# A Cryptographic Mechanism for Object-Instance-based Authorization in Object-Oriented Database Systems

Ahmad Baraani-Dastjerdi
Josef Pieprzyk
Reihaneh Safavi-Naini
Janusz R. Getta
Department of Computer Science
University of Wollongong
Wollongong, NSW 2522, AUSTRALIA
e-mail: [ahmadb,josef,rei,jrg]@cs.uow.edu.au

March 20, 1995

## Abstract

In this paper a mechanism for access control at the instance level of a class in 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 object. Each object-instance in the object-oriented model is associated with secure access keys that insure secure access to the object and all related objects. The security of the system depends on the difficulty of predicting the output of pseudo-random functions and on finding extra sibling collision for the sibling intractable function family. The authorization system supports ownership and granting/revoking of access rights.
**Keywords:** Authorization, Access control, DAC, MAC, Data security, Database security, Object-oriented databases, Application of cryptography.

## 1 Introduction

Most of the current models for authorization in database systems are developed for relational databases [10, 21]. Object-oriented databases have a much richer and complex structure for which the relational authorization models are not adequate. Aspects such as inheritance and composite data structures allow expression of flexible and arbitrarily specific discretionary policies. Hence, authorization models for object-oriented databases are more complex than their relational counterparts. Until now only a few authorization models for object-oriented databases are proposed [7, 14, 8, 1].

Inheritance and composite data structure create hierarchical structure in object-oriented databases under *inclusion* and *is part of* relations, respectively [26]. An interesting question is how to extend the known cryptographic solutions for hierarchical access control to access control in object-oriented databases. Two well-known solutions to hierarchical access control problem are based on RSA cryptosystem [4], and one way hash functions [25].

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 hierarchy become extremely large when the number of nodes is large, we use the second solution, proposed by Zheng, Hardjono, and Pieprzyk [25], which is based on the sibling intractable function family (SIFF) to develop a cryptographic solution for discretionary access control in object-oriented databases.

In [7], we proposed a similar approach to provide access control on class-level and to manage user groups in a structurally object-oriented. The independent existence of object-instances in an object-oriented database allows the object-instances to have their own access control (see [1]).

In this paper, we develop a mechanism to control access in object-instance-level authorization. The access at the instance level of a class requires unique identification of instances; the user may refer to an instance by providing its unique identifier in the query. We use pseudo-random functions, SIFFs, and an authorization class, instead of access control lists (or protection matrix), to provide a cryptographic mechanism for object-instance-based authorization. The advantages of our approach compared to the traditional methods of access control (based on the protection matrix and supporting implicit rights) are as follows.

- Supporting a protection matrix and searching at the granularity of object-instances when the number of objects grow fast is extremely hard and time consuming. Finding implicit rights will even be harder. We use pseudo-random functions and SIFFs to produce a pair of unique and secure *access key* and *password* for each object-instance of a class and its owner, respectively. Access keys and passwords may be also derived from related objects during query processing. We employ an authorization class, instead of access control lists (or protection matrix), to modify 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 alterations of the organization of user groups only requires manipulation of the authorization class rather than checking through all access control lists in the database. Moreover, because of data structure consistency, database system operation can be used to manipulate the authorization class. Hence, access request may be checked during query processing.

- 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 collisions for the SIFF, both of which known to be difficult.

- Other 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 easier.

- Enforcing multi-owners for an object is possible.

The organization of the paper is as follows. The next section discusses the basics of object-oriented database model. In Section 3, we present the discretionary policy that our system supports and present the intuitive definition of SIFF. In section 4, we give the notations and assumptions that will be used in this paper. In Section 5, we describe the proposed solution and in Section 6, consider the validation of access requests according to the proposed system. Next we discuss the effect of addition, deletion and/or replacement of objects, or users/groups, on the authorization system (Section 7 and 8). The security of the proposed system is discussed in Section 9. The difficulty of implementation and complexity of the system are discussed in Section 10, and we conclude the paper in the last section by summarizing the results.

## 2 Review of Object-Oriented Concepts

The object-oriented model is based on number of basic concepts [16]. Here, we only describe those concepts that are relevant to the subsequent discussion.

