A Cryptographic Solution to Discretionary Access Control in Structurally Object-Oriented Databases

Ahmad Baraani-Dastjerdi
Janusz R. Getta
Josef Pieprzyk
Reihaneh Safavi-Naini
Department of Computer Science

University of Wollongong
Wollongong, NSW 2522, AUSTRALIA

e-mail: [g9272860,josef,rei,jrg].cs.uow.edu.au

July 25, 1994

Abstract

A new approach to the discretionary access control in structurally object-oriented databases is suggested. The approach is based on the use of pseudo-random functions and sibling intractable functions, rather than on the traditional access control list associated with each class. The advantage of this approach is that each class in the hierarchy of a complex object class can be associated with one access key such that it can be directly accessed by its ancestor without the need to pass through any intermediate classes. The security of the model depends on the difficulty of predicting the output of pseudo-random functions and on finding extra sibling collision for the sibling intractable function family.

1 Introduction

In recent years there has been considerable efforts in the research and development of object-oriented databases; ORION [19, 10], ONTOS [17], OBJECTSTORE [11], O_2 [5], Objectivity/DB [15], and VERSANT [20] are examples of such efforts. The driving forces behind these efforts are the advantages offered by object-oriented data models including the ability of object-oriented model to represent the composite (or complex) structure of objects and to simulate the behavior of objects through operations encapsulated in the object classes.

Object-oriented database model is much richer and more complex than the relational model, and aspects such as inheritance and composite data structures allow expression of flexible and arbitrarily specific discretionary policies. Discretionary access control (DAC) models for object-oriented databases are more complex than their relational counterparts. Hence, developing authorization models for object-oriented databases have been the subject of intensive research [7, 4, 6, 18, 2].

In this paper, we propose a cryptographic solution to discretionary access control for structurally object-oriented database system based on the application of Sibling Intractable Function Families (SIFFs) introduced by Zheng, Hardjono, and Pieprzyk [21]. The approach is based on creating an access key for each class of complex (or composite) object class and a password for the owner of the class, using pseudo-random functions and the SIFF, such that each access key of the class or the password of the owner is derivable by the ancestors or users who are granted by the owner. In our DAC system ownership property, granting, revoking, and transitivity of the rights for the class are allowed. The security of the system relies on the indistinguishability of

the output of pseudo-random functions from truly random functions and the difficulty of finding extra collision for the SIFF.

The organization of this paper is as follows. The next section discusses the basics of object-oriented database model, and the necessary definitions of the SIFF to provide necessary background for the rest of the paper. This is followed by Section 3 which discusses the proposed solution. In Section 4, we present an algorithm for the evaluation of an access request according to the proposed system. The next two sections discuss the impact of adding, deleting and/or replacing object classes or users/groups on the DAC system (Section 5-6). In Section 7, we present the possibility of extension of the proposed solution to the more general cases. The security of the proposed system is discussed in Section 8. The last section summarizes the results and gives suggestions for further research.

2 Background

In this section, we first discuss briefly the basics of structurally object-oriented data model and then consider some of the security issues and the SIFF definition.

2.1 Structurally object-oriented data model concept

The data model in a general-purpose object-oriented database system models the "real world" by defining corresponding database objects. These database objects can have complex internal structures composed of other database objects that model the details of the "real world" objects. Database objects with complex internal structures are sometimes called composite or complex objects. A composite object is the part of a conventional nested object with the IS-PART-OF relationship. The internal structure of an object is implemented using properties and can be defined as:

object $=[P_i, ..., P_k]$,

where the property P_i is a primitive data value or a reference (object identifier) to another object. A collection of similar objects forms a class. All objects of one class contain the same properties. Classes are organized into a hierarchical structure.

For example, in Figure 1, O_1 classes 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 O_1 is the root and the classes O_2, O_3, O_4 create the first level of nodes. This structure can be extended to include classes $O_6, \ldots, O_8$ included in O_2, O_3, O_4. This forms a simple composite object class that the classes $O_2, O_3, \ldots, O_8$ are parts of the class O_1 (or they are descendants of class O_1).

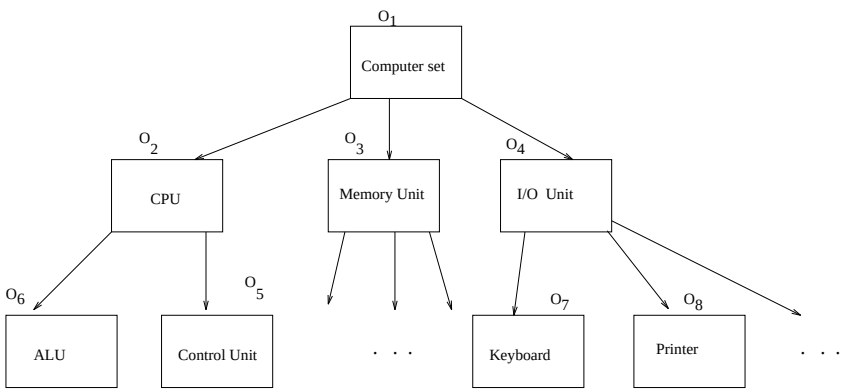

Figure 1: Computer set class.

A database system which provides mechanisms to define and to manipulate objects with complex internal structure is called *structurally object-oriented database system* (see Dittrich

[3]).

2.2 Security Considerations

In the following, we describe the main features of a discretionary access control policy for structurally object-oriented databases that allow grant and revoke of authorization rights to users or groups of users. Next we introduce a necessary cryptographic background to define our solution which is based on SIFF.

