), then all higher-level roles can also access it by Rule 1. The only way to solve this is to grant *read* privilege to all individual employees. If the negative *read* authorization for authorized roles is specified, conflicting authorizations may arise.

However, in our model, the role identity is given, and the first element of the authorization type $t$ will determine the type of authorization associated to the role hierarchy. Therefore, by assigning *full read* authorization to the role *Account_employee ((Account_employee, ((F, P, P), read)))*, the same right will be implied for the role *Employee.*

**Implication Rule 2** *Explicit full authorization for a role results in an implicit authorization for all associated lower-level roles.* □

For example, if the *Academic_manager* has *full read* privilege on a view $v$ ($v$ can be considered to contain research area and grants associated with the academic staff) then, all descendant roles of the *Academic_manager* such as *Academic_clerks,* and academic employees will implicitly have the same authorization on $v$.

We assume that partial authorization right for roles will become effective unless full authorization right is explicitly specified.

## 5.5.2 Authorization Access Privileges

As observed, a privilege $r$ in our authorization system is of the form $(t, op)$ where $t$ denotes the authorization type, *full* or *partial,* and *op* denotes the access privilege. So we first define allowable access privileges (access modes) and then consider the effect of authorization type $t$ on the authorizations.

We assume that an OODB system provides two sets of system-defined methods (called access privileges): *view,* and *view-instance privileges.* A typical collection of privileges for view are *create-view, modify-view, delete-view, grant,* and *revoke.* A view-instance privileges are: *create, delete, write, execute, read,* and *read-definition.*

A *create-view* privilege is used to create views. A *modify-view* privilege is used to change the definition of a view. A *delete-view* privilege is used to delete the definition of a view. *grant* and *revoke* privileges are used to grant and revoke both the view privileges, and view-instance privileges to and from users, respectively. The *grant* and *revoke* authorizations cannot be propagated. This means that only the owner of a view is allowed to give the privileges *grant* and *revoke* to other users. *read* and *read-definition* privileges are used to read the instances of a view and read the definition of a view, respectively. An *execute* privilege is used to perform the methods associated with a view. In other words, the *execute* privilege can be considered as an *invoker* that can call methods associated with a view. *Write* and *create* privileges are used to modify and to create an instance of a view, respectively. A *delete* privilege is used to delete an object instance of a view.

**Definition 5.7** *The set of privileges includes read-definition, read, write, delete, execute, create-view, delete-view, modify-view, grant, revoke. We assume these privileges are partially ordered and:*

*create-view > read-definition*

*modify-view > delete-view > read-definition*

*grant > revoke*

*write > execute > read > read-definition*

*create > execute > read > read-definition*

*delete > read > read-definition.* □

This means that the holder of an access privilege of a higher order possesses privileges of the lower order. For instance *execute* implies that its holder (a user)r has both *read* and *read-definition* privileges because the user must be able to read the values and definitions associated with the parameters of a method in order to execute the method. To delete an object instance, a user must first access it and later remove it. This implies *read* and *read-definition* authorizations. The access right *modify-view* implies that the rights *delete-view,* and *read-definition* are also allowed.

Using the above order and the following rule, it is sufficient to specify explicit authorization only for the highest access privilege.

**Implication Rule 3** *If an authorization is given for an access privilege in the privilege hierarchy, then this implies the authorization of all privileges below it.* □

**Implication Rule 4** *The full authorization for a privilege implies the partial authorization for the same privilege.* □

### 5.5.3 Authorization Views

In Section 5.3, we considered four types of inheritance among views. They were: *constraint, strict constraint, specialization,* and *proper specialization inheritance.*

In general, if the base views contain more information than the derived views, it is reasonable to assume that a user $u$ has the privilege $r$ on the derived view $v$ whenever the user has the privilege $r$ on the base view $w$ *(constraint inheritance* and *proper specialization inheritance).*

However, in the case of the *specialization inheritance* and *proper specialization inheritance* to preserve the view privacy, a privilege on a superview does not imply the same privilege on the derived views because they contain more information than the base views. If the full authorization for the superview is indicated, the same authorization implies on the derived views as well. The following two rules deal with this point.

**Implication Rule 5** *If a view $v$ relates to a view $w$ via either constraint inheritance or strict constraint inheritance, then explicit authorizations on the view $w$ generates the same implicit right on the view $v$.* □

The user has the same right on *Software_Supporters* and *Programmers* views as the views relate to *Technical_Staff* via the *constraint inheritance* (see Example 5.2).

**Implication Rule 6** *Assume that two views $v$ and $w$ are related via (proper or) specialization inheritance, or composition. The explicit full authorization on the view $w$ results in the implicit authorization on the view $v$ with same collection of privileges.* □

For example, the role *Employee* has the full *read* authorization on the view *Personal_Info* (see Example 5.7). When the view is evaluated, the components of the class *PERSON (ADDRESS* and *NAME)* are authorized for retrieval too.

## 5.6 Access Control

In the proposed system, the user's access to the database is controlled by a set of *access views* (*AV*).

**Definition 5.8** *Let AV be the set of all access views. AV may only be accessed or manipulated by the authorization system via the following commands:*

**grant**$(u,v,r)$ *- adds the pair* $(u,r)$ *to the* **ACL** *of the view* $v$.

**revoke**$(u,v,r)$ *- removes the* $(u,r)$ *from the* **ACL** *of the view* $v$.

**Own***(v) - retrieves the owner's identity of the view v.*

**change-own***(v) - changes the owner of the view v.*

**Accesslist***(v) - retrieves the authorization list associated with the view v for every $v \in AV$.* □

The only possible way for all users (except security officers) to access data in the database is via $AV$.

**Definition 5.9** *An access request is a triple $(u, v, r)$ where $u \in U$ is a user who requires the access $r \in R$ to a view $v \in AV$ where $U$ is the set of users/roles, and $R$ is the set of privilege in the system.* □

**Authorization Rule 5** *An access request $(u,v,r)$, $v \in AV$, $u \in U$, $r \in R$ is valid, if the entry of the view v is in $AV$, and the pair $(u, r)$ exists in the* **ACL** *of the view v or can be derived by applying the implication rules 1-6.* □

An access request $(u, v, r)$ is valid if and only if the following function returns true.

```
function Access($u, v, r$)
begin
If $u$= Own($v$) $\bigvee$ $(u, r) \in$ Accesslist($v$)
    then return true
    else return(Implicit_Access($u, v, r$));
end

function Implicit_Access($u, v, r$))
begin

/* checking access request against user authorization and privilege
hierarchies */

If   $\forall\ u_j$ role of the user $u$ $\exists$ $(u_j, r) \in$ Accesslist($v$) $\bigvee$
     $\forall\ u_j$ descendant of the role of the user $u$
     $\exists$ $(u_j, r) \in$ Accesslist($v$) $\bigvee$
     $\forall u_j$ ancestor of the role of the user $u$
```

```
        $\exists\ (u_j, ((F, , ), op)) \in \mathbf{Accesslist}(v)\ \bigvee$
        $\forall\ op_j$ higher-level of the privilege $op\ \exists\ (u, (t, op_j)) \in \mathbf{Accesslist}(v)$
    then return true

    /* checking access request against view authorization hierarchy */

    else If $\exists w$ that $v$ is (strict) constraint inheritance of $w\ \bigwedge$
        $\exists (u, r) \in \mathbf{Accesslist}(w)$
        then return true

        else If $\exists w$ that $v$ is (proper) specialization inheritance of
           $w\ \bigwedge\ \exists (u, r) \in \mathbf{Accesslist}(w)\ \bigwedge t_2 = F$
        then return true
        else If $\exists w$ that $v$ is component of $w\ \bigwedge$
        $\exists (u, r) \in \mathbf{Accesslist}(w)\ \bigwedge t_3 = F$
        then return true
        else return false;
    end
```

Where $\forall$ stands for "for all", $\bigvee$ - OR logical operation, $\exists$ - exist, and $\bigwedge$ - AND logical operation.

## 5.7 Summary

The chapter discusses the use of views in OODB systems for the discretionary access control. Special attention has been placed on time and content-dependent authorizations. *Parameterized views* have been introduced to increase the flexibility of the view model. It has been shown how a view hierarchy can be inferred from views. The inheritance hierarchies among views have been discussed. We have also defined *access views* as the mechanism for access control. We have discussed discretionary security requirements for authorization systems. Rules for computing an implicit authorization from the explicit ones have been formulated.

The impact of three authorization dimensions, users, access privileges, and views on the access views has been examined. Finally, we have presented the form of a valid access request and how the validity of requests can be verified.

# Chapter 6

# A Multi-level View Model for Secure OODB Systems

## 6.1 Introduction

Several secure models for OODB systems are discussed in [33, 39, 114, 121, 122, 123, 143, 144, 149, 150, 160, 210, 215]. The majority of models consider *single-level objects* only. This means that for every object, a unique security level is assigned which applies to the entire object [33, 114, 149, 210]. This approach is attractive for its simplicity and its compatibility with the security kernel. Moreover the multi-level update problem [210] does not exist.

However in the real world, there are situations where it is necessary to classify instance variables of an object at different security levels. That is, the security model has to support *multi-level objects.* There are also models write a finer grain of classification - the security level is assigned to each instance variable of an object [122, 140, 160]. Unfortunately, these proposals require both a trusted enforcement mechanism on the object layer and a complex security kernel.

In order to maintain the security kernel compatibility and to overcome the difficulties with multi-level objects, some researchers proposed to design a schema which handles various security constraints [114, 149, 210]. For example, if we want the $GAP$ instance variable of the class $STUDENT$[1] to be secret, we need to create a class $STUDENT\_GAP$ with security level SECRET. $STUDENT\_GAP$ is a subclass of the class $STUDENT$ (see Figure 6.1). If the security level of

[1]The specification of STUDENT and EMPLOYEE have been shown in Figure 2.3 in Subsection 2.2.7 of Chapter 2

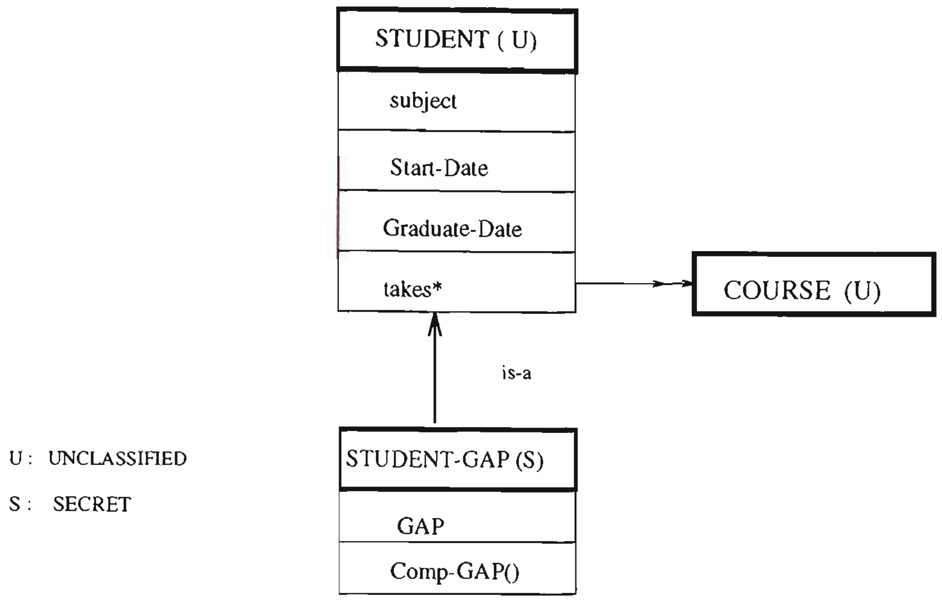


Figure 6.1: An example of the representation of simple constraint.

instance variable *address* depends on the value of instance variable *profession* of the class *EMPLOYEE* (the security level of *address* is secret if the employee is a chancellor or vice-chancellor, and otherwise is unclassified), then we shall create two classes *S_ADDRESS* and *U_ADDRESS* to be subclasses of *EMPLOYEE*. Each of them contains addresses related to secret or unclassified employees, respectively (see Figure 6.2).

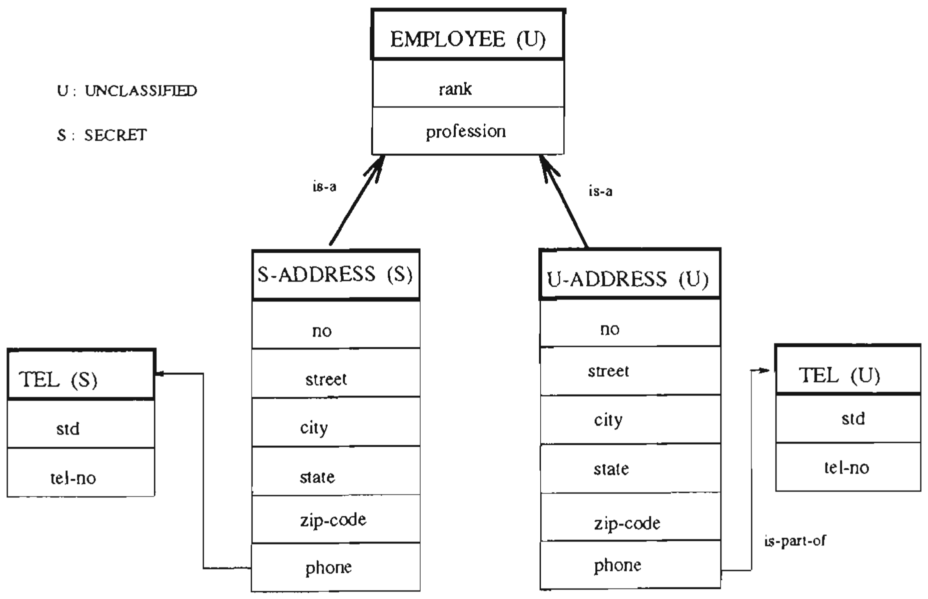


Figure 6.2: An example of the representation of content constraint.