In a general-purpose object-oriented database system, all "real world" entities are modeled as objects. Every object encapsulates *a state* and *a behavior.* The state of an object is implemented by *attributes* (or instance variables), and the behavior of an object is encapsulated in *methods* that operate on the state of the object. The state and behavior are collectively called *properties of object.* Furthermore, an object is associated with a unique identifier called *object identifier* (OID) and may also be given a name.

A collection of the objects that share the same set of properties forms a class. An object belongs to only one class and is an instance of that *class.*

A class may be regarded as a tuple (CLASS-NAME, List-of-Properties) where CLASS-NAME indicates the name of the class and

List-of-Properties is the list of attributes and their domains, and the list of methods. An object may also be regarded as a tuple (OID, OBJECT-NAME, CLASS-NAME, Values) where OID is the object identifier, OBJECT-NAME is the name of the object, CLASS-NAME indicates the name of the class to which the object belongs, and Values is the associated state of the object.

Classes can be organized into hierarchy of classes. There are two types of hierarchies which are orthogonal to each other: *class-composition hierarchy and class-inheritance hierarchy* [24]. 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.

In the class-composition hierarchy, a class may consist of a set of attributes where some or all of the attributes may be classes with their own sets of attributes, and so on. This can be represented as a directed graph of classes rooted at that class. Classes with complex internal structures are sometimes called composite or complex object classes. A database system which provides mechanisms to define and to manipulate objects with complex internal structure is called *a structurally object-oriented database system* (see [2]).

A class-inheritance hierarchy is a hierarchy of classes in which the child node is a specialization of the parent node (and conversely, the parent node is a generalization of the child node [13]). For a parent-child pair of nodes in the class hierarchy the parent is the superclass of the child, and the child is the subclass of the parent. The properties of a class are inherited by all its subclasses. Additional properties also may be specified for each of the subclasses. Some systems allow a class to have only one superclass, whereas others allow a class to have arbitrary number of superclasses. In the former, a class inherits properties from only one class; this is called *the single inheritance.* In the latter, a class inherits properties from more than one superclass; this is called *the multiple inheritance.* In a system which supports the single inheritance, the classes form a hierarchy, called a *class hierarchy.* If a system supports the multiple inheritance, the classes form a rooted directed acyclic graph, sometimes called a *class lattice* [24]. A database system which provides mechanisms to support the definition of class in a class-inheritance hierarchy and to define arbitrary methods is called *behaviorally object-oriented database system* (see [2]).

A third type of object-oriented databases called *fully object-oriented database systems* is defined by Dittrich [2]. Fully object-oriented database systems combine the properties of structurally and behaviorally object-oriented database systems.

A mechanism for providing authorization for object-oriented databases needs to address both the class-composition and the class-inheritance hierarchies. The mechanism presented here is designed to control access to a fully object-oriented database system at the object-instance level.

# 3 Security Considerations

To enforce security of data in a database, we start with an *authorization model.* The model defines *which users have what type of access to which information.*

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

**A discretionary access control policy** specifies the access types of users on data, and the rules whereby users can, at their discretion, grant and revoke their authorizations to other users. Several research efforts have addressed discretionary access controls for OODBSs [14, 8, 1].

**A mandatory access control policy**, specifies the rules whereby users can obtain direct or indirect access to classified data, and the rules for sanitizing and reclassifying data. These policies apply only to *multilevel databases,* which are databases that contain information of different classification. A number of papers [5, 12, 17, 20] address mandatory access control for OODBSs.

**A role based access control policy**, provides mechanisms for the enforcement of least privilege and separation of duties. These policies apply to database applications, where individuals act in certain roles and perform only specific tasks defined for the role. This model is first introduced by Clark and Wilson [9].

In this paper, we only consider discretionary access control policy (DAC). In the following, we first describe the main features of DAC security policies supported by our system, and then introduce the concept of the SIFF.

### 3.1 DAC Security Policies

In the following, we review the main aspects of an authorization system for object-oriented databases and give the choices made for the rest of this paper.

**1- Granularity of Authorization.** In the authorization system for an object-oriented database, the granularity of controls, i.e. the smallest unit of authorization, may be a class, an object-instance, and/or an attribute (instance variables) [1].