2.2.1 Visibility from above policy

Two orthogonal views of the objects in a database can be used to define access controls. The first view considers the objects as composite ones with object hierarchies where the root object corresponds to some "real world" object, and the other objects in the hierarchy define their internal structure (as shown in Figure 1). Hence, a class may have many descendants (or equivalently, a class may have many ancestors). This view is completely independent of the inheritance hierarchy for objects, and must be used in structurally object-oriented systems. In this view, access controls must be defined to deal with composite objects. This means if a user is authorized to access a class, he/she will also be authorized to access all information of the descendants of that class. But the user must not be able to access to the information of the ancestors of that class. In other words, the data must only be *visible from above* and not be visible from below (see [12]). For example, if user U_1 is authorized to access class O_4 (Figure 1), then he/she should be able to access information of O_4 and its two descendants O_7 and O_8 but he/she must not be able to access the class O_1.

One of the important policies which must be considered in the design of the access control mechanism for a structurally object-oriented database is *the visibility from above policy.*

2.2.2 Ownership vs Administration

Other policies that must be considered are the ownership versus the administration of data. The ownership defines the policy which determines the objects that the users own and the way they administer them. The policy of central administration views the enterprise as the owner of the information. Users are given necessary access rights so they can use the suitable information stored in the database.

In our DAC system, when a user creates a class, he/she will be the owner of it and will have full access to it (full authorization rights).

2.2.3 Grouping

In the discretionary access control model, users and groups of users need to be explicit parts of the security model. Discretionary access control models not only determine which subjects may obtain access to which objects, but also determine granting and revoking authorization policy to and from users and groups. Thus, sometimes it is necessary to consider a policy which allows to create user groups which are administered by their sponsors. The policy has to describe how to grant and revoke access rights to or from sponsors.

In our DAC system, users may be members of one group or groups. Groups again may become members of higher level groups, resulting in a group hierarchy as shown in Figure 2. Any group has a sponsor who administers it and the sponsor can add new members to the group or expel members from the group. Any user with grant-option right may create a new group and become its sponsor.

There is also a database administrator who, for example, admits new users to the database system.

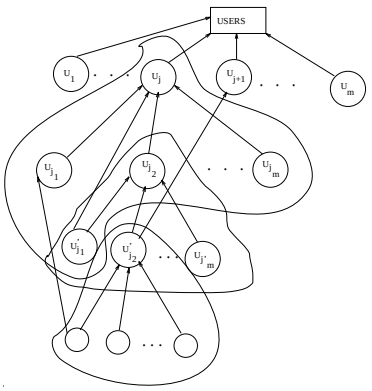

Figure 2: User groups Hierarchy.

2.2.4 Open vs Closed system

Our DAC system is selected to be a closed system, i.e. each access right must be explicitly authorized. Hence, the absence of appropriate access rights is interpreted as "access not allowed".

2.3 Sibling Intractable Function Families (SIFFs)

Zheng, Hardjono, and Pieprzyk in [21] introduced the notion of the sibling intractable function family (SIFF) which is a generalization of the concept of the universal one-way hash function defined by Naor and Yung [14]. In the class of universal one-way hash functions, the number of functions that map any collection of r distinct input strings to the same hash value is fixed. Sibling intractable function families (SIFFs) are universal one-way hash function families which have the added property that given a set of colliding sequences, it is computationally infeasible to find another sequence that collides with the initial sets. As shown in [21], the sibling intractable function families (SIFFs) can be constructed from any universal one-way hash function family.

In the following, we give the definition of SIFF. For detailed discussion and explanation of the notation, the reader is referred to the paper[21]. 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 of degree n from $\aleph$ to $\aleph$. We use $x \in_R X$ to denote an element x randomly chosen from the set X. A sibling finder F is a probabilistic polynomial time algorithm that on input $X = \{x_1, x_2, \ldots, x_i\}$ and 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, 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: Let $k = k(n)$ be a polynomial with $k(n) \geq 1$ and $H = \{H_n | n \in \aleph\}$ be a family of functions that is computable in polynomial time and is samplable. Moreover it has the collision accessibility property and maps $l(n)$-bit input into m(n)-bit output strings. Assume $X = \{x_1, x_2, \ldots, x_i\}$ be a set of i initial strings, where $1 \leq i \leq k$. H is *a k-sibling intractable function family (or k-SIFF)* if for all $1 \leq i \leq k$, any sibling finder F, any polynomial Q, and for all sufficiently large n,

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

where h is chosen randomly and uniformly from $H_n^X \subset H_n$, the set of all functions in H_n that map $x_1, x_2, \ldots, x_i$ to the same strings in $\sum^{m(n)}$, and 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.

In next section, we present a mechanism for DAC in structurally object-oriented database systems. The construction is based on the SIFF.

3 Proposed Solution

Existence of complex object, and the visibility from the above requirement introduce additional complexity to the DAC model for object oriented databases.

The visibility from above policy insures that in a composite object class, low-level classes (or descendants) can be accessed by the higher-level classes (or ancestors) and no low-level classes can access higher-level classes. Hence, the visibility from above requirement for an object-oriented database can be satisfied by employing the solutions of the hierarchical access control problem. Such solution was first proposed by Akl and Taylor [1].

Hierarchical access control problem is extensively studied [8, 16, 13, 9, 22, 21]. We propose a cryptographic solution to DAC in the structurally object-oriented databases which allows granting of the owner's right to users or groups and can be considered as an extension of Zheng et. al. [22, 21] solution to the hierarchical access control. In the following, we first present the notations and assumptions used and then we present our solution for DAC.

3.1 Notations and Assumptions

(i) O_i is the name of the i-th class and U_j denotes the j-th user's login-name. We use an $l(n)$-bit string to represent O_i and U_j. $\oplus$ denotes exclusive-or (XOR).

(ii) $F = \{F_n | n \in \aleph\}$ is a pseudo-random function family, where $F_n = \{f_K | f_K : \sum^{l(n)} \rightarrow \sum^n,\ K \in \sum^n\}$, and a n-bit string K specifies $f_K \in F_n$.