There are several problems with this approach. If the value of an instance variable is changed dynamically, the schema evolution should then reflect the change. As object instances do not have to be at the same security level as their class, it may happen that there are object instances at levels higher than the level of the corresponding class. Therefore, certain object instances might end up in unexpected locations and be inaccessible to authorized users. For example, if a secret address object is created as an instance of *U_ADDRESS* by a secret user, no reference to it would appear in the instance variables in the *U_ADDRESS* object. As the result, secret subjects would fail to find it in the expected place under the secret subclass *S_ADDRESS*.

The view update problem disappears and views can almost freely be updated in an OODB system when a query language used for view definitions preserves object identities [190]. Recall that in relational databases, views containing the key of their (one) underlying base relation can be updated. We assume that the query language used for view definitions has the object preservation property. The advantages of the usage of view approach are as follows:

1. View definitions can be regarded as subclasses, or superclasses of the base classes (virtual subclasses or superclasses) [1, 21, 190]. Therefore, views provide the facility for a dynamic modification of the database schema but yet they retain their older versions. They also provide a tool to handle various security constraints.

2. Views may be defined on arbitrary sets of classes and other views with different security levels. These views are called *multi-level views.* So by defining a multi-level view for unclassified and secret users, the possibility of storing certain data in an unexpected location which is not accessible to authorized users is eliminated.

3. A view definition can also be regarded as a constraint relating derived data to other data (stored or derived). It can be used to restrict the user access to the data that they actually need. Thus the view approach allows to handle inference and aggregation problems or at least minimize difficulties associated with them.

4. View definitions are independent of the underlying data. If the database contents changes, it will not be necessary to reclassify the database because views will enforce the required classification rules. For example, if an unclassified employee becomes a vice-chancellor or chancellor, his related data

must be reclassified to secret. Having declared views $U\_ADDRESS$ and $S\_ADDRESS$, it is not necessary to reclassify the data.

The objective of this chapter is to show how the view concept can be used to implement a multi-level security policy on the top of a single-level OODB system. In Section 6.2, we describe the basic concept of multi-level secure databases. In Section 6.3, we discuss the essential features of the view model proposed by Bertino [31]. Later we present possible extensions of the view model to incorporate the mandatory label-based security policy. We show how the multi-level view, the content, context, and dynamic classification can be supported by the model. In Section 6.4, we develop a security model for OODB systems based on object views. Aggregation and inference problems are addressed in Section 6.5. Polyinstantiation is discussed in Section 6.6. Section 6.7 provides an evaluation of the model. Section 6.8 concludes the chapter.

## 6.2 A Multi-level secure Databases

A multi-level secure database contains information with different security (or sensitivity) levels. The security levels may be assigned to the data depending on the content, context, aggregation, or time. An effective security policy for multi-level databases must ensure that users have a suitable clearance before they access the information. To fulfill this requirement, each entity is assigned a security attribute. Attributes associated with active entities (or subjects) are called *clearance levels* while attributes associated with passive entities (or objects) are termed *security (sensitivity or classification) levels.* Subjects are not allowed to modify these attributes and their values. The modification of these attributes can only be done by the system security officers. The set of security and clearance levels form a partially ordered lattice with ordering relation "$\geq$" (for example, UNCLASSIFIED < CONFIDENTIAL < SECRET < TOP-SECRET). For security levels $L_1$ and $L_2$ , $L_1 \geq L_2$ means that security level $L_1$ *dominates* security level $L_2$ (if $L_1 > L_2$, it means that $L_1$ *strictly dominates* $L_2$).

The security attributes may be extended to include non-hierarchical cases that incorporate the *need-to-know* requirements. For example, we may have a security level named [SECRET,ASIACRYPT] - users with clearance SECRET and members of ASIACRYPT can access it. In other words, not all SECRET users are allowed the access. Such divisions create so-called *compartments (or categories).* In OODB systems, the encapsulation feature combined with the security labels provides a natural protection for objects. However after the value leaves the protection of the encapsulated object, the security cannot be guaranteed. To ensure that the security will not be compromised, the *flow of information* has to be restricted in some way. A number of models have proposed, the earliest and the best known is the Bell-LaPadula model [26] (see also Section 3.7).

The Bell-LaPadula model is based on two properties: the *simple security property* and the **-property.* According to the simple security property, a subject is allowed to read information from an object (or a passive entity) if the clearance level of the subject dominates the security level of the object. The *-property requires that a subject has the write access to an object if the subject clearance level is dominated by the security level of the object. Informally, a subject can *read-down* (simple security property) and can *write-up* (*-property). A number of extensions to the Bell-LaPadula security model were proposed [69, 87, 114, 122, 140, 149, 160, 210, 223]. These extensions address some specific problems related to database systems, for instance inferring unauthorized information from the legitimate responses and the information flow that occurs as a result of inheritance and the message passing in OODB systems.

## 6.3 View Model

Smith in [196] identifies three "dimensions" of the protection:

1. *the data itself may be classified,*

2. *the existence of the data may be classified, and*

3. *the reason for classifying the data may be classified.*

We define *access-views* to deal with the first two dimensions of the protection, The third security dimension is addressed by the introduction of *security-constraints.* The sets of access views and security constraints constitute meta-classes ACCESS-VIEWS and SECURITY-CONSTRAINTS, respectively. The meta-class ACCESS-VIEWS is used to enforce of the mandatory access control, and the meta-class SECURITY-CONSTRAINTS (which specifies restrictions that the view must satisfy) is used to control the security levels of views.

Note that Bertino's conceptual model [31] is proposed for a context not relevant to security. An instance of the ACCESS-VIEWS has the following format:

> **access-view** Viewname [parameters]
> – view specification as described in Section 2.4 of Chapter 2
> [**av_level** view_security_level]
> [**av_range** upper_and_lower_security_level]
> [**mac-constraint set-of** <SECURITY-CONSTRAINTS> mac_names]

The *Viewname* is the unique name of the access view and must be different from the names of other access views, security constraints, and classes. Parameters are fixed to the actual values at the time the view is evaluated, and corresponding objects are evaluated dynamically.

**av_range** imposes restrictions on the range of security levels of base classes (views) and properties used in the definition of a view. **av_level** indicates the security level of the name of the view. If the range is not indicated, then the view can contain the *single-level* properties and base classes (or views) that is indicated by **av_level**.

**mac-constraint** stands for mandatory access control constraints and consists of the set of security rules that the view must satisfy. Each element in this set is an instance of the class SECURITY-CONSTRAINTS.

The concepts are illustrated on two examples which use a university database shown in Figure 2.3 in Section 2.2.7 of Chapter 2.

**Example 6.1** *Define the multi-level class STUDENT whose instance variable GAP is classified SECRET and others are UNCLASSIFIED.* □

The class STUDENT can be represented as two views: a view *U_Student* with security level UNCLASSIFIED (containing the unclassified instance variables) and a view *S_Student* with security level SECRET containing the secret instance variable *GAP* and superview *U_Student.*

> **access-view** U_Student
> **select** S.idno, S.name, S.age, S.status, S.address, S.spouse, S.sex, S.subject, S.Start-Date, S.Graduate-Date, S.takes
> **from** S:STUDENT
> **av_level** UNCLASSIFIED

**access-view** S_Student
**properties** [Greatest_Point, SECRET]
**select** S.GAP
**from** S:STUDENT
**superview** U_Student
**av_level** SECRET
**av_range** [UNCLASSIFIED, SECRET]

The access view *U_Student* is a single-level view, and contains all variable instances of the base class STUDENT except GAP. It is labeled UNCLASSIFIED. The access view *S_Student* which is defined on top of *U_Student* is a multi-level view because in addition to secret instance variable *Greatest_Point* it inherits all properties of superview *U_Student* which are unclassified.

**Example 6.2** *Define the class EMPLOYEE such that the instance variable "address" is classified SECRET if the value of the instance variable "profession" is a "chancellor" or a "vice-chancellor" otherwise is UNCLASSIFIED.* □

One possible representation of that is to create two views, *U_Employee* and *S_Employee,* labeled UNCLASSIFIED and SECRET, respectively.

**access-view** U_Employee
**select** E
**from** E: EMPLOYEE
**where not** (E.profession ="chancellor" **or** E.profession ="vice-chancellor")
**av_level** UNCLASSIFIED

**access-view** S_Employee
**select** E
**from** E: EMPLOYEE
**where** (E.profession =”chancellor” **or** E.profession =”vice-chancellor”)
**av_level** SECRET

As shown in the above examples, the enforcement of security constraint has been done without the redesign of the schema.

There is a problem with the above examples. Because the security level of the view *U_Employee* is unclassified, unclassified users will know that there are expected employees who are classified at a higher level. In order to hide the classification constraint but still enforce it (and also to simplify verification and assurance process), we introduce the meta-class SECURITY-CONSTRAINTS whose instances have the following format.

> **security-constraint** sc-name [parameters]
> [predicates] [property-name.**level** = property_security_level]
> [predicates] [view-name.**level** = view_security_level]
> [**sc_level** sc_security_level]

The *sc-name* is the unique name of the security constraint (it must be different from the names of other security constraints, classes, and access views). Parameters are fixed at the time the view is evaluated. So when the values of the parameters are determined, the corresponding security constraints are evaluated for the view and its properties.

**sc_level** indicates the security level of the security constraint. Note that the security level of properties and views can be indicated either in the access view definitions or in the security constraint definitions.

Constraints can be *simple, content,* and *aggregate security* ones [79]. A *simple constraint* classifies an entire view property or view. For example, the view property *Greatest_point* in view *S_Student* of Example 6.1 is secret (all *Greatest_point* values will be secret). The view *S_Student* can be redefined as follows.

| | |
|---|---|
| **access-view** S_Student | **security-constraint** Sc_S_Student |
| **properties** [Greatest_Point] | Greatest_Point.**level**= SECRET |
| **select** S.GAP | S_Student.**av_level** SECRET |
| **from** S:STUDENT | |

> **superview** U_Student
> **mac-constraint set-of** < SECURITY-CONSTRAINTS > mac_S_Student
> **av_range** [UNCLASSIFIED, SECRET]

If the statement *S_Student.mac_S_Student* = **insert***(Sc_S_Student)* is executed, the address of *Sc_S_Student* will be added to the set of *mac_S_Student* automatically.

A *content-based constraint* provides a means of classifying data at the object or property value level by using a predicate based on the values of some objects and/or properties. For example, the security-constraint *Sc_Employee* classifies the view *Av_Employee* at either secret or unclassified level depending on the value of the property *profession*. Using the security constraint definition, Example 6.2 can be redefined as follows.

```
access-view Av_Employee
select E
from E: EMPLOYEE
mac-constraint set-of < SECURITY-CONSTRAINTS > mac_Employee

security-constraint Sc_Employee
if (E.profession ="chancellor" or E.profession ="vice-chancellor")
then Av_Employee.level = SECRET
else Av_Employee.level = UNCLASSIFIED
```

If the statement *Av_Employee.mac_Employee* = **insert** *(Sc_Employee)* is executed, then the address of *Sc_Employee* will be added to the set of *mac_Employee* automatically. Whenever the view *Av_Employee* is evaluated, the view is classified secret if it contains instances with profession "chancellor" or "vice-chancellor", otherwise is unclassified.

An *aggregate constraint* classifies a collection of property values (say ten or more/less) or relationship among data at a higher security level. For example, if we define a view which retrieves accessories supplied by a specific supplier, the associated security constraint can be declared. If the number of the accessories is more than ten, the view is classified at a higher level say SECRET.

## 6.4 Secure Multi-level View Model

Let us emphasize two points. First, our design supports multi-level objects at the view layer only. The multi-level views will be mapped onto single-level objects. Second, our design for mandatory access control relies on the underlying mandatory security kernel. An OODB system is considered secure if (see [160]):

1. no subject is able to obtain information without authorization,
2. no subject is able to modify information without authorization,
3. no mechanism exists whereby a subject authorized to obtain information can communicate that information to a subject not authorized to obtain it, and

4. no subject is able to execute a method without authorization.

Requirements (1), (2), and (4) are usually addressed by the discretionary access control while properties (1) and (3) are normally addressed by the mandatory access control. Our concern in this chapter is to enforce mandatory security policy using a secure multi-level view object model. Discretionary access control through view object models is discussed in Chapter 5.

We assume that the entities of the security classification are all kinds of objects. The entities include access views, security constraints, classes, objects, methods, and instance variables. Each entity in the database (except atomic objects) and each user has an associated *security level.* We also assume that the following methods are available to retrieve security information associated with an entity $e$.

**Level**(e) - displays the security level of entity $e$.

**Lower**(e) - displays the lower level range of entity $e$.

**Upper**(e) - displays the upper level range of entity $e$.

Denote by LUB the least upper bound, and GLB the greatest lower bound.

## 6.4.1 View

In general, a view can be constructed in two distinct ways [1, 21]: *top-down* or *bottom-up*. In the *top-down* approach, large classes (or views) are divided into smaller ones via *specialization* (a similar operation in relational systems is to define a view by selecting a subset of tuples from a large table). In the *bottom-up* approach, small classes (or views) are combined to form larger classes via *generalization* (the analogous operation in relational systems is to define a view as the union of several tables). A view constructed in the latter case may contain more information of various levels of security and hence, it must be classified at the highest of these levels.