In *class-based authorization,* an authorized subject can access a class and all its instances.

In *instance-based authorization,* instances of a class require. In this case, a subject may have authorization for a subset of the instances of a class.

Finally, in *instance-variable-based authorization,* access control on instance-variables or attributes is enforced. Therefore a subject can be limited to access only part of an object.

In this paper, an authorization mechanism for instance-based authorization is developed.

**2- Operations or Access types.** An authorization system allows a number of possible operations on the protected objects. We assume the following set of operations in the authorization system: *read, write, delete, execute,* and *create.* The operations are partially ordered such that access right to operations of higher order implies access right to operations of lower order. The assumed order is: *read* $<$ *execute* $<$ *write* $<$ *delete* $<$ *create.*

**3- Authorization Types.** Rabitti, Woelk, and Kim [1] distinguish two types of authorization in an OODBS: *full and partial authorization.*

A user with *full authorization* right for a set of operations on an object has the same operation rights to the components of the object, i.e, the objects that form the structural part of the object.

In the case of *partial authorization,* access to an object does not extend to the components of the object. Similarly access right to a set of objects does not imply right to access members of the set.

In our authorization system, two types of authorization, *full and partial,* are allowed. In the full authorization mode, *"read"* and *"execute"* rights are implicitly propagated to component objects of the object that a user is authorized to access. For other operations such as *"write"*, *"delete"*, and *"create"*, the user must be explicitly authorized by the owner of the component objects, unless the two objects have the same owner.

Next we describe two policies and the circumstances that the full authorization can be applied.

**4- Visibility from above.** As described before, there are two different types of object hierarchies in OODBSs. In the first type object-composition hierarchy, composite ones, the root object corresponds to a complex object, and other objects in the hierarchy define its internal structure. Hence, an object may have many descendants (or equivalently, an object may have many ancestors). In this case, if a user is authorized to access an object, (s)he must also be authorized to access all information of the descendants of that object. On the other hand, the user must not be able to access the information of the ancestors of that object. In other words, the data must only be *visible from above* and not be visible from below [18]. Thus, if the full authorization right for an operation (*"read"* and *"execute"* only) to a component of the composite object is given, same access rights must apply to all low level objects (descendants) of the component. Whereas if the partial authorization type is indicated, the user can only access the object to which (s)he is explicitly permitted. Note that neither in the case of the full authorization nor partial authorization, users cannot access the ancestors of the objects of a composite object unless they own the object or have authorization rights.

**5- Visibility from below.** The second view considers the objects organized in an inheritance or class/subclass hierarchy which is different from previous one. In this view, the access to a subclass implies access to those attribute values of the superclasses that correspond to the subclass because of inheritance property. This property is called *visibility from below* [18]. Thus, when users have full authorization rights to some operations on an object-instance of a subclass, they have implicitly the same authorization rights to the relevant object-instances of the superclasses. In the case of partial authorization rights for an object-instance of a subclass, a user can only access the object-instance.

However, users that are given access permissions to an object-instance of a class, will not be authorized to access the object-instances of subclasses of that class unless they are authorized explicitly or are the owners of the instances of those subclasses.

**6- Ownership.** Each object-instance has its owner. Every time, a user creates an object, (s)he will be its owner and have the full authority on it (the full authorization rights). He/she is responsible for granting and revoking access rights to other users. The owner's authority is limited to the object which (s)he creates. The owner has only implicit *"read"* and *"execute"* authorization rights to the objects which has relationships with the owner's object. The owner must get permission explicitly for other operations such as *"write"*, *"delete"*, and *"create"*. The ownership can also be granted and revoked by the creator of the object.

**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 grant and revoke authorization policy for 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.

The following security policy rules regarding user groups are applied in our authorization system:

**7- Group policy.** Users may be members of one group or more. Groups again may become members of higher order groups, resulting in a group hierarchy as shown in Figure 1 of Appendix A. A group has a sponsor who administers it. The sponsor can add new members to the group or remove members from the group. Any user who has sponsorship privilege may create a new group and may grant sponsorship privilege to other users.