(iii) $H = \{H_n | n \in \aleph\}$ is a k-SIFF mapping n-bit input to n-bit output strings where k is a polynomial in n such that no class has more than k ancestors or no group could have more than k users.

(iv) We consider access control at the class level. This is analogous to access control at the relational (or table) level for relational database systems.

(v) A class is specified as in Figure 3. CLASS-NAME indicates the name of the class; CLASS-KEY specifies the access key of the class; HASH-FUNCTIONS indicates the list of hashing functions which can be used to derive the access key of descendants. The value of CLASS-KEY and HASH-FUNCTIONS are encrypted by a trusted party. The encryption and decryption keys are kept in a secure place and are only accessible to the trusted party. The trusted party is a separate unit in the database system.

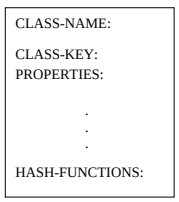

Figure 3: Object class specification.

(vi) We assume that there is an authorization table that specifies authorization rights of users to classes (see Figure 4). GRANTEE indicates the user who is authorized to access this class. CLASS-NAME specifies the class name. GRANTOR specifies the user who has authorized the user, identified by the GRANTEE, to access class identified by CLASS-NAME. GRANT-OPTION (YES or NO) indicates if the user (identified by the GRANTEE) is able to propagate the access right to other users or not. H-FUNCTION specifies hash function which can be used to derive the grantor's password. G-PSWD indicates the user's password. The value of the columns G-PSWD and H-FUNCTION are encrypted by the trusted party. The encryption and decryption keys are kept secret and are accessible to the trusted party only.

(vii) PS_j indicates the login password of the user U_j which is chosen by user U_j after the database administrator admits him/her be a user of the database system; it should be long enough and kept secret by the user.

GRANTEE	CLASS-NAME	GRANTOR	· · ·	GRANT-OPTION	H-FUNCTION	G-PSWD

Figure 4: Authorization table.

3.2 Access Key Generation For An Object Class

In order to control access to a class, we produce an access key K_i for the class O_i such that it can be derived from its ancestors' key. This access key acts as an authorization key. Every time a user requests access to a specific class, the access key of that object is computed and compared to the stored one. When they match, the access is permitted. Otherwise, the request is denied. The process of generating access key for a class is as follows.

When a user U_j (with login password PS_j) creates a class O_i, the system will complete the following steps.

Step 1. Calculate the password, $n_{j,i}$, the user U_j for the class O_i as $n_{j,i} = f_{PS_j}(U_j \oplus O_i)$.

Step 2. Select at random the access key K_i for the object class O_i ($K_i \in_R \sum^n$).

Step 3. If the class O_i *is a part of* the classes $O_{l_1}, O_{l_2}, \ldots, O_{l_p}$ (or equivalently, $O_{l_1}, O_{l_2}, \ldots, O_{l_p}$ are ancestors of O_i) and $K_{l_1}, K_{l_2}, \ldots$, and K_{l_p}, are the class O_i ancestor's keys, then select at random a hash function $h_i \in_R H_n$ such that

$$h_i(f_{n_{j,i}}(O_i)) = h_i(f_{K_{l_1}}(O_i)) = h_i(f_{K_{l_2}}(O_i)) = \ldots = h_i(f_{K_{l_p}}(O_i)) = K_i \quad (1)$$

So the ancestors of the class O_i can access it directly by generating the access key K_i from their access key K_{l_s} ($1 \leq s \leq p$).

Step 4. Make public the hash function h_i to all ancestors of object class O_i.

Step 5. Append tuple $< U_j, O_i, U_j, \ldots, \text{“}yes\text{”}, h_i, n_{j,i} >$ to the authorization table.

3.3 Propagation of Authorization

In this section, we propose a mechanism to propagate access privileges to users. The revocation of access rights is considered in Section 6.

In our discretionary access control (DAC) system, ownership is allowed. When a user creates a new class, he/she will be the owner of the class and have the full access to it. The owner can give access rights to other users by running grant-command.

Suppose that the user U_j with the password $n_{j,i}$ who can be the owner of the class O_i (or has the grant-option) wants to give m users ($U_{l_1}, U_{l_2}, \ldots, U_{l_m}$ with login passwords $PS_{l_1}, PS_{l_2}, \ldots, PS_{l_m}$, respectively) access rights to the class O_i by running the grant-command. The following steps are completed by the system.

Step 1. Calculate the password, $n_{l_s,j,i}$, of user U_{l_s} for the object O_i granted by the user U_j as $n_{l_s,j,i} = f_{PS_{l_s}}(U_j \oplus O_i)$, $s = 1, \ldots, m$.

Step 2. Select at random a hash function $h_{j_i} \in_R H_n$ such that

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

Therefore, in this way all users who were involved in the mapping can compute the password of the user U_j directly.

Step 3. Append the tuple $< U_{l_s}, O_i, U_j, \ldots, \text{“}yes/no\text{”}, h_{j,i}, n_{l_s,j,i} >$ to the authorization table for all $s = 1, \ldots, m$..

3.4 Example

The following example illustrates our proposed solution. Assume that the users U_1, U_2, U_3 (with login password PS_1, PS_2, PS_3, respectively) have created seven classes $O_1, O_2, \ldots, O_7$ by calling the following sequence of commands:

create class O_1 owner U_1;
create class O_2 owner U_2;
create class O_4 owner U_1;
create class O_3 owner U_2;
create class O_5 owner U_3;
create class O_6 owner U_1;
create class O_7 owner U_3;

Figure 5 shows the class structure and Figure 6 illustrates the authorization table.