**Classification Rule 1** *(View Property). If an access view $v$ is constructed on classes (or views) $v_1, v_2, \ldots, v_n$, the security level of $v$ must satisfy:*

$$v.\mathbf{av\_level} \geq LUB\{v_1.\mathbf{av\_level}, v_2.\mathbf{av\_level}, \ldots, v_n.\mathbf{av\_level}\}.$$

*The view range must contain the security level of $v$,*

$$\mathbf{Lower}(v) \leq v.\mathbf{av\_level} \leq \mathbf{Upper}(v).\square$$

From now on we refer to properties and methods of an access view as *view facets*. *View facets* can be derived from the base views or classes, or can be defined independently. Moreover, the security level associated with each facet can also be derived or defined.

**Classification Rule 2** *(View Facet Property). If $x$ is a facet of a view $v$ and is derived from a base view $w$, the security level of $x$ must satisfy both*

$$\textbf{Level}(v.x) \geq v.\textbf{av\_level},$$

*and*

$$\textbf{Level}(v.x) \geq \textbf{Level}(w.x).$$

*If $x$ is defined or redefined in $v$, the security level of $x$ must dominate the security level of $v$, i.e.,*

$$\textbf{Level}(v.x) \geq v.\textbf{av\_level}.\Box$$

**Classification Rule 3** *The security range of a view must contain the security level of all facets contained in the view, i.e., if $x_1, x_2, \ldots, x_n$ are facets of $v$, then the following must be held:*

$$GLB\{\textbf{Level}(v.x_1), \textbf{Level}(v.x_2), \ldots, \textbf{Level}(v.x_n)\} \geq \textbf{Lower}(v),$$

*and*

$$LUB\{\textbf{Level}(v.x_1), \textbf{Level}(v.x_2), \ldots, \textbf{Level}(v.x_n)\} \leq \textbf{Upper}(v).\Box$$

A *view name* is the external representation of an access view. When a user wants to access a view definition, they must first be authorized to access *view name*. Every access view $v$ is defined by a view specification which has a security level, **av_level**. A user is able to access the view if the security level of the view is dominated by the security level of the user.

**Classification Rule 4** *(Subject Property). A user $u$ can access an access view $v$ if one of the following holds:*
*a) if the view $v$ is a single-level view, and the security level of the user $u$ dominates the security level of the access view $v$, i.e., ($v.$**av_level** $\leq$ **Level**$(u)$);*
*b) if the view $v$ is a multi-level view, and the lowest security range of the view $v$ is dominated by the security level of the user $u$ (**Lower**$(v) \leq$ **Level**$(u)$).*

*The user $u$ is able to access facet $x$ of the view $v$ if the security level of $x$ is dominated by the security level of the user $u$, i.e.,* **Level**$(v.x) \leq$ **Level**$(u)$. $\Box$

Rule 4 is the simple security property specified in the Bell-LaPadula model. Only the *read-up* is permitted. Users with security level lower than the security level of a view are not able to access the view definition and consequently the view instances.

**Classification Rule 5** *(View-instance Property). If a view $v$ consists of objects $o_1, o_2, \ldots, o_n$, then the security level of the view $v$ must satisfy:*

$$v.\textbf{av\_level} \geq LUB\{\textbf{Level}(o_1), \textbf{Level}(o_2), \ldots, \textbf{Level}(o_n)\}.\square$$

The set of database objects contained in the view instance is controlled by the view specification and a set of associated security constraints. Every *security-constraint* is defined by a set of constraints, and a security level (**sc_level**).

**Classification Rule 6** *(Security-Constraint Property). If $x_1, x_2, \ldots, x_n$, and $s_1, s_2, \ldots, s_m$ are facets and their nominated security levels contained in the security-constraint sc, respectively then the following must be held:*
*a) The nominated security levels $s_i$ $(1 \leq i \leq n)$ for the facets contained in constraints of a security-constraint must dominate the security levels of the facets, i.e., $s_i \geq \textbf{Level}(x_i)$ for all $i$, $1 \leq i \leq n$;*
*b) the security level of the security-constraint is the least upper bound of the security levels of all facets and all nominated security levels contained in the constraints, i.e.,*

$$sc.\textbf{sc\_level} \geq LUB\{\textbf{Level}(x_1), \textbf{Level}(x_2), \ldots, \textbf{Level}(x_n), s_1, s_2, \ldots, s_m\},$$

*and*

$$\textbf{Level}(sc) \geq \textbf{Level}(v)$$

*where $v$ is the view associated with sc.* $\square$

The above rule indicates that the security level of a security constraint must be at least equal to the least upper bound of security levels of information contained by the constraints. Moreover, the security level of this must dominate the security level of the associated view. For example, by applying the above rule to *Sc_Employee* in Example 6.2 of Section 6.2, the security level of the *Sc_Employee* must be at least SECRET. If the **sc_level** is not indicated by the user, the computed security level will be considered for **sc_level**.

**Classification Rule 7** *A user $u$ can access a security constraint $sc$ if the security level of $sc$ is dominated by the security level of the user $u$,* **Level(u)** $\geq$ $sc$.**sc_level**. □

There are four types of inheritance among views: *constraint, strict constraint, proper specialization,* and *specialization* inheritance (for detailed description see Chapter 5). In the *constraint inheritance,* views consist of all object instances of base classes (or views) that satisfy given constraints. We only use selection operation. In the *strict constraint inheritance,* not only the set of database objects contained by a view is constrained, but the base classes (or views) properties are also projected. We apply selection and projection operations. In the *specialization inheritance,* not only views inherit all the properties of the base classes (or views), but they also have some new additional properties and methods. In the *proper specialization inheritance,* views filter out the object instances of the base classes and project the properties. They also contain new additional properties or methods.

In the case *constraint* and *strict constraint inheritance*, the amount of information provided by a sub-view is smaller than the information contained in the base view. So the security level of the sub-view may be the same as the security level of the base view.

**Classification Rule 8** *(Hierarchy Property). Suppose a view $v$ is derived from a view $w$, then their security levels must satisfy one of the following:*

**Level**($v$) = **Level**($w$), *if the relationship is the strict constraint or constraint inheritance,*

*or* **Level**($v$) $\geq$ **Level**($w$), *if the relationship is the proper specialization or specialization inheritance.*

*The set of the constraints of the view $v$ must contain the set of constraints of the view $w$.* □

A view may inherit or derive the facets from one or more super-views or the base-views. In the case of a conflict, the following rule will be used to resolve the conflict.

**Classification Rule 9** *(Multiple Inheritance). Assume a view $v$ inherits the facet $x$ from views $v_1, v_2, \ldots, v_n$, then the security level of $x$ in $v$ must dominate* *the least upper bound of the security levels of $x$ in $v_i$ $(1 \leq i \leq n)$,*

$$\textbf{Level}(v.x) \geq LUB\{\textbf{Level}(v_1.x), \textbf{Level}(v_2.x), \ldots, \textbf{Level}(v_n.x).\}$$

*The view $v$ inherits the facet $x$ with the highest security level in $v_i$ $(1 \leq i \leq n)$. If there are more than one such $x$, then a priori rule*[2] *must be enforced to resolve the conflict.* □

It is possible to define composite (or aggregation) hierarchy on views. The domain of the instance variable can be a view or a class. The instance variable is provided with the keyword **additional-properties** for a view property. A view $v$ may then be a composite of views $v_1, v_2, \ldots, v_n$.

**Classification Rule 10** *(Composite Property). Let a view $v$ be a composite view of the views (or classes) $v_1, v_2, \ldots, v_n$, then the security level of $v$ must satisfy:*

$$v.\textbf{av\_level} \geq LUB\{v_1.\textbf{av\_level}, v_2.\textbf{av\_level}, \ldots, v_n.\textbf{av\_level}\}.\square$$

## 6.4.2 Derivation Rules

The next two rules are needed to obtain a view from a single-level database.

**Classification Rule 11** *(Single-level View Instantiation). If an object in the database has a security level dominated by the security level of a view, the object may be derived as an instance of the view.* □

Note that a view may be evaluated dynamically [31]. Then view instances are created only if a user or process requests it.

**Classification Rule 12** *(Multi-level View Instantiation). If the security level of the view property dominates the security level of the corresponding property value of an object in the database, then that object is derived as an instance of the view from the database.* □

All derived single-level objects corresponding to the view properties are joined to instantiate a multi-level view. As said in Rule 5, the security level of derived data is greater than or equal to the least upper bound of the security levels associated with the derived data. Those derived data will be presented to a user whose security levels are dominated by the security level of the user.

[2]For example, the facet $x$ associated with the view which is closer to the view $v$, dominates other views.

### 6.4.3 Updates and Object Creation

In relational databases, updates of any views are impossible due to the so-called *view update problem* (if views only contain the key of their (one) underlying base relation; they can be updated). As shown by Scholl, Laasch and Tresch in [190], the view update problem disappears in OODB systems, if the query language preserves object identities. This happens because objects have an identity independent of their associated values. Views can almost freely be updated since the objects contained in the result of a query are the base objects. For detailed discussion of the properties of the query language that allows the updates of views, the reader is directed to [190].

**Classification Rule 13** *(Insertion). The security level of the inserted object is computed according to a set of security constraints associated with views. If the computed security level for the entire object is unique, then the single security level is assigned to the object and the single-level object is stored into the database. If the computed security levels are different for every properties, the inserted datum is decomposed to single-level objects according the computed security levels, and then stored.* □

According to the above rule, if the inserted object is multi-level (note that the underlying database is single-level), the object must be decomposed into several single-level objects. The solution to the object decomposition was proposed in [38].

**Classification Rule 14** *(Updating). For every view property value, the value with the computed security level is stored back into the database if either*
*1) the computed security level dominates the security level associated with the user on whose behalf it was computed, and the computed security level is equal to the security level of the corresponding property in the database; or*
*2) the downgrade of the property value is authorized and confirmed by the security officer.* □

## 6.5 Aggregation and Inference

### 6.5.1 Aggregation

The aggregation problem occurs when a user can collect some data of low security classification and deduce other data with higher security classification.

**Example 6.3** *Personal specification and monthly wage of the academic staff are unclassified, but the association of a particular monthly wage with a specific individual academic staff is classified secret.* □

One possible solution is that the unclassified view *U_Academic_Staff* for personal specification of the academic staff and the unclassified view *U_Monthly_Wage* for monthly wage of the academic staff are declared. The secret view *S_Academic_Monthly_Wage* is declared to protect the association of academic staff with their monthly wages.

Note that, by declaring the above views, the protection against inference may still fail. For example, if there were only a few academic staff members and their rank was known or could be guessed, the association could be reconstructed. In such cases, the preferred solution would be to classify one of the two entities in the relationship at the higher level. For example, putting the monthly wages in a separate view at the secret level would protect the association, but it would also, in effect, classify the monthly wages.

If a view $v$ is the aggregate of views $v_1, v_2, \ldots, v_n$ and the security level of $v$ strictly dominates the security level of all $v_i$ $(1 \leq i \leq n)$, then the aggregation problem occurs, and $v.\textbf{av\_level} > v_i.\textbf{av\_level}$ for all $i$, $1 \leq i \leq n$. To solve that, the following rule must be fulfilled.

**Classification Rule 15** *If a view $v$ is the aggregate of views $v_1, v_2, \ldots, v_n$, then there should exist at least a single $v_i$ such that its security level is equal to the security level of the $v$ and $v.\textbf{av\_level} = v_i.\textbf{av\_level}$ for some $i$, $1 \leq i \leq n$.* □

### 6.5.2 Inference

The inference problem occurs when a user can deduce (or infer) information from a collection of individual inquires. Solutions to the inference problem were proposed in [70, 60, 111, 209] in the context of statistical and relational databases. Some typical approaches to handle the problem are:

1. the introduction of restrictions on the set of allowable queries generated by a user,
2. the addition of "noise" to the data, and
3. the augmentation of a database with a logic-based inference engine to detect security violations.

Views provide exactly the right basis to address the inference problem or at least to decrease the *"size"* of the inference problem. If properly used, views impose restrictions on the derived data which are extracted from the base data. The derived data are visible to the user (via the view) and can be tailored to the user needs. Views represent the real-world semantic relationship among data. This relationship is usually exploited during the inference process. A user is constrained to a smaller portion of the database through the proper use of the views. This reduces the number and types of queries issued by the user. Therefore views can be used to impose the first and the second approaches of handling the inference problem (the introduction of restriction on the set of allowed queries and the addition of noise to the data). The multi-level security model can be augmented with logical-based inference engine which will detect security violations.

Note that some researchers believe that aggregation and inference are the different faces of the same problem [132].

## 6.6 Polyinstantiation

Polyinstantiantion generally occurs when two subjects at different security levels see two different forms of a single entity in the real world. In a multi-level view model, the different forms of an entity could relate to different view definitions, to the same view-name with different security-levels, and to different entity values. So we have three possible polyinstantiations.

- *View-definition polyinstantiation.* A view may contain different facets with different security-levels. For instance, an unclassified user sees a view $v$ which contains the properties *(idno, name)* and has a method *change-name.* A secret user sees $v$ which contains *(idno, name, salary)*, and has two methods *change-salary* and *change-name.*

- *View-name polyinstantiation.* An actual view is identified by the pair: a view-name $v$ and the security-level (**av-level**). So that there may be several views identified by the same view-name $v$ which correspond to different security-levels.

- *View-instance polyinstantiation.* An object is identified by an object identifier (OID) and an associated security-level, so that a multi-level view may contain several object instances for an OID corresponding to different security-levels.

A polyinstantiated view-name may arise whenever an unclassified (or confidential) user requests to use the same name for defining a view which is already used by a secret user. To handle this type of polyinstantiation, we suggest that a view name associated with the security-level of the user is used to name the view. For instance, for a view $v$, the unclassified user will use *U-v* and the secret user will use *S-v*.

A polyinstantiated view-definition may occur whenever an unclassified user modifies the view definition and includes the same facets which already exist at higher security level. This type of polyinstantiation will not occur in our model as the security level of a view must be the least upper bound of the security levels of all facets contained in the view (Rule 1). The view can be accessed by a user $u$ if the security level of $u$ dominates the security level of the view (Rule 4).