The database administrator, administration duties include admitting new users to the database system, or revoking/replacing of the owners of the objects.

**8- Open vs Closed system.** Our authorization 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"*.

### 3.2 Sibling Intractable Function Families (SIFFs)

In this section, the intuitive concept of SIFF is presented. For detailed discussion and formal definition, the reader is directed to [25].

Zheng, Hardjono, and Pieprzyk [25] 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 [19]. A universal one-way hash function is a class of hash function with the property that 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. This means that if a hash function $h$ maps the bit string $x_1, x_2, \ldots, x_i$ to the same string, it must computationally infeasible to find another bit string $x'$ such that

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

As shown in [25], the sibling intractable function families (SIFFs) can be constructed from any universal one-way hash function family.

## 4 Notations and Assumptions

### 4.1 Notations

1. Denote by $\aleph$ the set of all positive integers, $n \in \aleph$ the security parameter, $\sum$ the alphabet $\{0, 1\}$, and $x \in_R X$ an element $x$ randomly chosen from the set $X$.

2. $O_i$ and $OID_i$ are the name of the i-th object and the i-th object identifier, respectively. $U_j$ denotes the j-th user's login-name. $AOID_{j,i}$ denotes the authorization-instance identifier related to the user $U_j$ for the object $OID_i$. We use an $n$-bit string to represent $OID_i$, $U_j$, and $AOID_j$.

3. $PS_j$ indicates the login password of the user $U_j$. This password is chosen by $U_j$ after the database administrator admits him/her to be a user of the database system; it should be long enough and be kept secret by the user.

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

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

6. $K_{db}$ denotes the database key which is kept in a secure place and is only accessible to the TM, $\{x\}_{K_{db}}$ is the ciphertext of $x$.

### 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 input to $n$-bit output strings. $k$ is a polynomial in $n$ and has the property that no object has more than $k$ relevant objects or no group could have more than $k$ users.

3. The random $n$-bit strings $K^r, K^w, K^e, K^d$, $K^c$, corresponding to each of the operations *"read"*, *"write"*, *"execute"*, *"delete"*, and *"create"*, respectively are chosen and are only accessible to the TM.

4. An object in our authorization system is specified by (OID, OBJECT-NAME, CLASS-NAME, Values, SECURITY-INFO). OID is the identifier of the object created uniquely for each object by the DBMS. OBJECT-NAME is the name of the object given by the creator. CLASS-NAME indicates the name of the class. *"Values"* is the associated state of the object. SECURITY-INFO consist of OBJECT-KEY-LIST, is a pair of access keys $(K_i^P, K_i^F)$ corresponding to partial and full authorizations, and H-FUNCTION which indicates the hash function that must be used by the related objects to derive the access key of the object, $K_i^F$. The value of SECURITY-INFO is encrypted with the $K_{db}$ by the TM.

5. In order to enforce security requirements, we define an authorization class (AC) which is of the form (GRANTEE, OBJECT-ID, GRANTOR, MEMBER-LIST, DAC-INFO). GRANTEE indicates the user who is authorized to access this object. OBJECT-ID specifies the identifier of the object. GRANTOR specifies the user who has authorized the GRANTEE, to access the object identified by the OBJECT-ID. MEMBER-LIST is the list of users who are the members of the group whose sponsor is the GRANTEE. DAC-INFO is of the form (ACCESS-TYPE-LIST, AUTH-TYPE, SPONSORSHIP, OWNERSHIP, H-FUNCTION, PSWORD). ACCESS-TYPE-LIST indicates the list of authorization rights which the GRANTEE has on the object OBJECT-ID (it could be *"read"*, *"write"*, *"execute"*, *"delete"*, *"create"*, and *"all"*; the word *"all"* is used to indicate all possible operations). AUTH-TYPE (F or P) indicates if the specified operations (indicated by ACCESS-TYPE-LIST) imply in the relevant objects or not (full or partial authorization). SPONSORSHIP (YES or NO) indicates if the GRANTEE is sponsor of a group (or groups) (indicated by the GRANTOR), and is able to propagate his/her access right to his group's members or not. OWNERSHIP (YES or NO) indicates if the GRANTEE has ownership privilege or not. H-FUNCTION specifies hash function which can be used to derive the grantor's password. PSWORD indicates the user's password. The value of the DAC-INFO is encrypted with the $K_{db}$ by the TM.