The password keys for the classes O_1, O_4, and O_6 owned by U_1 are calculated as:
$n_{1,1} = f_{PS_1}(U_1 \oplus O_1)$,
$n_{1,4} = f_{PS_1}(U_1 \oplus O_4)$, and
$n_{1,6} = f_{PS_1}(U_1 \oplus O_6)$

The password keys for the classes O_2 and O_3 owned by U_2 are calculated as:
$n_{2,2} = f_{PS_2}(U_2 \oplus O_2)$, and
$n_{2,3} = f_{PS_2}(U_2 \oplus O_3)$

Similarly for the classes O_5 and O_7 owned by U_3, the passwords are computed as:
$n_{3,5} = f_{PS_3}(U_3 \oplus O_5)$, and
$n_{3,7} = f_{PS_3}(U_3 \oplus O_7)$

Assume that the access keys $K_1, K_2, \ldots, K_7$ are used to control the access to the corresponding classes $O_1, O_2, \ldots, O_7$. They are chosen at random. Also the collection of hash function $h_1, h_2, \ldots, h_7$ are chosen in such a way that:

$h_1(f_{n_{1,1}}(O_1)) = K_1$,
$h_2(f_{n_{2,2}}(O_2)) = h_2(f_{K_1}(O_2)) = K_2$,
$h_3(f_{n_{2,3}}(O_3)) = h_3(f_{K_1}(O_3)) = K_3$,
$h_4(f_{n_{1,4}}(O_4)) = h_4(f_{K_1}(O_4)) = K_4$,
$h_5(f_{n_{3,5}}(O_5)) = h_5(f_{K_1}(O_5)) = h_5(f_{K_2}(O_5)) = K_5$,
$h_6(f_{n_{1,6}}(O_6)) = h_6(f_{K_1}(O_6)) = h_6(f_{K_3}(O_6)) = K_6$, and
$h_7(f_{n_{3,7}}(O_7)) = h_7(f_{K_1}(O_7)) = h_7(f_{K_3}(O_7)) = h_7(f_{K_4}(O_7)) = K_7$.

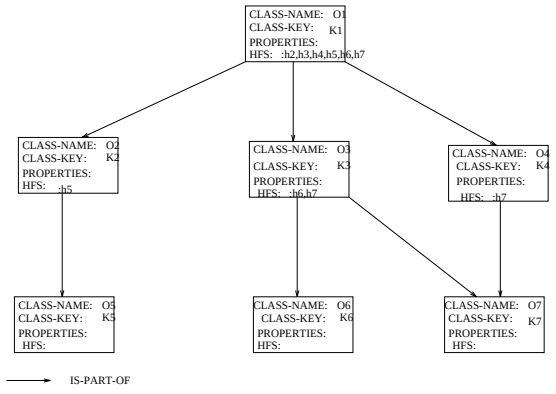

Figure 5: Object structure.

Suppose that the login password of users $U_8, U_9, \ldots, U_{14}$ are $PS_8, PS_9, \ldots, PS_{14}$, respectively. Let us assume that:

U_1 grants U_8 access to O_1;
U_1 grants U_9 access to O_4 and O_6 with grant-option right;
U_1 grants U_{10} access to O_4 with grant-option right;
U_3 grants U_{11} access to O_5 with grant-option right;

GRANTEE	CLASS-NAME	GRANTOR	. . .	GRANT-OPTION	H-FUNCTION	G-PSWRD
U1	O1	U1		"YES"	h_1	$n_{1,1}$
U2	O2	U2		"YES"	h_2	$n_{2,2}$
U1	O4	U1		"YES"	h_3	$n_{1,4}$
U2	O3	U2		"YES"	h_4	$n_{2,3}$
U3	O5	U3		"YES"	h_5	$n_{3,5}$
U1	O6	U1		"YES"	h_6	$n_{1,6}$
U3	O7	U3		"YES"	h_7	$n_{3,7}$

Figure 6: Authorization table.

U_3 grants U_8 access to O_7;
U_9 grants U_{12} access to O_4 with grant-option right;
U_9 grants U_{13} access to O_4;
U_9 grants U_{14} access to O_6 with grant-option right;
U_{11} grants U_{12} access to O_5 with grant-option right;
U_{11} grants U_1 access to O_5 with grant-option right;
U_{10} grants U_{13} access to O_4;
U_{10} grants U_{14} access to O_4 with grant-option right;

The authorization table for users $U_1, U_2, \ldots, U_{14}$ is shown in Figure 7, and the resulting group/class access organization in Figure 8.

The passwords for the users $U_8, U_9, \ldots, U_{14}$ and the classes which they can access are as follows:

$n_{8,1,1} = f_{PS_8}(U_1 \oplus O_1)$,
$n_{9,1,4} = f_{PS_9}(U_1 \oplus O_4)$,
$n_{10,1,4} = f_{PS_{10}}(U_1 \oplus O_4)$,
$n_{12,9,4} = f_{PS_{12}}(U_9 \oplus O_4)$,
$n_{13,9,4} = f_{PS_{13}}(U_9 \oplus O_4)$,
$n_{13,10,4} = f_{PS_{13}}(U_{10} \oplus O_4)$,
$n_{14,10,4} = f_{PS_{14}}(U_{10} \oplus O_4)$,
$n_{9,1,6} = f_{PS_9}(U_1 \oplus O_6)$,
$n_{14,9,6} = f_{PS_{14}}(U_9 \oplus O_6)$,
$n_{11,3,5} = f_{PS_{11}}(U_3 \oplus O_5)$,
$n_{12,11,5} = f_{PS_{12}}(U_{11} \oplus O_5)$,
$n_{1,11,5} = f_{PS_1}(U_{11} \oplus O_5)$, and
$n_{8,3,7} = f_{PS_8}(U_3 \oplus O_7)$

where $n_{j,k,i}$ is the password of the user j to the class O_i granted by the user k. Hash functions are chosen in such a way that:

$h_{1,1}(f_{n_{8,1,1}}(U_1)) = n_{1,1}$,
$h_{1,4}(f_{n_{9,1,4}}(U_1)) = h_{1,4}(f_{n_{10,1,4}}(U_1)) = n_{1,4}$,
$h_{9,4}(f_{n_{12,9,4}}(U_9)) = h_{9,4}(f_{n_{13,9,4}}(U_9)) = n_{9,4}$,
$h_{10,4}(f_{n_{13,10,4}}(U_{10})) = h_{10,4}(f_{n_{14,10,4}}(U_{10})) = n_{10,4}$,
$h_{1,6}(f_{n_{9,1,6}}(U_1)) = n_{1,6}$,
$h_{9,6}(f_{n_{14,9,6}}(U_9)) = n_{9,6}$,
$h_{3,5}(f_{n_{11,3,5}}(U_3)) = n_{3,5}$,
$h_{11,5}(f_{n_{12,11,5}}(U_{11})) = h_{11,5}(f_{n_{1,11,5}}(U_{11})) = n_{11,5}$, and
$h_{3,7}(f_{n_{8,3,7}}(U_3)) = n_{3,7}$.

4 Validation of Access Requests

The processing of a user query starts by checking if the user has appropriate access rights to the classes specified in the query. This is done by the DAC system.

GRANTEE	CLASS-NAME	GRANTOR	. . .	GRANT-OPTION	H-FUNCTION	G-PSWRD
U1	O1	U1		"YES"	h_1	$n_{1,1}$
U2	O2	U2		"YES"	h_2	$n_{2,2}$
U2	O3	U2		"YES"	h_3	$n_{2,3}$
U1	O4	U1		"YES"	h_4	$n_{1,4}$
U3	O5	U3		"YES"	h_5	$n_{3,5}$
U1	O6	U1		"YES"	h_6	$n_{1,6}$
U3	O7	U3		"YES"	h_7	$n_{3,7}$
U8	O1	U1		"NO"	$h_{1,1}$	$n_{8,1,1}$
U9	O4	U1		"YES"	$h_{1,4}$	$n_{9,1,4}$
U10	O4	U1		"YES"	$h_{1,4}$	$n_{10,1,4}$
U9	O6	U1		"YES"	$h_{1,6}$	$n_{9,1,6}$
U11	O5	U3		"YES"	$h_{3,5}$	$n_{11,3,5}$
U8	O7	U3		"YES"	$h_{3,7}$	$n_{8,3,7}$
U12	O4	U9		"YES"	$h_{9,4}$	$n_{12,9,4}$
U13	O4	U9		"NO"	$h_{9,4}$	$n_{13,9,4}$
U14	O6	U9		"YES"	$h_{9,6}$	$n_{14,9,6}$
U12	O5	U11		"YES"	$h_{11,5}$	$n_{12,11,5}$
U1	O5	U11		"YES"	$h_{11,5}$	$n_{1,11,5}$
U13	O4	U10		"NO"	$h_{10,4}$	$n_{13,10,4}$
U14	O4	U10		"YES"	$h_{10,4}$	$n_{14,10,4}$

Figure 7: Authorization table.

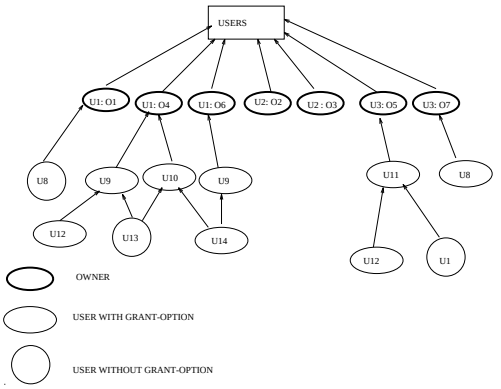

Figure 8: Group/class access organization.

In this section, we show how the validation of access requests are carried out in our DAC system. We assume also that any user query can take on one of the two possible form.

There are two possible kinds of query which may be issued by a user U_l:

- *retrieve O_j.all [from O_i];* i.e. retrieve the value of all properties of the class O_j and its components (or descendants).
- *retrieve O_j.* [from O_i];* i.e. retrieve only the value of properties of the class O_j.

  Note that if the "from clause" is indicated, it means that the class O_j is accessible via class O_i.

Later, we evaluate the security of the proposed DAC system.

4.1 Access Validation

The access validation proceeds according to the following steps:

Step 1. Find an appropriate entry from the authorization table (the entry corresponds to GRANTEE = U_l and CLASS-NAME = O_j).

If there is no such entry, check if the class O_j can be accessed indirectly via other class; check "from clause" exists. If such a class exists and is O_i then search the authorization table for an entry (U_l, O_i). In general, a sequence of classes can exist, in such case, it is necessary to check the suitable sequence of entries in the authorization table.

If the search is not successful then reject the request and exit.

Suppose that the search has been successful and the extracted entry is

$$< U_l, O_k, U_{l'}, \ldots, \text{“}yes\text{”}/\text{“}no\text{”}, h_{l',k}, n_{l,l',k} >$$

Step 2. Calculate $n'_{l,l',k} = f_{PS_l}(U_{l'} \oplus O_k)$ and check $n_{l,l',k} = n'_{l,l',k}$. If $n_{l,l',k} \neq n'_{l,l',k}$ then reject the request and exit. Otherwise while (GRANTEE $\neq$ GRANTOR) go to Step 3.

Step 3. Derive the password for the grantor $U_{l'}$ of the class O_k by obtaining $n_{l',k} = h_{l',k}(f_{n_{l,l',k}}(U_{l'}))$ and finding the corresponding entry in the authorization table.

Step 4. Derive access key for the class O_k from the owner's password as $K'_k = h_k(f_{n_{l',k}}(O_k))$ and check $K'_k = K_k$. If $K'_k \neq K_k$ then reject the request and exit. Otherwise go to the next step.