A polyinstantiated view-instance may happen if two subjects at different security levels request the same identifier for two different objects which represent two different entities. Therefore, if globally unique object identifiers are used to identify objects, the polyinstantiation will not occur in the model.

Boulahia-Cuppen et. al. in [38] presented a decomposition algorithm for OODB systems which provides the possibility of choosing globally unique object identifiers (by partitioning the set of object identifier into several pairwise disjoint subsets associated with each security level). For instance a multi-level object *O*, which is an instance of a multi-level view *S_Student*, will be decomposed to *U-O*, *C-O*, and *S-O*. Each of them is actually a single-level object corresponding to unclassified, confidential, and secret which is physically stored in a single-level database. Dynamic links are also created between these objects. For example, in the *S-O*, there are pointers to the confidential properties stored in the *C-O*. In the *C-O*, there are pointers to unclassified properties of the *U-O*. This means that if a classified user updates the unclassified properties of *U-O*, the update will be automatically propagated to the instances of *C-O* and *S-O*. Note that the values of the properties which point to low-level objects can be updated by the users cleared to access them. If this happens, the pointer to the low-level database is broken, and the value of the object in the low-level database stands as a cover story. For detailed discussion, the reader is directed to [38].

Therefore by employing the above approach to decompose multi-level views into single-level objects and selecting object identifiers corresponding to each security levels, view-instance polyinstantiation does not occur in our model. Furthermore, there is no *referential ambiguity* as all references to an object use the unique OID.

Note that although there is no view-instance polyinstantiation because OIDs are unique, but it is possible that the value of the property *idno* is not unique. One way of avoiding this is the use of security constraints that requires all instances of *PERSON* to have the same security level.

## 6.7 Evaluation of the Proposed Model

Gajnak in [87] chooses a view of a multi-level secure database as a set of associated facts, and presents three general principles for a *well formed* multi-level secure database. They are: *the granularity principle, the dependency principle,* and *the determinacy principle.* The granularity principle states that the finest level of granularity for the protection purpose should be a structure which correspond to atomic facts. The dependency principle requires that the security level of a fact dominate the security level of any other fact it depends upon. The determinacy principle states that factual dependencies should not be ambiguous.

A *fact* is an encapsulated unit of information. Facts which do not depend on other facts, are called atomic. Six types of atomic facts can be distinguished in the multi-level view model. They are:

1. the fact of the object existence (which is presented by the object identifier),
2. the association between an object and the values of its properties,
3. the association between an object and a view-instance,
4. the association between a view definition and a view name,

5. the association between a view definition and the corresponding security constraint, and

6. the hierarchical association (inheritance or composite).

The following four factual dependencies also exist among these facts.

1. The fact of association of a property value with an object depends on the existence of the object.

2. The fact of association of a view-instance with object depends on the existence of objects.

3. The fact of association of a view-name with a view-definition depends on the existence of the view definition.

4. The fact of association of a security constraint with a view definition depends on the existence of the view definition.

Now, consider the three principles, *granularity, dependency,* and *determinacy.* The granularity principle states that the finest level of labeling granularity must cover all the above mentioned facts. This is exactly the entities that we have assumed for labeling.

The dependency principle states that the security level of an association must dominate the security level of its components. Rules 1-2, 5-6, and 8-10 take care of that.

The determinacy property addresses the problem of interpreting the database in the face of polyinstantiation. As discussed before because we assume that globally unique object identifiers (OID) are used to identify objects, then the polyinstantiation will not occur in the model.

We believe that the proposed multi-level view model is practical as it maps its components to the set of associated facts and it directly supports the three principles of multi-level data: granularity, dependency, and determinacy.

## 6.8 Conclusions and Remarks

In this chapter, our objective was to provide a multi-level view model derived from a single-level secure OODB system. The model allows us to use the existing security kernel. It also overcomes the difficulties of handling OODB systems with multi-level objects. One distinct advantage of our approach is that the multi-level view model relies on an underlying security kernel for the enforcement of mandatory security properties.

Our model can be seen as an extension of the view model proposed in [31]. We have introduced the notion of *security constraint* which is associated with each view. We have also discussed the usage of views and security constraints to handle simple, content-dependent, and context-dependent classifications. We have then described the multi-level security properties for a secure multi-level view model based on a secure single-level OODB system. Finally, we have discussed issues such as aggregation, inference, and polyinstantiation.

Note that two types of users require access to views: the database *security officers* and *users.* It is clear that the security officers require unrestricted access to the views in order to define and maintain the database. Users require some access to the view definitions in order to be able to query the database. Two policies of access control have been proposed. The first one allows users to browse through the external schema. The second policy assigns discretionary access rights to views. As the first policy violates the least privilege principle, the second policy is recommended. In other words, the discretionary access control should be implemented on the top of the multi-level view model.

# Chapter 7

# Summary, Results and Future Directions

## 7.1 Summary

The *main goal* of the thesis was to investigate the access control model for object-oriented databases. The motivation for the project was the lack of an acceptable OODB model which could be used for the access control. Chapter 2 presented key concepts of O-O models such as types/classes, objects, complex objects, aggregation, and discussed O-O properties such as inheritance, encapsulation, and methods. Persistence, secondary-storage management, security, authorization, concurrency, and transactions in OODB systems were also examined. Chapter 2 was concluded by a survey of view models in OODB system. Their properties were also pointed out.

Review of security in databases was given in Chapter 3. Security models for relational and object-oriented databases were presented. Common threats to the database and possible countermeasures were also considered. A security policy specifies security requirements which must be satisfied by database management system. A list of security policies for the design of the access control were provided. The two major access control models, mandatory access control (MAC) and discretionary access control (DAC) were described. The access matrix and Bell-LaPadula models were presented. The security issues in the O-O environment such as access privileges (or access modes), propagation of authorizations, degree of granularity, control of authorizations, etc. were also discussed. The

ORION, Irish, DAMOKLES, SORION, SODA and several other access control models were presented. Those models were selected as they, in our view, offer some partial solutions to the security issues present in OODB systems. We also discussed common security drawbacks of those models. A list of classification rules for secure OODB systems were provided. Examples of architecture to construct secure multi-level databases were given. They were: *Trusted Subject*, *Kernelized*, *Replicated*, and *Integrity Lock.*

The O-O concept allows expression of rules for computing implicit authorizations from the explicit ones. It is necessary to have an efficient mechanism to evaluate implicit authorizations each time an access request is checked for validity. Chapter 4 provided a cryptographic mechanism which was based on unique and secure access keys for each access entity (object or class). In this mechanism, owners and user groups are identified by their unique passwords. Access keys and passwords for implicit authorizations are derived from related entities by applying pseudo-random and SIFF functions during the query processing. The security of the system is based on the difficulty of predicting the output of the pseudo-random function and finding extra collisions for the SIFF function, both of which are known to be computationally difficult.

Chapter 5 addressed content-dependent authorizations in OODB systems. We used the view model proposed by Bertino and showed how discretionary security requirements for authorization systems can be incorporated into the model. Parameterized views were discussed. We described how a view hierarchy could be inferred from views. Four types of inheritance among views were discussed: *constraint, strict constraint, proper specialization,* and *specialization* inheritance. Rules for computing implicit authorizations from the explicit ones were also formulated.

In Chapter 6, a new design approach for a secure multi-level object-oriented database system based on views were proposed. The central idea was to provide the user with a *multi-level view* derived from a single-level secure object-oriented database. The database operations performed on the multi-level views are decomposed into a set of operations on the single-level objects which can be implemented on any conventional mandatory security kernel. We also presented security properties for a secure multi-level view model. We showed that this approach allowed us to overcome the difficulties of handling content and context dependent classification, dynamic classification, aggregation and inference problems in multi-level object-oriented databases.

## 7.2 Results

Results achieved during the work over the project have been published or submitted for publication. The complete list of papers published or submitted is as follows:

- "A Cryptographic Solution to Discretionary Access Control in Structurally Object-Oriented Databases," based on Chapter 4 (published in *Proceedings of the 6th Australian Database Conference (ADC'95)* (R. Sacks and J. Zobel, eds.), vol. 17(2), (Adelaide, Australia), pp. 36–45, Australian Computer Science Communications, Jan. 1995);

- "A Cryptographic Mechanism for Object-Instance-Based Authorization in Object-Oriented Database Systems," based on Chapter 4 (published in *Proceedings of The 14th International Conference on Object-Oriented & Entity Relationship Modeling, Queensland*,(M. P. Papazoglou, ed.), vol. 1021 of *Lecture Notes in Computer Science*, (Queensland, Australia), pp. 44–54, Springer-Verlag, Dec. 1995);

- "A Model of Authorization for Object-oriented Databases based on Object Views, based on Chapter 5 (published in *Proceedings of The 4th International Conference on Deductive and Object-Oriented Databases, Singapore* (T. Ling, A. Mendelzon, and L. Vieille, eds.), vol. 1013 of *Lecture Notes in Computer Science*, (Singapore), pp. 503–520, Springer-Verlag, Dec. 1995); and

- "A Multi-level View Model for Secure Object-Oriented Databases," based on Chapter 6 (accepted for publication by the Journal of Data & Knowledge Engineering, 1996).

- "Modeling A Multi-level Secure Object-Oriented Database Using Views", based on Chapter 6 (Pre-proceedings of the Australian Conference on Information Security and Privacy, The University of Wollongong, NSW, Australia, May 24-26, 1996 (accepted)).

## 7.3 Future Direction

Some aspects of the research presented in this thesis need further investigation. This section outlines some of these directions which need to be addressed.

1. **Implementation and Experimentation**

   In Chapter 4, a cryptographic mechanism for OODB systems was proposed. Since a normal proof of the mechanism is impractical and impossible to achieve for the scale and difficulty of the problem, there is a need to implement a prototype of the cryptographic access control mechanism to investigate its applicability, efficiency, and performance. Also a more robust database management system needs to be implemented to experiment with authorization administration. This will give a clearer picture on how complex the administration becomes in real life. Moreover, an insight may be gained into what other requirements are essential for a successful cryptographic access control mechanism. We assumed that there were two types of access: partial and full. In the full authorization, a lower-node is accessed by all higher-level nodes of the hierarchical structure. In the partial authorization, a specific node is accessed only. There is also a need for further study of a situation where a lower-level node in the hierarchical structure may be accessed only by some higher-level nodes.

2. **View-based Protection and Negative Authorization**

   Positive authorizations only were considered in the view-based authorization model proposed in Chapter 5. Our consideration could be extended by taking into account negative authorizations as well. Access control models with both positive and negative authorizations are lacking the consistency and completeness of the authorizations. This is due to the fact that satisfaction of the conditions in the views depends on the values of object properties that can change over time. Suppose that a user $u_1$ has a positive read privilege on a view $v_1$ which contains all students with start date later than "Feb 19, 1993". The user $u_1$ is forbidden to access all foreign students. The user $u_1$ has a negative read privilege on a view $v_2$ which contains all foreign students. If no foreign student with start date later than "Feb 19, 1993" exists, the access control state is consistent. However, since property values can change and new objects can be added, access control may become inconsistent. Therefore, the need for consistency and completeness criteria, and mechanisms to enforce them requires more investigation.

3. **Inference Detection and Elimination**

   Chapter 6 mentioned a number of ways in which views may be used to restrict the inference problem (for example, we may impose restrictions on the set of allowable queries generated by a user, or to add noise to the data). But these solutions are not sufficient to protect database against all inference threats. Therefore, there is a need for a further study of how to augment the secure multi-level view model with a logical-based inference engine as to provide protection against security violations during query processing.

4. **Implementation Strategies**

   Any security model adds overhead costs to the operation of a database system. If the security model adds too much overhead costs, the model is not practical. The security model must be simple to implement. Only if simple implementation strategies exist, it will be possible to trust that the implementation accurately reflects the model. Hence, there is a need for a study to find implementation strategies that economically and correctly implement the security model proposed in Chapter 6. Examples of implementation strategies for conventional databases are found in [127].

This is a sample only of possible directions future work in this area. We hope that contributions presented in this thesis, will encourage and stimulate other researchers working in the area to advance the theory and practice of the access control in O-O environments.

# Appendix A

# Hardjono et. al.'s Database Authentication based on SIFF

Hardjono, Zheng and Seberry [101, 102] have suggested a method of the authentication of data in database systems based on the use of pseudo-random function families and the SIFF. The scheme employs a trusted central authority (or party) which holds a database secret key $K_{db}$ and the secret information $s_{db}$ necessary for the checksum generation and validation and for the encipherment and decipherment of data in the record. In addition it is assumed that record $i$ has a unique record identifier $R_i$ and field $j$ has a field identifier $F_j$. $M_{ij}$ denotes the actual value of field $j$ in record $i$. Let $H = \{H_n | n \in \aleph\}$ be a *(t+1)-SIFF* mapping *n-bit* input to *n-bit* output strings. Furthermore, assume that $F = \{F_n | n \in \aleph\}$ is a pseudo-random function family, where $F = \{f_K | f_K : \sum^{l(n)} \rightarrow \sum^n, K \in \sum^n\}$ and each function $f_K \in F_n$ is specified by a *n-bit* string K.

To construct authenticator (checksum) for each record $i$, an instance of SIFF $h_i$ is chosen uniformly and randomly from $H_n$ such that:

$$h_i(f_{s_{db}}(R_i||s_{db})) = h_i(f_{K_{db}}(M_{i1}||R_i||F_1)) = \ldots = h_i(f_{K_{db}}(M_{it}||R_i||F_t)) = S_i,$$

where $S_i$ is a randomly chosen *n-bit* string, the checksum for record $i$. Figure A.1 shows this process.