Step 5. If the " from clause" exists then derive the access key of the class O_j by obtaining $K'_j = h_j(f_{K_i}(O_j))$ and check $K'_j = K_j$.

Step 6. If the check is not successful then reject the request and exit. Otherwise fetch the object O_k and extract the value of the properties requested.

If the "all clause" exists then for all descendants of the object class O_k, say O_s, do Step 7. Otherwise finish.

Step 7. Derive the access key of the class O_s by obtaining $K'_s = h_s(f_{K_k}(O_s))$. Check if $K'_s = K_s$ then fetch the object O_s and extract the value of the properties requested.

4.2 Example

Let user U_{13} issue the following query:

retrieve O_4*.all;*

then the evaluation of access according to the presented algorithm is as follows:

(i) First, look for an entry in the authorization table which corresponds to (U_{13}, O_4). Such an entry exists and is:

$$< U_{13}, O_4, U_9, \ldots, \text{“}NO\text{”}, h_{9,4}, n_{13,9,4} >$$

(ii) Next, the password $n'_{13,9,4} = f_{PS_{13}}(U_9 \oplus O_4)$ is computed and if $n_{13,9,4} = n'_{13,9,4}$ satisfies, the following steps will be executed.

(iii) Calculate the password $n'_{9,4} = h_{9,4}(f_{n_{13,9,4}}(U_9))$. Compare the result to the value of $n_{9,4}$ taken from the entity (U_9, O_4) in the authorization table. The entry is

$$< U_9, O_4, U_1, \ldots, \text{“}YES\text{”}, h_{1,4}, n_{9,1,4} >$$

The next password is computed $n'_{1,4} = h_{1,4}(f_{n_{9,1,4}}(U_1))$. The result is compared to the value $n_{1,4}$ in the entry (U_1, O_4) of the authorization table. The entry is:

$$< U_1, O_4, U_1, \ldots, \text{“}YES\text{”}, h_4, n_{1,4} >$$

Note that GRANTEE = GRANTOR so the following steps will be executed.

(iv) The access key K_4 is computed as $K_4 = h_4(f_{n_{1,4}}(O_4))$. If the access key matches the proper access key of the class O_4, then the access is granted and the value of its properties are fetched.

(v) Since the value of the associated properties of all classes that have relation with O_4 is requested, then the access key of the descendant O_7 is computed as $K_7 = h_7(f_{K_4}(O_7))$. The object class O_7 and the value of its properties are fetched.

5 Class Restructuring

Sometime it is essential that the structure of a complex object class be changed. Some typical changes include addition and deletion of classes, the introduction and removal of links (or relationships) between classes, and replacement or transfer of the ownership of a class. In some cases, this requires that the information of the classes be updated.

If we assume that the access keys of the classes do not change and are fixed, then these alterations will affect the number of ancestors of classes only. This requires that the SIFFs in the associated ancestors have to be reproduced. Next, we consider the effect of these changes.

5.1 Deletion of Classes

Figure 9 shows a number of possibilities for the deletion of the classes (deleted elements are marked by dotted lines). A class can be deleted from leaves, or from intermediate nodes, or a relationship can be removed.

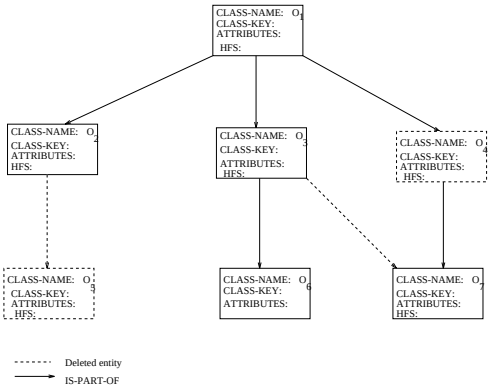

Figure 9: Illustration of possible deletions of relationship.

Deletion of a class from leaves. First, the hash function of the deleted class must be removed from all its ancestors. Second, the authorization table must be updated, i.e. the associated entities which correspond to the deleted class, have to be removed from the authorization table.

Deletion of an intermediate class. In this case, the descendants of the deleted class becomes the descendent of the parent of the deleted class. For example, if the class O_4 is deleted, then the class O_7 becomes a child of O_1. Thus, because the ancestors of all classes below is changed, new SIFF which satisfies Equation (1) must be reproduced and be replaced for the old ones. The authorization table must be updated too.

Deletion of a relationship. The deletion of a relationship may affect the complex object class in two ways.

First, the deletion causes a part of the component class to be deleted. In this case, it is sufficient that the hash function of the deleted part be removed from the associated ancestors and if the deleted part does not exist in the database schema any more, the authorization table must be updated. Otherwise, the authorization table is left intact.

Second, the deletion causes the changes in the ancestors of the descendants below. For example, the ancestor of the class O_7 are classes O_4 and O_1. Then, new SIFFs for all the descendants below must be reproduced and the old ones in the ancestors be replaced with.

5.2 Addition of Classes

Figure 10 illustrates various possibilities of addition to a class structure. It includes adding classes to leaves, or at intermediate node, and adding new relationships. The new classes are shown in Figure 10 as shadowed rectangles and the new relationship as the thick line.

Addition of Classes to leaves. The process described in Subsection 3.2 must be completed.

Addition of Classes at intermediate node. First, the process described in Subsection 3.2 must be completed for new classes and then for all subsequent classes whose ancestors are changed, new SIFFs must be reproduced (see Step 3 of Subsection 3.2) and they replace the old ones in the associated ancestors.

Addition of a New Relationship Just, for all descendants below, new SIFF as described in Step 3 of Subsection 3.2 must be chosen and all old ones in the associated ancestors are replaced.