One advantage of the method is that the checksum is calculated for each record. However, it is possible the authentication of each data element $M_{ij}(1 \leq j \leq t)$ is performed by checking

$$h_i(f_{K_{db}}(M_{ij}||R_i||F_j)) = S_i.$$

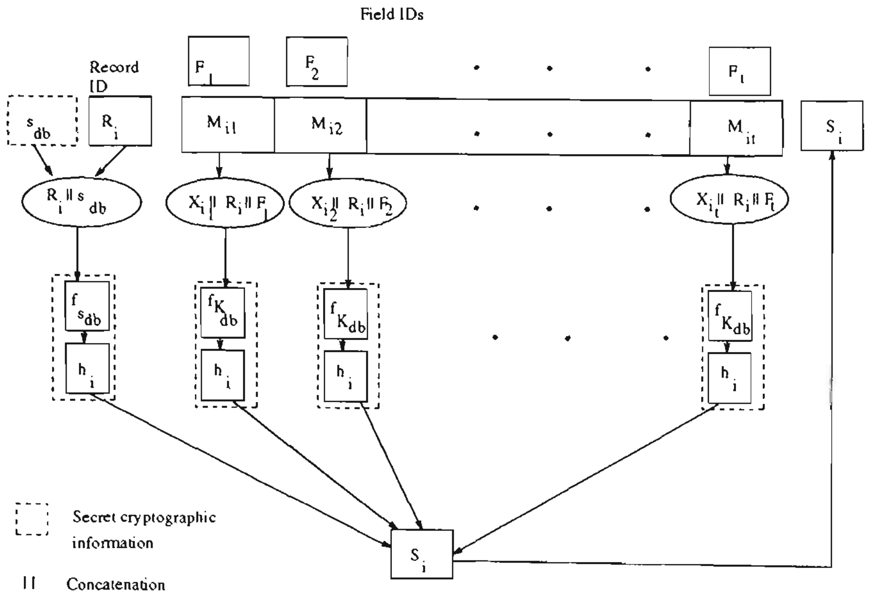


Figure A.1: Using SIFF for Record Authentication.

Another advantage is the possibility to place the description of the instances of SIFF associated with each record in the same storage as records. Finally, the authentication of data elements could be done without any secret cryptographic information. For a detailed discussion, the reader is directed to [101, 102]

# Appendix B

# An Improvement of Implementation of k-SIFF

Assume that a polynomial $P(x)$ of degree $k$ over finite $GF(2^n)$ has $k$ colliding points.

$$P(x) = a_0 + a_1 x + \ldots + a_{k-1} x^{k-1} = (x + b_0)(x + b_1) \ldots (x + b_{k-1})$$

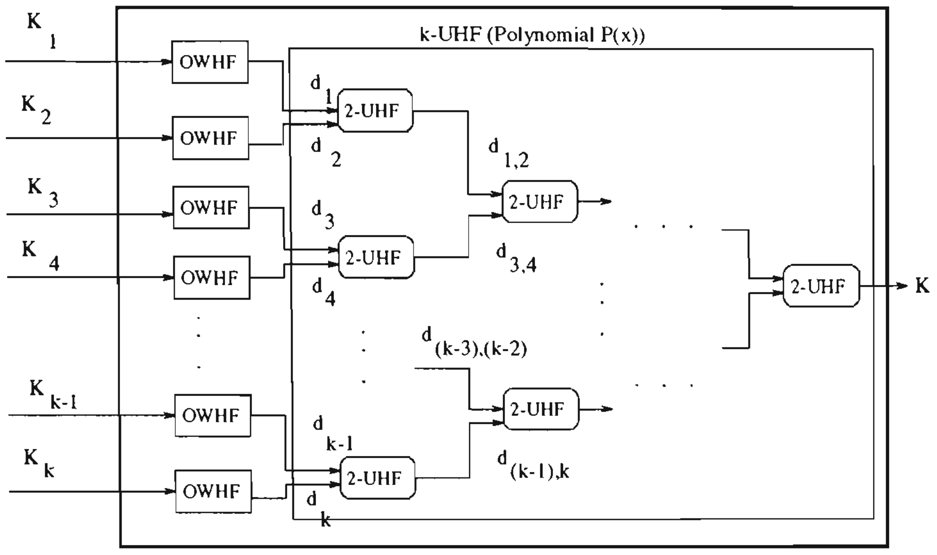
where $a_0, a_1, \ldots, a_{k-1}, b_0, b_1, \ldots, b_{k-1} \in GF(2^n)$.

When it is evaluated, the evaluation costs $k$ modular multiplications, $O(k)$. In order to convert $P(x)$ in such a way that the number of modular multiplications is reduced, we can apply the approach depicted in Figure B.1. The OWHF is any one-way hash function such as MD4, MD5 [172, 173], or HAVAL [226]. The UHF is any universal hash function with the collision accessibility - this is our polynomial $P(x)$

In the first layer, polynomials of degree two are defined or in other words they have two collisions. It is then required to calculate $k/2$ polynomials $P_{1,2}(x), P_{3,4}(x), \ldots, P_{(k-1),(k)}(x)$ such that $P_{1,2}(d_1) = P_{1,2}(d_2) = d_{1,2}, P_{3,4}(d_3) = P_{3,4}(d_4) = d_{3,4}, \ldots,$

In the second layer, again $k/4$ polynomials of degree two are defined such that they collide each pair outputs of the polynomials of the first layer, i.e., the polynomials $P_{1,2,3,4}(x), P_{5,6,7,8}(x), \ldots, P_{(k-3),(k-2),(k-1),(k)}(x)$ such that $P_{1,2,3,4}(d_{1,2}) = P_{1,2,3,4}(d_{3,4}) = d_{1,2,3,4}, P_{5,6,7,8}(d_{5,6}) = P_{5,6,7,8}(d_{7,8}) = d_{5,6,7,8}, \ldots$

If this is continued, the resulting last polynomial will generate the key $K$. Thus, the above approach provides this possibility to get the same number of collisions ($k$) while derivation for a given key will take $O(\log k)$ modular multiplication (each polynomial has degree two so its calculation takes $O(1)$ modular



OWHF : OneWay Hash Function

2-UHF : 2-Universal Hash Function

Figure B.1: Some improvement of implementation of k-SIFF.

multiplication).

There is however one problem to be solved; as there are many different polynomials and only one "path" is used, the system must know which key is being used. To clarify this, $P(x)$ generates the proper key as long as we plug in the correct key (one from $K_1, K_2, \ldots, K_k$). In order for key $K_i$ the proper path is chosen, it is suggested that if $i$ is odd, $P_{i,(i+1)}(x)$ is used. In case that $i$ is even, $P_{(i-1),i}(x)$ is used.

# Bibliography

[1] S. Abiteboul and A. Bonner, "Objects and Views," in *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data* (J. Clifford and R. King, eds.), pp. 238–247, SIGMOD RECORD, ACM Press, 1991.

[2] M. D. Abrams, "Role-Based Access Control Position Paper," in *Proceedings of the 17th National Computer Security Conference*, vol. 2, (Baltimore, Maryland), p. 491, Oct. 1994.

[3] N. R. Adam and J. c. Wortmann, "Security-Control Methods for Statistical Databases: A Comparative Study," *ACM Computing Surveys*, vol. 21, pp. 515–556, Dec. 1989.

[4] R. Ahad, J. davis, S. Gower, P. Lyngbaek, A. Marynowski, and E. Onuegbe, "Supporting Access Control in an Object-Oriented Database Language," in *Proceedings of the 3rd International Conference on Extending Database Technology, EDBT'92*, vol. 580 of *Lecture Notes in Computer Science*, (Vienna), pp. 184–200, Springer-Verlag, Mar. 1992.

[5] S. G. Akl and P. D. Taylor, "Cryptographic Solution To A Multilevel Security Problem," in *Advances in Cryptology Proceedings of CRYPTO'82* (D. Chaum, L. Rivest, and A. T. Sherman, eds.), pp. 237–250, Plenum Press, NY, Aug. 1982.

[6] S. G. Akl and P. D. Taylor, "Cryptographic Solution To A Multilevel Security Problem," *ACM Transactions on Computer Systems*, vol. 1, no. 3, pp. 239–248, 1983.

[7] A. Albano, G. Gheli, G. Occhiuto, and R. Orsini, "Galileo: A Strongly Typed Interactive Conceptual Language," *ACM Transactions on Database Systems*, vol. 10, June 1985.

[8] P. E. Ammann and R. S. Sandhu, "Implementing Transaction Control Expressions by Checking for Absence of Access Rights," in *8th Annual Computer Security Applications Conference*, pp. 131–140, IEEE Computer Society Press, 1992.

[9] P. E. Ammann and R. S. Sandhu, "The Extended Schematic Protection Model," *Journal of Computer Security*, vol. 1, no. 3,4, 1992.

[10] M. Atkinson, D. DeWitt, D. Maier, F. Bancilhon, and K. Dittrich, "The Object-Oriented Database System Manifesto," in *Proceedings of First International Conference on DOOD89*, (Research Park, Kyoto, Japan), pp. 223–240, Elsevier Science Publishers B. V. (North-Holland) IFIP, Dec. 1989.

[11] T. Atwood, J. Dubl, G. Ferran, M. Loomis, and D. Wade, *The Object Database Standard: ODMG-93.* Morgan Kaufmann Publishers, R. G. G. Cattell (ed), Release 1.1, CA, ISBN 1-55860-302-6 ed., 1994.

[12] T. Atwood, "ODMG93: The object DBMS standard," *Object Magazine*, pp. 37–44, Sept. 1993.

[13] R. W. Baldwin, "Naming and Grouping Privileges to Simplify Security Management Databases," in *Proceedings of the 1990 IEEE Symposium on Security and Privacy*, pp. 116–132, IEEE Computer Society Press, 1990.

[14] F. Bancilhon, "Object-Oriented Database Systems," in *Proceedings 7th ACM Symposium on Principles of Database Systems*, ACM, Mar. 1988.

[15] F. Bancilhon and N. Spyratos, "Update Semantics of Relational Views," *ACM Transactions on Database Systems*, vol. 6, Dec. 1981.

[16] J. Banerjee, H.-T. Chou, J. F. Garza, W. Kim, D. Woelk, and N. Ballou, "Data Model Issues for Object-Oriented Applications," *ACM Transaction on Office Information Systems*, vol. 5, pp. 3–26, Jan. 1987.

[17] A. Baraani-Dastjerdi, J. R. Getta, J. Pieprzyk, and R. Safavi-Naini, "A Cryptographic Solution to Discretionary Access Control in Structurally

Object-Oriented Databases," in *Proceedings of the 6th Australian Database Conference (ADC'95)* (R. Sacks and J. Zobel, eds.), vol. 17(2), (Adelaide, Australia), pp. 36–45, Australian Computer Science Communications, Jan. 1995.

[18] A. Baraani-Dastjerdi, J. Pieprzyk, and R. Safavi-Naini, "A Multi-level View Model for Secure Object-Oriented Databases," *Accepted for publication by Data & Knowledge Engineering*, 1996.

[19] A. Baraani-Dastjerdi, J. Pieprzyk, and R. Safavi-Naini, "Modeling A Multi-level Secure Object-Oriented Database Using Views," in *Pre-proceedings of the Australian Conference on Information Security and Privacy (accepted)*, (The University of Wollongong, NSW, Australia), May 1996.

[20] A. Baraani-Dastjerdi, J. Pieprzyk, R. Safavi-Naini, and J. R. Getta, "A Cryptographic Mechanism for Object-Instance-Based Authorization in Object-Oriented Database Systems," in *Proceedings of The 14th International Conference on Object-Oriented & Entity Relationship Modeling (OOER'95)* (M. P. Papazoglou, ed.), vol. 1021 of *Lecture Notes in Computer Science*, (Queensland, Australia), pp. 44–54, Springer-Verlag, Dec. 1995.

[21] A. Baraani-Dastjerdi, J. Pieprzyk, R. Safavi-Naini, and J. R. Getta, "A Model of Authorization for Object-Oriented Databases based on Object Views," in *Proceedings of The Fourth International Conference on Deductive and Object-Oriented Databases* (T. Ling, A. Mendelzon, and L. Vieille, eds.), vol. 1013 of *Lecture Notes in Computer Science*, (Singapore), pp. 503–520, Springer-Verlag, Dec. 1995.

[22] P. J. Barclay and J. B. Kennedy, "Viewing Objects," in *Advances in Databases, Proceedings 11th British National Conference on Databases (BNCOD)* (M. Worboys and A. Grundy, eds.), vol. 696 of *Lecture Notes in Computer Science*, (Keele, UK), pp. 93–110, Springer-Verlag, July 1993.

[23] D. K. Barry, "ODBMS Feature Listing," *Object Magazine*, pp. 48–53, Jan. 1993.

[24] C. Beeri, "Formal Model for Object-Oriented Databases," in *Proceedings of First International Conference on DOOD89*, (Research Park, Kyoto,

Japan), pp. 223–240, Elsevier Science Publishers B. V. (North-Holland) IFIP, Dec. 1989.

[25] D. E. Bell, "Lattices, Policies, and Implementations," in *Proceedings 13th National Computer Security Conference*, Oct. 1990.

[26] D. Bell and L. LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation," Technical Report MTR-2997, MITRE Corporation, Bedford, MA, July 1975.

[27] E. Bertino, S. Jojodia, and P. Samarati, "Access Controls in Object-Oriented Database Systems: Some Approaches and Issues," in *Advanced Database Concepts and Research Issues* (N. Adam and B. Bhargava, eds.), vol. 759 of *Lecture Notes in Computer Science*, Springer-Verlag, 1993.

[28] E. Bertino and L. Martino, *Object-Oriented Database Systems: Concepts and Architectures.* Addison-Wesley ISBN 0 201 624397, 1993.

[29] E. Bertino, F. Origgi, and P. Samarati, "A New Authorization Model for Object-Oriented Databases," in *Database Security VIII (A-60)* (J. Biskup, M. Morgenstern, and C. E. Landwehr, eds.), pp. 199–222, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1994.

[30] E. Bertino and H. Weigand, "An Approach to Authorization Modeling in Object-Oriented Database Systems," in *Data & Knowledge Engineering* (P. P. Chen and R. P. V. de Riet, eds.), pp. 1–29, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1994.

[31] E. Bertino, "A View Mechanism for Object-Oriented Databases," in *Proceedings 3rd International Conference on Extending Data Base Technology (EDBT)*, vol. 580 of *Lecture Notes in Computer Science*, (Vienna, Austria), pp. 136–151, Springer-Verlag, Mar. 1992.

[32] E. Bertino, "Data Hiding and Security in Object-Oriented Databases," in *Proceedings of the Eight International Conference on Data Engineering* (F. Golshani, ed.), pp. 338–347, IEEE Computer Society Press, 1992.

[33] E. Bertino and S. Jajodia, "Modeling Multilevel Entities Using Single Level Objects," in *Proceedings of the Deductive and Object-Oriented Databases,*

*Third International Conference, DOOD'93*, vol. 760 of *Lecture Notes in Computer Science*, (Phoenix, Arizona, USA), pp. 415–428, Springer-Verlag, Dec. 1993.

[34] E. Bertino and P. Samarati, "Research Issues in Discretionary Authorizations for Object Bases," in *OOPSLA'93 Workshops on Security for Object-Oriented Systems*, pp. 183–199, ACM SIGPLAN Notices, Oct. 1993.

[35] L. J. Binns, "Inference and Cover Stories," in *Database Security VI* (B. M. Thuraisingham and Landwehr, eds.), pp. 169–178, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1993.

[36] L. J. Binns, "Inference Through Secondary Path Analysis," in *Database Security VI* (B. M. Thuraisingham and Landwehr, eds.), pp. 195–208, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1993.

[37] W. E. Boebert, W. D. Young, R. Y. Kain, and S. A. Hansohn, "Security Ada Target: Issues, System Design and Verfication," in *Proceedings of the 1985 IEEE symposium on security and privacy*, pp. 176–184, IEEE Computer Society Press, Apr. 1985.

[38] N. Boulahia-Cuppens, F. Cuppens, A. Gabillon, and K. Yazdanian, "Decomposition of Multilevel Objects in an Object-Oriented Database," in *Computer Security ESORICS 94, Third European Symposium on Research in Computer Security*, vol. 875 of *Lecture Notes in Computer Science*, pp. 375–402, Springer-Verlag, Nov. 1994.

[39] N. Boulahia-Cuppens, F. Cuppens, A. Gabillon, and K. Yazdanian, "Virtual View Model to Design a Secure Object-Oriented Database," in *Proceedings of the 17th National Computer Security Conference*, vol. 2, (Baltimore, Maryland), pp. 66–76, Oct. 1994.

[40] A. W. Brown, *Object-Oriented Database Applications in Software Engineering*. McGraw-Hill, 1991.

[41] K. B. Bruce and P. Wegner, "An Algebraic Model of Subtype and Inheritance," in *Advances in database programming languages* (F. Bancilhon and P. Buneman, eds.), pp. 75–96, ACM Press ; Reading, Mass.: Addison-Wesley Co., 1990.

[42] L. J. Buczkowski, "Database Inference Controller," in *Database Security III* (D. L. Spooner and Landwehr, eds.), pp. 311–322, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1990.

[43] P. Butterworth, A. Otis, and J. Stein, "The Gemstone Object Database Management System," *Communications of the ACM*, vol. 34, pp. 65–77, Oct. 1991.

[44] Canadian System Security Centre Communications Security Establishment Government of Canada, *The Canadian Trusted Computer Product Evaluation Criteria (CTCPEC).* Version 3.0e, ftp.cse.dnd.ca, Jan. 1993.

[45] M. Carey, D. DeWitt, and S. Vandenberg, "A Data Model and Query Language for EXODUS," in *Proceedings of the ACM SIGMOD Conference*, (Chicago), June 1988.

[46] S. Castano, M. Fugini, G. Martella, and P. Samarati, *Database Security.* Addison-Wesley, ACM press., 1995.

[47] R. G. G. Cattell, *Object Data Management:object-oriented and extended relational database systems.* Addison-Wesley, 1992.

[48] G. C. Chick and S. E. Tavares, "Flexible Access Control With Master Keys," in *Advances in Cryptology Proceedings of CRYPTO'89* (G. Brassard, ed.), pp. 316–322, Springer-Verlag, 1990.

[49] D. D. Clark and D. R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," in *Proceedings IEEE Computer Society Symposium on Security and Privacy*, (Oakland, CA.), pp. 184–194, 1987.

[50] B. G. Claybrook, "Using Views in A Multilevel Secure Database Management System," in *Proceedings of the 1983 IEEE symposium on security and privacy*, pp. 4–17, IEEE Computer Society Press, Apr. 1983.

[51] B. G. Claybrook, A. M. Claybrook, and J. Williams, "Defining Database Views as Data Abstractions," *IEEE Transactions on Software Engineering*, vol. 11, Jan. 1985.

[52] E. F. Codd, "A Relation Model for Large Shared Data Banks," *Communications of the ACM*, vol. 13, pp. 377–387, June 1970.

[53] E. F. Codd, "Extending the Database Relational Model to Capture More Meaning," *ACM Transactions on Database Systems*, vol. 4, Dec. 1979.

[54] Commission of the European Communities, "Information Technology Security Evaluation Criteria," tech. rep., Brussels, Sept. 1992.

[55] Committee on Multilevel Data Management Security, "Multilevel Data Management Security," technical report, Washington, D. C.: Air Force Studies Board, National Research Council, National Academy Press, 1983. For Official Use Only.

[56] R. W. Conway, W. L. Maxwell, and H. L. Morgan, "On the Implementation of Security Measures in Information Systems," *Communications of the ACM*, vol. 15, Apr. 1974.

[57] S. S. Cosmadakis and C. H. Papadimitriou, "Updates of Relational Views," *Journal of the ACM*, vol. 31, Oct. 1984.

[58] O. Costich, "Transaction Processing Using an Untrasted Scheduler in a Multilevel Database with Replicated Architecture," in *Database Security V* (S. Jajodia and C. Lanwehr, eds.), pp. 173–191, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1992.

[59] O. Costich and J. McDermott, "A Multilevel Transaction Problem for Multilevel Secure Database Systems and its Solution for Replicated Architecture," in *Proceedings of IEEE computer Society Symposium on Research in Security and Privacy*, (Oakland, CA.), pp. 192–203, IEEE Computer Society Press, May 1992.

[60] F. Cuppens, "A Model Logic Framework to Solve Aggregation Problems," in *Database Security V* (C. E. Landwehr and S. Jajodia, eds.), pp. 315–333, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1992.

[61] C. J. Date, *An Introduction to Database Systems*, vol. I. Addison-Wesley, 5 ed., 1991.

[62] G. I. Davida and Y. Yeh, "Cryptographic Relational Algebra," in *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, pp. 111–116, IEEE Computer Society Press, 1982.

[63] U. Dayal, "Queries and views in an Object-Oriented Data Model," *International Workshop on Data Base Programming Languages*, vol. 2, 1989.

[64] U. Dayal and P. A. Bernstein, "On the Correct Translation of Update Operations on Relational Views," *ACM Transactions on Database Systems*, vol. 8, Sept. 1982.

[65] D. E. Denning, *Cryptography and Data Security.* Addison-Wesley Publishing Company, 1983.

[66] D. E. Denning, "Cryptographic Checksums for Multilevel Database Security," in *Proceedings of the 1984 IEEE symposium on security and privacy*, pp. 52–61, IEEE Computer Society Press, Apr. 1984.

[67] D. E. Denning, "Commutative Filters for Reducing Inference Threats in Multilevel Database Systems," in *Proceedings of the 1985 IEEE symposium on security and privacy*, pp. 134–146, IEEE Computer Society Press, Apr. 1985.

[68] D. E. Denning, S. G. Akl, M. Heckman, T. F. Lunt, M. Morgenstern, P. G. Neumann, and R. R. Schell, "Views for Multilevel Database Security," *IEEE Transactions on Software Engineering*, vol. SE-13, pp. 129–140, Feb. 1987.

[69] D. E. Denning and T. F. Lunt, "A Multilevel Relational Data Model," in *Proceedings of Symposium on Computer Security and Privacy*, (Oakland, CA.), pp. 220–234, IEEE Computer Society Press, 1987.

[70] D. E. Denning and J. Schlorer, "Inference Control for Statistical Databases," *Computer*, vol. 16, pp. 69–82, July 1983.

[71] D. E. Denning, "Database Security," *Annual Reviews Inc.*, vol. 3, pp. 1–22, 1988.

[72] D. E. Denning, "An Evolution of Views," in *Discussions of topics presented at a Workshop held at the Vallombrosa, Conference and Retreat Centre, Research Directions in Database Security* (T. F. Lunt, ed.), (Menlo Park, CA May 1988), pp. 91-95, Springer-Verlag, 1992.

[73] Department of Defense, "Department of Defense Trusted Computer System Evaluation Criteria," Technical Report DOD 5200.28-STD, Department of Defense, Dec. 1985.

[74] O. Deux, "The $O_2$ System," *Communications of the ACM*, vol. 34, pp. 34–48, Oct. 1991.

[75] K. Dittrich, "Object-Oriented Database Systems: The Notations and Issues," in *Proceedings of the First International Workshop on Object-Oriented Database Systems*, (Pacific Grove, CA.), IEEE Computer Society Press, Sept. 1986.

[76] K. R. Dittrich, W. Gotthard, and P. C. Lockemann, "Complex Entities for Engineering Applications," in *Entity-Relationship Approach* (S. Spaccapietra, ed.), North-Holland, 1987.

[77] K. R. Dittrich, W. Gotthard, and P. C. Lockemann, "DAMOKLES - The Database System for the UNIBASE Software Engineering Environment," in *IEEE Data Engineering*, vol. 10(1), 1987.

[78] K. R. Dittrich, M. Hartig, and H. Pfefferle, "Discretionary Access Control in Structurally Object-Oriented Database Systems," in *Database Security II: Status and Prospects* (C. E. Landwehr, ed.), pp. 105–121, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1989.

[79] P. A. Dwyer, G. D. Jelatis, and M. B. Thuraisingham, "Multilevel Security in Database Management Systems," *Computers & Security*, vol. 6, pp. 252–260, June 1987.

[80] D. B. Faatz and D. L. Spooner, "Discretionary Access Control in Object-Oriented Engineering Database Systems," in *Database Security IV* (S. Jajodia and C. Lanwehr, eds.), pp. 73–83, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1991.

[81] E. B. Fernandez, R. C. Summers, and C. D. Coleman, "An Authorization Model for a Shared Database," in *Proceedings of the 1975 ACM SIGMOD International Conference*, ACM Press, 1975.

[82] E. B. Fernandez, R. C. Summers, and C. Wood, *Database Security and Integrity.* Addison-Wesley Publishing Company, 1981.

[83] E. B. Fernandez, J. Wu, and M. H. Fernandez, "User Group Structures in Object-Oriented Database Authorization," in *Database Security VIII (A-60)* (J. Biskup, M. Morgenstern, and C. E. Landwehr, eds.), pp. 57–76, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1994.

[84] D. Ferrailo and R. Kuhn, "Role-based Access Controls," in *Proceedings of the 15th National Computer Security Conference*, vol. II, (Baltimore, Maryland), pp. 554–563, Oct. 1992.

[85] D. F. Ferraiolo, J. A. Cigini, and D. R. Kuhn, "Role-Based Access Control (RBAC): Features and Motivations," in *Proceedings of The 11th Annual Computer Security Applications Conference*, (New Orleans, USA), pp. 241–248, IEEE Computer Society Press, Dec. 1995.

[86] A. Furtato and M. Casanova, "Updating Relational Views," in *Query Processing in Database Systems* (Kim, Reiner, and Batory, eds.), Lecture Notes in Computer Science, Springer-Verlag, 1985.

[87] G. E. Gajnak, "Some Result from the Entity/Relationship Multilevel Secure DBMS Project," in *Discussions of topics presented at a Workshop held at the Vallombrosa, Conference and Retreat Centre, Menlo Park, CA May 1988, Research Directions in Database Security* (T. Lunt, ed.), pp. 173–190, Springer-Verlag, 1992.

[88] C. Garvey and A. Wu, "ASD-Views," in *Proceedings of the 1988 IEEE Symposium on Security and Privacy, Washington*, pp. 85–95, IEEE Computer Society Press, 1988.

[89] T. D. Garvey, T. F. Lunt, X. Qian, and M. E. Stickel, "Toward a Tool to Detect and Eliminate Inference Problems in the Design Of Multilevel Databases," in *Database Security VI* (B. M. Thuraisingham and Landwehr, eds.), pp. 149–167, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1993.

[90] L. Giuri, "A New Model for Role-Based Access Control," in *Proceedings of The 11th Annual Computer Security Applications Conference*, (New Orleans, USA), pp. 249–255, IEEE Computer Society Press, Dec. 1995.

[91] A. Goldberg and D. Robson, *SMALLTALK-80 The Language and Implementation.* Addison-Wesley, Reading, MA, 1983.

[92] L. Gong, *Cryptographic Protocols for Distributed Systems.* PhD thesis, Jesus College, University of Cambridge, United Kindom, 1990.

[93] G. S. Graham and P. J. Denning, "Protection- principles and practice," in *Proceedings of the Spring Joint Computer Conference*, vol. 40, (Montvale, New York), AFIPS Press, 1972.