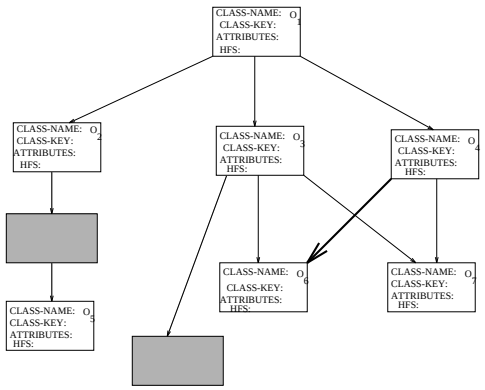

Figure 10: Illustration of possible addition of relationship.

5.3 Replacing and/or Transferring the Ownership

There are two possibilities for replacing the ownership: the owner of the class is replaced by a new one or the login password of the owner has been changed. For the two cases, first, the process described in Subsection 3.2 is completed and then a new SIFF for all users directly granted by him/her must be produced and they replace the old ones in the associated entries of the authorization table.

In our DAC system, it is also possible that the ownership is granted to other users. In other words, it is possible that a class has several owners and they act independently. Supposed that the previous owner of the class O_i is U_j and he/she wants to grant the ownership of the class O_i to user U_r and U_s. It requires the following steps to be completed.

Step 1. For all new owners, password according to Step1 in Subsection 3.2 are chosen: $n_{r,i} = f_{PS_r}(U_r \oplus O_i)$ and $n_{s,i} = f_{PS_s}(U_s \oplus O_i)$

Step 2. A new SIFF is selected such that

$$h_i'(f_{n_{r,i}}(O_i)) = h_i'(f_{n_{s,i}}(O_i)) = h_i'(f_{K_{l_1}}(O_i)) = h_i'(f_{K_{l_2}}(O_i)) = \ldots = h_i'(f_{K_{l_p}}(O_i)) = K_i$$

Step 3. The associated entry of the user U_j in the authorization table is updated and the new entries for users U_r and U_s are appended.

6 Grouping and Group Updating

When a user has grant authorization, he/she can create a user group and becomes the sponsor of the group. He/she can give his/her rights, partially or totally, to the members of the group by running grant commands (see Subsection 3.3). Again a member of the group with grant option can propagate privileges to other users as well, and so on. As a result, we can have a user group hierarchy as shown in Figure 2. One of the important problems in group hierarchy is group updating. In the following, we consider cases of this problem and then suggest ways of handling them.

There are three possible cases that the updating is required in group organization. They are:

(i) a member of a group, who is the sponsor of the group, is deleted;

(ii) a new user and/or a group is added, and/or

(iii) a member of the group, or the sponsor of the group, is replaced by another one.

The effect of each case on grouping organization and the necessary updates is considered next.

6.1 Deletion of Memberships

Deletion of memberships, in a group structure, is done by revoking the right by the sponsor of the group that he/she is a member of. As shown in Figure 11, there are two possibilities to delete a member of a group (deleted elements are represented by dotted lines).

First, the deleted user is just a member of the group. In this case, it is sufficient that the associated tuple is removed from the authorization table, and a new random hashing h', with one degree less collision accessibility property is selected. This hash function can derive the password of the rooted group, say $n_{j,i}$, i.e.

$$h'_{j,i}(f_{n_{l_1,j,i}}(U_j)) = h'_{j,i}(f_{n_{l_2,j,i}}(U_j)) = \ldots = h'_{j,i}(f_{n_{l_{m-1},j,i}}(U_j)) = n_{j,i},$$

and replaces the old one in the associated tuples of the authorization table.

Second, the deleted user is the sponsor of a group. In such case, not only the associated tuple of the user must be deleted from the authorization table but also all the subsequent members and groups, granted by that user must be removed from the authorization table. It is interesting to 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 class by these users will be denied immediately after the deletion of the sponsor. Then the same process, described before, must be done for the group in which the deleted user is a member of (a new SIFF for the rest members of the group must be regenerated).

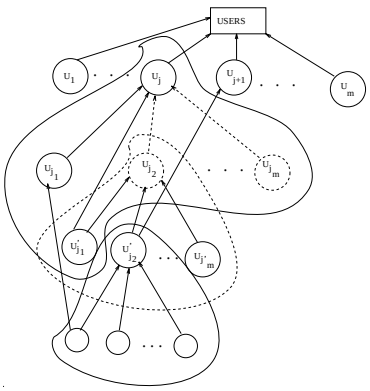

Figure 11: A number of possibilities of membership deletion.

6.2 Addition of New Memberships

Adding a new membership, means that a user with the grant-option gives partial, or total, authorization to a new user. The new user can be an individual user or the sponsor of a group. The shadowed elements in Figure 12 show the possibilities of membership addition to a user group hierarchy. In each cases, it is required that the process described in Subsection 3.3 be completed.

6.3 Replacing

A member in a group can be replaced by a new member. In the case that the member which was replaced was not the sponsor of the group, the process is simply an addition and a deletion as described before. But if the user is the sponsor of a group, not only the deletion and the addition processes have to be completed, for the group which the user is a member of, but also a new SIFF for the group associated with the deleted sponsor must be chosen and the associated entries in the authorization table have to be updated.

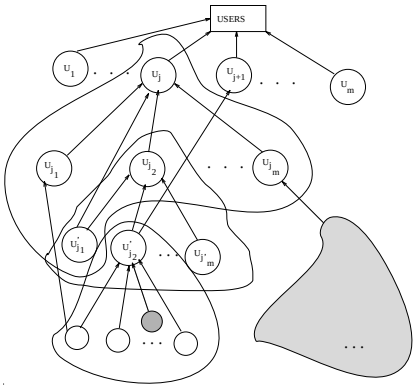

Figure 12: A number of possibilities of membership addition.

7 Extension of the Authorization System

The proposed solution can be extended for all the relationships except inheritance hierarchy ("is-a") where it is required that the following modification, in Step 3 of Subsection 3.2 be made.