[94] R. D. Graubart and K. J. Duffy, "Design Overview for Retrofitting Integrity-lock Architecture onto a Commercial DBMS," in *Proceedings of the 1985 IEEE symposium on security and privacy*, pp. 147–159, IEEE Computer Society Press, Apr. 1985.

[95] N. A. B. Gray, *Programming with class : a practical introduction to object-oriented programming with C++.* Wiley, 1994.

[96] E. Gudes, H. S. Koch, and F. A. Stahl, "The Application of Cryptography for Database Security," *National Computer Conference*, pp. 97–107, 1976.

[97] E. Gudes, H. Song, and E. B. Fernandez, "Evaluation of Negative, Predicate, and Instance-based Authorization in Object-Oriented Databases," in *Database Security IV* (S. Jajodia and C. Lanwehr, eds.), pp. 85–98, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1991.

[98] J. T. Haigh, R. C. O'Brien, P. D. Stachour, and D. L. Toups, "The LDV Approach to Database Security," in *Database Security III* (D. L. Spooner and Landwehr, eds.), pp. 323–339, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1990.

[99] J. T. Haigh, R. C. O'Brien, and D. J. Thomsen, "The LDV Secure Relational DBMS Model," in *Database Security IV* (S. Jajodia and C. Lanwehr, eds.), pp. 265–279, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1991.

[100] B. Hailpern and B. Ossher, "Extending Objects to Support Multiple Interfaces and Access Control," *IEEE Transactions on Software Engineering*, vol. 16, no. 11, pp. 1247–1257, 1990.

[101] T. Hardjono, Y. Zheng, and J. Seberry, "A New Approach to Database Authentication," in *Research and Practical Issues in Databases: Proceedings of the Third Australian Database Conference (Database'92)*, pp. 334–342, 1992.

[102] T. Hardjono, Y. Zheng, and J. Seberry, "Database authentication revisited," *Computers & Security*, vol. 13, no. 7, pp. 573–580, 1994.

[103] T. Hardjono, *Applications of Cryptography for the Security of Database and Distributed Database Systems.* PhD thesis, University College, University of NSW, Sydney, Australia, 1991.

[104] L. Harn, Y.-R. Chien, and T. Kiesler, "An Extended Cryptographic Key Generation Scheme For Multilevel Data Security," in *Proceedings of the IEEE Computer Society Symposium on Security and Privacy*, (Oakland, CA.), IEEE Computer Society Press, May 1990.

[105] L. Harn and H. Y. Lin, "A Cryptographic Key Generation Scheme for Multilevel Data Security," *Computers & Security*, vol. 9, no. 6, pp. 539–546, 1990.

[106] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman, "Protection in Operating Systems," *Communications of the ACM*, vol. 19, Aug. 1976.

[107] S. Heiler and S. Zdonik, "Object Views: Extending the Vision," in *Proceedings 6th Data Engineering Conference*, pp. 86–93, IEEE Computer Society Press, 1990.

[108] W. R. Herndon, "An Interpretation of Clark-Wilson for Object-Oriented DBMSs," in *Database Security VII* (T. F. Keefe and Landwehr, eds.), pp. 65–85, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1994.

[109] H. R. Hinke, C. Garvey, N. Jensen, J. Wilson, and A. Wu, "A1 Secure DBMS Design," in *Proceedings of the 11th National Computer Security Conference*, (Baltimore, Maryland), Oct. 1988.

[110] T. Hinke, "Inference Aggregation Detection in Database Management Systems," in *Proceedings of the IEEE Symposium on Security and Privacy*, (Oakland, CA.), IEEE Computer Society Press, Apr. 1988.

[111] T. H. Hinke and H. S. Delugach, "AERIE: An Inference Modeling and Detection Approach For Databases," in *Database Security VI* (B. M. Thuraisingham and Landwehr, eds.), pp. 179–193, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1993.

[112] D. K. Hsiao, M. J. Kohler, and S. W. Stround, "Query Modifications as Means of Controlling Accesses to Multilevel Secure Databases," in *Database Security IV* (S. Jajodia and C. Lanwehr, eds.), pp. 221–240, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1991.

[113] H. Ishikawa, Y. Izumida, N. Kawato, and T. Hayashi, "An Object-Oriented Database System and its View Mechanism for Schema Integration," in *Future Databases'92, Proceedings 2nd Far-East Workshop on Future Database Systems* (Q. Chen, Y. Kambayashi, and R. Sacks-Davis, eds.), vol. 3 of *Advanced Database Research and Development Series*, (Kyoto, Japan), pp. 194–200, Apr. 1992.

[114] S. Jajodia and B. Kogan, "Integrating an Object-Oriented Data Model with Multilevel Security," *IEEE Computer Society Press*, pp. 76–85, 1990.

[115] S. Jajodia and R. Sandhu, "A Novel Decomposition of Multilevel Relations into Single-level Relations," in *IEEE Symposium on Research in Security and Privacy*, (Oakland, CA.), 1991.

[116] S. Jajodia and R. Sandhu, "Polyinstantiation Integrity in Multilevel Relations Revisited," in *Database Security IV* (S. Jajodia and C. Lanwehr, eds.), pp. 297–307, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1991.

[117] S. Jajodia and R. Sandhu, "Toward a Multilevel Relational Data Model," in *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data* (J. Clifford and R. King, eds.), SIGMOD RECORD, ACM Press, 1991.

[118] S. Jajodia and B. Kogan, "Transaction Processing in Multilevel Secure Databases Using Replicated Architecture," in *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, pp. 360–368, IEEE Computer Society Press, 1990.

[119] I. E. Kang and T. F. Keefe, "Recovery Management for Multilevel Secure Database Systems," in *Database Security VI* (B. M. Thuraisingham and Landwehr, eds.), pp. 225–249, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1993.

[120] T. F. Keefe, M. B. Thuraisingham, and W. T. Tsai, "Secure Query Processing Strategies," *IEEE Computer Society Press*, vol. 22, pp. 63–70, Mar. 1989.

[121] T. F. Keefe and W. T. Tsai, "Security Model Consistency in Secure Object-Oriented Systems," in *5th Annual Computer Security Applications Conference Tucson, Arizona*, pp. 290–298, IEEE Computer Society Press, Dec. 1989.

[122] T. F. Keefe and W. T. Tsai, "Prototyping the SODA Security Model," in *Database Security II* (D. L. Spooner and C. E. Landwehr, eds.), pp. 211–235, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1990.

[123] T. F. Keefe, W. T. Tsai, and M. B. Thuraisingham, "A Multilevel Security Model For Object-Oriented Systems," in *Proceedings of the 11th National Computer Security Conference*, (Baltimore, Maryland), pp. 1–9, Oct. 1988.

[124] W. Kim, N. Ballou, J. Banerjee, H.-T. Chou, J. F. Garza, and D. Woelk, "Features of The ORION Object-Oriented Database System," in *Proceedings of the 13th International VLDB Conference*, pp. 319–329, 1987.

[125] W. Kim and F. H. L. (Eds.), *Object-Oriented Concepts, Databases, and Applications.* Addison-Wesley, Reading, Massachusetts, 1988.

[126] W. Kim, "Object-Oriented Databases: Definition and Research Directions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, pp. 327–341, Sept. 1990.

[127] C. Laferriere, "A Discussion of Implementation Strategies for Secure Database Management Systems," *Computers & Security*, vol. 9, pp. 235–244, 1990.

[128] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb, "The Objectstore Database System," *Communications of the ACM*, vol. 34, pp. 50–63, Oct. 1991.

[129] B. W. Lampson, "protection," in *Proceedings of the 5th Princeton Symposium on Information Science and Systems*, (Reprinted in ACM operating System Review, Vol. 8 (1). Jan. 1974), Mar. 1971.

[130] M. M. Larrondo-Petrie, E. Guides, H. Song, and E. B. Fernandez, "Security Policies in Object-Oriented Databases," in *Database Security III* (D. L. Spooner and Landwehr, eds.), pp. 257–269, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1990.

[131] C. Lecluse, P. Richard, and F. Velez, "$O_2$, an Object-Oriented Data Model," in *Advances in Database Programming Languages* (F. Bancilbon and P. Buneman, eds.), (New York), pp. 257–276, ACM Press ; Reading, Mass. : Addison-Wesley Pub., 1990.

[132] T. Y. Lin, "Inference Secure Multilevel Databases," in *Database Security VI* (B. M. Thuraisingham and Landwehr, eds.), pp. 317–333, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1993.

[133] G. M. Lohman, B. Lindsay, H. Pirahesh, and K. B. Schiefer, "Extensions to Starburt: Objects, Types, Functions, and Rules," *Communications of the ACM*, vol. 34, pp. 94–109, Oct. 1991.

[134] M. E. S. Loomis, "Object and Relational Technologies," *Object Magazine*, pp. 35–43, Jan. 1993.

[135] M. E. S. Loomis, T. Atwood, R. Cattell, J. Duhl, G. Ferran, and D. Wade, "ODBMS: The ODMG object model," *Journal of Object-Oriented Programming*, pp. 64–69, June 1993.

[136] R. Lorie and W. Plouffe, "Complex Objects and their Use in Design Transaction," in *Proceedings of the ACM SIGMOD Conference Database Week*, 1983.

[137] J. J. Lu, G. Moerkotte, J. Schue, and V. S. Subrahmanian, "Efficient Maintenance of Materialized Mediated Views," in *Proceedings of the 1995 ACM SIGMOD International Conference on Managemant of Data*, vol. 24(2), (San Jose, California), pp. 340–351, SIGMOD RECORD, ACM Press, May 1995.

[138] T. F. Lunt, "Access Control Policies For Database Systems," in *Database Security II: Status and Prospects* (C. E. Landwehr, ed.), pp. 41–52, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1989.

[139] T. F. Lunt, "Aggregation and Inference: Facts and Fallacies," in *Proceedings of the IEEE Computer Society Symposium on Security and Privacy*, (Oakland, CA), pp. 102–109, IEEE Computer Society Press, May 1989.

[140] T. F. Lunt, "Multilevel Security for Object-Oriented Database Systems," in *Database Security III* (D. L. Spooner and Landwehr, eds.), pp. 199–209, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1990.

[141] T. F. Lunt, "SeaView," in *Discussions of topics presented at a Workshop held at the Vallombrosa, Conference and Retreat Centre, Menlo Park, CA May 1988, Research Directions in Database Security* (T. F. Lunt, ed.), pp. 2–32, Springer-Verlag, 1992.

[142] T. F. Lunt and P. K. Boucher, "The SeaView Prototype: Project Summary," in *Proceedings of the 17th National Computer Security Conference*, vol. 2, (Baltimore, Maryland), pp. 88–102, Oct. 1994.

[143] T. F. Lunt, D. E. Denning, R. R. Schell, W. R. Shockley, and M. Heckman, "The SeaView Security Model," *IEEE Transactions on Software Engineering*, June 1990.

[144] T. F. Lunt and E. B. Fernandez, "Database Security," *SIGMOD RECORD, ACM Press*, vol. 19, pp. 90–97, Dec. 1990.

[145] N. M. Mattos, K. Meyer-Wegener, and B. Mitschang, "Grand Tour of Concepts for Object-Orientation from a Database Point of View," *Data & Knowledge Engineering*, vol. 9, pp. 321–352, 1992.

[146] C. McCollum and L. Notargiacomo, "Distributed Concurrency Control with Optional Data Replication," in *Database Security V* (S. Jajodia and C. Lanwehr, eds.), pp. 149–173, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1992.

[147] J. Mclean, "The Specification and Modeling of Computer Security," *Computer*, Jan. 1990.

[148] C. Meadows, "Constructing Containers Using A Multilevel Relational Data Model," in *Database Security III* (D. L. Spooner and Landwehr, eds.), pp. 127–141, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1990.

[149] J. K. Millen and T. F. Lunt, "Security for Object-Oriented Database Systems," in *Proceedings of IEEE computer Society Symposium on Research in Security and Privacy*, (Oakland, CA.), pp. 260–272, IEEE Computer Society Press, May 1992.

[150] M. Morgenstern, "A Security Model for Multilevel Object With Bidirectional Relationships," in *Database Security IV: Status and Prospects* (S. Jajodia and C. E. Landwehr, eds.), pp. 53–71, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1991.

[151] M. Naor and M. Yung, "Universal one-way hash functions and their cryptographic applications," in *Proceedings of the 21st ACM Symposium on Theory of Computing*, pp. 33–43, ACM Press, 1989.

[152] L. Notargiacomo, B. T. Blaustein, and C. D. McCollum, "A Model of Integrity and Dynamic Separation of Duty for a Trusted DBMS," in *Database Security VII* (T. F. Keefe and Landwehr, eds.), pp. 237–246, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1994.

[153] L. Notargiacomo, "Metadata and View Classification," in *Discussions of topics presented at a Workshop held at the Vallombrosa, Conference and Retreat Centre, Research Directions in Database Security* (T. F. Lunt, ed.), (Menlo Park, CA May 1988), pp. 243–247, Springer-Verlag, 1992.

[154] G. M. Nyanchama, *Commercial Integrity, Roles and Object Orientation.* PhD thesis, Department of Computer Science, Faculty of Graduate Studies , University of Western Ontario, London, Ontario, Canada, 1994.

[155] M. Nyanchama and S. Osborn, "Role-Based Security, Object Oriented Databases and Separation of Duty," *SIGMOD RECORD, ACM Press*, vol. 22, pp. 45–51, Dec. 1993.

[156] Object Database Corporation, Cambridge, Massachusetts, *GBase Reference Manual*, 1990.

[157] Object Design Inc, Burlington, MA 01803, *Objectstore Reference Manual and Objectstore User Guide*, 1990.

[158] Object Design Inc, Burlington, MA 01803, *Objectstore Technical Overview*, 1990.