Step 3. If the classes $O_{l_1}, O_{l_2}, \ldots, O_{l_p}$ with keys $K_{l_1}, K_{l_2}, \ldots, K_{l_p}$ have relationship with the class O_i , then select a SIFF, h_i, such that

$$h_i(f_{n_{j,i}}(O_i)) = h_i(f_{K_{l_1}}(O_i)) = h_i(f_{K_{l_2}}(O_i)) = \ldots = h_i(f_{K_{l_p}}(O_i)) = K_i$$

In this way all the classes which have relationship with O_i and are involved in the generation of K_i can access O_i directly.

8 Security of the Authorization System

The proposed DAC system is considered secure if for an outsider, or an insider who misuses his/her privileges, it is computationally infeasible to find the access key of the ancestor of a class that he/she is not authorized to access.

Proposition 1 *If the cryptosystem, which is used to encrypt passwords, access class keys and hash functions (see Subsection 3.1), the pseudo-random functions and the SIFF scheme, which are used to construct the DAC system for the object-oriented database system, are secure and the computational power of an outsider is polynomially bounded, then the discretionary access control system is secure.*

Sketch of the Proof:

There are two possibilities of security violation in the authorization system: (i) An outsider may try to access a class to which he/she does not have access rights; (ii) an insider, who has access to some information such as the access key of an class or the password of a user, may try to find out other information such as the access key of the ancestors of the class or may try to find the login passwords of the grantors.

In the first case, this means that the outsider should be able to find valid access keys or valid passwords. To do so, he/she should be able to find a collision for the SIFF or to break the cryptosystem which contradicts our assumptions that the cryptosystem and the SIFF scheme are secure.

In the second case, this means that the outsider, who knows for example the access key of the class O_i or the password, $n_{j,i}$, of the user U_j, may try to find the access keys of the ancestors of O_i and/or the owner's password of the descendants of the object class O_i's owners or may try to find the login password, PS_j, of the user U_j. In either case, it is equivalent to predicting the output of the pseudo-random function or to be able to invert the SIFF. This contradicts our assumption about the security of the SIFF.

9 Conclusion and Future Plans

This paper proposes an approach for providing a cryptographic solution to discretionary access controls in a structurally object-oriented database system. The approach is based on creating class access key by using pseudo-random functions and SIFFs. Similar to SQL, we use a table of authorization identifiers to specify security and then use the given algorithms to evaluate access request and enforce the security policy. The security of the model is based on the difficulty of predicting the output of pseudo-random functions and finding extra collisions for SIFF, both of which known to be difficult.

The proposed authorization model is not a complete security model for an object-oriented environment. A complete model requires a mechanism to enforce:

- the inheritance hierarchy of classes, and
- negative authorization policy.

These will be the subject of our future work.

References

[1] 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.*, pages 237–250. Plenum Press, NY, August 1982.

[2] Hans H. Bruggemann. Rights in an object-oriented environment. In *Database Security V, S. Jajodia,and C.E. Lanwehr (Editors)*, pages 99–115. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1992.

[3] 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, September 1986.

[4] 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. Landweter (Editor)*, pages 105–121. Elsevier Science Publishs B. V. (North-Holland), IFIP, 1989.

[5] O. Deux et al. The O_2 system. *Communications of the ACM*, 34(10):34–48, October 1991.

[6] D. B. Faatz and D. L. Spooner. Discretionary access control in object-oriented engineering database systems. In *Database Security IV, S. Jajodia,and C.E. Lanwehr (Editors)*, pages 73–83. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1991.

[7] 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.E. Lanwehr (Editors)*, pages 85–98. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1991.

[8] T. Hardjono and J. Seberry. A multilevel encryption scheme for database security. In *Proceedings of The 12th Australian Computer Science Conference*, pages 209–218, 1989.

[9] L. Harn and H. Y. Lin. A cryptographic key generation scheme for multilevel data security. *Computer and Security*, 9(6):539–546, 1990.

[10] W. Kim and F. H. Lochovsky (Eds.). *Object-Oriented Concepts, Databases, and Applications.* Addison-Wesley, Reading, Massachusetts, 1988.

[11] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb. The objectstore database system. *Communications of the ACM*, 34(10):50–63, October 1991.

[12] 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 (Editors)*. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1990.

[13] S. J. MacKinnon, P. D. Taylor, H. Meijer, and S. G. Akl. A optimal algorithm for assigning cryptographic keys to access control in a hierarchy. *IEEE Transactions Computer*, C-34(9):797–802, 1985.

[14] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the 21-st ACM Symposium on Theory of Computing*, pages 33–43. ACM, 1989.

[15] Objectivity. *Objectivity Database System Overview.* Objectivity, Inc. Menlo Park, CA, 1990.

[16] K. Ohta, T. Okamoto, and K. Koyama. Membership authentication for hierarchical multigroup using the extended Fiat-Shamir scheme. In *Advances in Cryptology Proceedings of EUROCRYPT'90, I. B. Damgard, Ed.*, pages 446–457. Springer-Verlag, Vol. 473, 1991.

[17] Ontos. *ONTOS Reference Manual.* Ontologic, Inc., Billerica, Massachusetts, 1989.

[18] 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, Venice, LNCS 303*, pages 231–250. Springer, March 1988.

[19] S. M. Thatte. A modular and open object-oriented database system. *SIGMOD RECORD*, 20(1), March 1991.

[20] Versant. *VERSANT Technical Overview.* Versant Object Technologies, Inc., Menlo Park, CA, 1990.

[21] Y. Zheng, T. Hardjono, and J. Pieprzyk. The sibling intractable function family (SIFF): Notation, construction and applications. *IEICE Transactions, Fundamentals*, E76-A(1):4–13, January 1993.

[22] Y. Zheng, T. Hardjono, and J. Seberry. New solutions to access control in a hierarchy. Technical report, Department of Computer Science, The University of Wollongong, 1992.