[159] Objectivity, Inc., Menlo Park, CA, *Objectivity Database System Overview*, 1990.

[160] M. S. Olivier and S. H. V. Solms, "A Taxonomy for Secure Object-Oriented Databases," *ACM Transactions on Database Systems*, vol. 19, pp. 3–46, Mar. 1993.

[161] M. S. Olivier, *Secure Object-Oriented Databases.* PhD thesis, Computer Science, Faculty of Natural Sciences, RAND Afrikaans University, johannesburg, South Africa, Dec. 1991.

[162] F. Olken and D. Rotem, "Maintenance of Materialized Views of Sampling Queries," in *Proceedings of the Eight International Conference on Data Engineering* (F. Golshani, ed.), pp. 632–641, IEEE Computer Society Press, 1992.

[163] I. Olson and A. Marshall, "Computer Access Control Policy Choices," *Computers & Security*, vol. 9, no. 8, 1990.

[164] Ontologic, Inc., Billerica, Massachusetts, *ONTOS Reference Manual*, 1989.

[165] R. Pascale and J. R. M. Enerney, "Using THETA to Implement Access Control for Separation of Duties," in *Proceedings of the 17th National Computer Security Conference*, vol. 2, (Baltimore, Maryland), pp. 47–55, Oct. 1994.

[166] C. P. Pfleeger, *Security in Computing.* Prentice-Hall, Inc., 1989.

[167] W. J. Premerlani, M. R. Blaha, J. E. Rumbaugh, and T. A. Varwing, "An Object-Oriented Relational Database," *Communications of the ACM*, vol. 33, Nov. 1990.

[168] X. Qian, M. E. Stickel, P. D. Karp, T. F. Lunt, and T. D. Garvey, "Detection and Elimination of Inference Channels in Multilevel Relational Database Systems," in *Proceedings 1993 IEEE Symposium on Research in Security and Privacy*, pp. 196–205, IEEE Computer Society Press, 1993.

[169] F. Rabitti, E. Bertino, W. Kim, and D. Woelk, "A Model of Authorization for Next-Generation Database Systems," *ACM Transactions on Database Systems*, vol. 16, pp. 88–131, Mar. 1991.

[170] F. Rabitti, D. Woelk, and W. Kim, "A Model of Authorization for Object-Oriented and Semantic Databases," in *Proceedings of International Conference on Extending Database Technology*, vol. 303 of *Lecture Notes in Computer Science*, pp. 231–250, Springer-Verlag, Mar. 1988.

[171] R. L. Rivest, A. Shamir, and L. Adleman, "A Method For Obtaining Digital Signatures And Public-Key Cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–128, 1978.

[172] R. L. Rivest, "The MD4 Message Digest Algorithm," in *Advances in Cryptology, Proceedings of CRYPTO'90*, pp. 281–291, Springer-Verlag, 1990.

[173] R. L. Rivest, "The MD5 Message Digest Algorithm." MIT Laboratory for Computer Science and RSA Data Security, Inc., Request for Comments (RFC), 1992.

[174] J. Rumbaugh, M. Blaba, W. Premerlani, F. Eddy, and W. Larensen, *Object-Oriented Modeling and Design.* Printice Hall Inc., 1991.

[175] E. A. Rundensteiner, *Object-Oriented Views: A Novel Approach for Tool Integration in Design Environments.* Technical report 92-83, University of California, Irvine, USA, 1992.

[176] D. Russell and G. T. Gangemi, *Computer Security Basic.* O'Reill & Associates, 1991.

[177] S. Wiseman et. al., "The Trouble with Secure Database," in *Proceedings of MILCOMP 89*, pp. 164–170, Sept. 1989.

[178] R. S. Sandhu, "Cryptographic Implementation of A Tree Hierarchy For Access Control," *Information Processing Letter*, vol. 27, no. 2, pp. 95–98, 1988.

[179] R. S. Sandhu, "Expressive Power of the Schematic Protection Model," *Journal of Computer Security*, vol. 1, 1992.

[180] R. Sandhu, "Separation of Duties in Computerized Information Systems," in *Database Security IV* (S. Jajodia and C. Lanwehr, eds.), pp. 179–189, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1991.

[181] R. S. Sandhu, "The Typed Access Matrix Model," in *Proceedings of IEEE computer Society Symposium on Research in Security and Privacy*, (Oakland, CA.), pp. 122–136, IEEE Computer Society Press, May 1992.

[182] R. S. Sandhu, "Role-Based Access Control: A position Statement," in *Proceedings of the 17th National Computer Security Conference*, vol. 2, (Baltimore, Maryland), p. 492, Oct. 1994.

[183] R. S. Sandhu and H. Feinstein, "A Three TIER Architecture for Role-Based Access Control," in *Proceedings of the 17th National Computer Security Conference*, vol. 2, (Baltimore, Maryland), pp. 34–46, Oct. 1994.

[184] R. S. Sandhu and S. Jajodia, "Referential Integrity in Multilevel Security Databases," in *Proceedings of the 16th National Computer Security Conference*, (Baltimore, Maryland), pp. 39–52, Sept. 1993.

[185] R. S. Sandhu and P. Samarati, "Access Control: Principles and Practice," *IEEE Communications Magazine*, vol. 9, pp. 40–48, Sept. 1994.

[186] R. S. Sandhu, "Recognizing Immediacy in an N-Tree Hierarchy and Its Application to Protection Groups," *IEEE Transactions on Software Engineering*, vol. 15, pp. 1518-1525, Dec. 1989.

[187] R. R. Schell and T. F. Tao, "Microcomputer-based Trusted Systems for Communication and Workstation Applications," in *Proceedings of The 7th DOD/NBS Computer Security Conference*, pp. 277–290, Sept. 1984.

[188] L. M. Schlipper, J. Filsinger, and V. M. Doshi, "A Multilevel Secure Database Management System Benchmark," in *Proceedings of the 15th National Computer Security Conference*, vol. II, (Baltimore, Maryland), pp. 399–408, Oct. 1992.

[189] M. H. Scholl, C. Laasch, and M. Tresch, "Views in Object-Oriented Databases," in *Proceedings 2nd Workshop on Foundations of Models and Languages of Data and Objects*, pp. 37–58, Sept. 1990.

[190] M. H. Scholl, C. Laasch, and M. Tresch, "Updatable Views in Object-Oriented Databases," in *Proceedings of the Deductive and Object-Oriented Databases, Second International Conference, DOOD'91* (C. Delobel, M. Kifer, and Y. Masunga, eds.), vol. 566 of *Lecture Notes in Computer Science*, (München, FRG), pp. 189–207, Springer-Verlag, Dec. 1991.

[191] J. Seberry and J. Pieprzyk, *CRYPTOGRAPHY: An Introduction to Computer Security, Advances in Computer Science Series.* Prentice Hall Inc., 1989.

[192] J. J. Shilling and P. F. Sweeney, "Three Steps to Views: Extending the Object-Oriented Paradigm," in *Proceedings International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, (New York), pp. 353–361, ACM Press, Oct. 1989.

[193] W. R. Shockley, "Implementing The Clark/Wilson Integrity Policy Using Current Technology," in *Proceedings of the 11th National Computer Security Conference*, (Baltimore, Maryland), pp. 29–37, Oct. 1988.

[194] A. Silberschatz, M. Stonebraker, and J. Ullman, "Database Systems: Achievements and Opportunities," *Communications of the ACM*, vol. 34, pp. 110–120, Oct. 1991.

[195] G. Smith, "Modeling Security-Relevant Data Semantics," in *Proceedings of the IEEE Computer Society Symposium on Security and Privacy*, (Oakland, CA.), IEEE Computer Society Press, May 1990.

[196] G. W. Smith, "Identifying and Representing the Security Semantics of an Application," in *Proceedings of the Fourth Aerospace Computer Security Applications Conference*, , Dec. 1988.

[197] J. M. Smith and D. C. P. Smith, "Database Abstraction: Aggregation and Generalization," *ACM Transactions on Database Systems*, vol. 2, pp. 105–133, June 1977.

[198] B. Sowerbutts and S. Cordingley, "Data Base Architectonics and Inferential Security," in *Database Security IV* (S. Jajodia and Landwehr, eds.), pp. 309–324, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1991.

[199] D. L. Spooner, "The Impact of Inheritance On Security In Object-Oriented Database Systems," in *Database Security II* (C. E. Landwehr, ed.), pp. 141–150, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1989.

[200] P. D. Stachour and B. Thuraisingham, "Design of LDV: A Multilevel Secure Relational Database Management System," *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, pp. 190–209, June 1990.

[201] P. Stachour, "LOCK Data Views," in *Discussions of topics presented at a Workshop held at the Vallombrosa, Conference and Retreat Centre, Research Directions in Database Security* (T. F. Lunt, ed.), (Menlo Park, CA May 1988), pp. 63–80, Springer-Verlag, 1992.

[202] M. Stonebraker and G. Kemnitz, "The POSTGRES Next Generation Database Management System," *Communications of the ACM*, vol. 34, pp. 78–92, Oct. 1991.

[203] M. Stonebraker and L. A. Rowe, "The design of POSTGRES," in *Proceedings of the ACM SIGMOD conference*, (Washington D.C.), 1986.

[204] T.-A. Su, "Multivalued Dependency Inference in Multilevel Relational Database Systems," in *Database Security III* (D. L. Spooner and Landwehr, eds.), pp. 293–300, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1990.

[205] S. M. Thatte, "A Modular and Open Object-Oriented Database System," *SIGMOD RECORD, ACM Press*, vol. 20, Mar. 1991.

[206] R. K. Thomas and R. S. Sandhu, "Discretionary Access Control in Object-Oriented Database: Issues and Research Directions," in *Proceedings of the 16th National Computer Security Conference*, (Baltimore, Maryland), pp. 63–74, Sept. 1993.

[207] R. K. Thomas and R. S. Sandhu, "Conceptual Foundations for a Model of Task-based Authorization," *IEEE Computer Security Foundations Workshop*, vol. 7, pp. 66–79, 1994.

[208] D. J. Thomsen, W. T. Tsai, and M. B. Thuraisingham, "Prototyping as a Research Tool for MLS/DBMS," in *Database Security II* (C. E. Landwehr,

ed.), pp. 63–84, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1989.

[209] M. B. Thuraisingham, "Security Checking in Relational Database Management System Augmented with Inference Engines," *Computers & Security*, vol. 6, Dec. 1987.

[210] M. B. Thuraisingham, "Mandatory Security in Object-Oriented Database Systems," in *Proceedings International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, (New Orleans), pp. 203–210, Oct. 1989.

[211] M. B. Thuraisingham, "Recent Developments in Database Security," in *Tutorial Proceedings of the IEEE COMPSAC Conference*, (Orlando, FL), Sept. 1989.

[212] M. B. Thuraisingham, "Towards the Design of a Secure Data/Knowledge Base Management System," *Data & Knowledge Engineering*, vol. 5, Mar. 1990.

[213] M. B. Thuraisingham, "The Use of Conceptual Structures for Handling The Inference Problem," in *Database Security V* (C. E. Landwehr and S. Jajodia, eds.), pp. 333–362, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1992.

[214] T. C. Ting, "A User-Role Based Data Security Approach," in *Database Security: Status and Prospects* (C. E. Landwehr, ed.), pp. 187–208, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1988.

[215] T. C. Ting, S. A. Demurjian, and M. Y. Hu, "Requirements, Capabilities, and Functionalities of User-Role Based Security for an Object-Oriented Design Model," in *Database Security V: Status and Prospects* (S. Jajodia and C. E. Landwehr, eds.), pp. 275–296, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1992.

[216] H.-M. Tsai and C.-C. Chang, "A Cryptographic Implementation for Dynamic Access Control in a User Hierarchy," *Computers & Security*, vol. 14, pp. 159–166, 1995.

[217] J. D. Ullman, *Principles of database and knowledge-base systems.* Rockville, Md.: Computer Science Press, 1988.

[218] Versant Object Technologies, Inc., Menlo Park, CA, *VERSANT Technical Overview*, 1990.

[219] Y. Wand, "A Proposal for a Formal Model of Objects," in *Object-Oriented Concepts, Databases, and Applications* (W. Kim and F. H. Lochovsky, eds.), pp. 537–559, Addison-Wesley, Reading, Massachusetts, ACM Press, 1989.

[220] M. N. Wegman and J. L. Carter, "New Hash Functions and Their Use in Authentication and Set Equality," *Journal of Computer and System Sciences*, vol. 22, pp. 265–279, 1981.

[221] G. Wiederhold, "Views, Objects, and Databases," *IEEE Computer*, pp. 37–44, Dec. 1986.

[222] K. Wilkinson, P. Lyngbaek, and W. Hasan, "The Iris Architecture and Implementation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, Mar. 1990.

[223] J. Wilson, "Views as the Security Objects in a Multilevel Secure Relational Database Management System," in *Proceedings of Symposium on Computer Security and Privacy*, (Oakland, CA.), IEEE Computer Society Press, Apr. 1988.

[224] S. R. Wiseman, "On the Problem of Security in Databases," in *Database Security III* (D. L. Spooner and Landwehr, eds.), pp. 301–311, Elsevier Science Publishers B. V. (North-Holland) IFIP, 1990.

[225] Y. Zheng, T. Hardjono, and J. Pieprzyk, "The Sibling Intractable Function Family (SIFF): Notation, Construction and Applications," *IEICE Transactions, Fundamentals*, vol. E76-A, pp. 4–13, Jan. 1993.

[226] Y. Zheng, J. Pieprzyk, and J. Seberry, "HAVAL- A One-Way Hashing Algorithm with Variable Lenght of Output (Extended Abstract)," in *Advances in Cryptology, Proceedings of AUSCRYPT'92*, vol. 718 of *Lecture Notes in Computer Science*, pp. 83–104, Springer-Verlag, 1992.