6. We consider access control at the object-instance level. This is analogous to access control at the tuple level for relational database systems.

## 5 Proposed Solution

Before we proceed to provide some algorithms in order to generate access keys/passwords and SIFFs associated with objects and users, it is important to stress three things. First because here our concern is only to provide a cryptographic mechanism for discretionary access control in object-oriented database, we therefore do not consider the issues such as authentication and secrecy of stored data. To enforce authentication and secrecy, the scheme proposed by Hardjono, Zheng and Seberry [6] for database authentication based on SIFF can be applied.

Second, the user authentication is done by the underlying operating system, and is secure.

Third, the security of our mechanism relies on the tamper-proof module (TM) for performing generation, encryption, and decryption of the security information such as access key and SIFFs; and for checking validity of access. The TM depends on the security of underlying operating system and DBMS. It can be an intermediary between the user and the database system or between the database and physical layer, or it can be a separate function in the database system. Figure 2 in Appendix A shows the case that the TM is a separate function in the DBMS.

The database managemant system (DBMS) provides essential authorization information such as the object identifier and operation in plain form; and user password, access key, and SIFF in cipher form to the TM. Then the TM evaluates the request according to the following algorithms, and pass the result to the DBMS.

In the rest of this section and following section, some algorithms for generating access key for new object, propagating the authorization right, and for checking validity of access request are provided.

### 5.1 Access Key Generation Processing

In order to control access to an object, we produce access keys $K_i^P$ and $K_i^F$ for the object $OID_i$, corresponding to the partial and the full authorization types respectively, such that $K_i^F$ 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 relationship, the instances of subclasses of the object $OID_i$ can derive the access key $K_i^F$. Whereas in the case of the aggregation relationship, the object-instances of ancestors of the object $OID_i$ are able to compute the $K_i^F$.

Every time a user requests access to a specific object, either the access key $K_i^F$ for full authorization rights, or $K_i^P$ for partial authorization rights to this object, is computed and is compared with the stored one by the TM. When they match, the access is permitted. Otherwise, the request is denied.

When a user $U_j$ (with login password $PS_j$) creates an object $OID_i$ by running *create command,* the following two phases to generate access keys for the object will be completed by the DBMS. After the object $OID_i$ is created, the user $U_j$ will be the owner of that and have the full control over it. The owner can give access rights to other users by running the *grant command.*

**Phase 1. Partial authorization.**

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

**Step 2.** Select, at random, the access keys $K_i^P$ and $K_i^F$ for the object $OID_i$ ($K_i^P$ and $K_i^F \in_R \sum^n$) for full and partial authorizations, respectively.

**Step 3.** Select at random hash function $h_i^P \in_R H_n$ corresponding to the partial authorization type such that

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

$K_i^P$ can only be generated by the password of the owner, $n_{j,i}$, using $h_i^P$.

After the above steps are completed by the TM, it also encrypts DAC-INFO, $\{(\text{"}all\text{"}, \text{"}F\text{"}, \text{"}yes\text{"}, \text{"}yes\text{"}, h_i^P, n_{j,i})\}_{K_{db}}$. Then the following step is completed by the DBMS.

**Step 4.** Create the object $(U_j, OID_i, U_j,$ MEMBER-LIST, DAC-INFO) as an instance of the class AC.

**Phase 2. Fully authorization.**

Suppose that objects $OID_{l_1}, OID_{l_2}, \ldots, OID_{l_p}$ with access keys $K^F_{l_1}, K^F_{l_2}, \ldots, K^F_{l_p}$ are related to the object $OID_i$.

**Step 1.** Select at random hash functions $h^F_i \in_R H_n$ corresponding to the full authorization type and
$h^F_i(K^F_{l_1}\|K^r) = h^F_i(K^F_{l_1}\|K^e) = h^F_i(K^F_{l_2}\|K^r) = h^F_i(K^F_{l_2}\|K^e) = \ldots = h^F_i(K^F_{l_p}\|K^r) = h^F_i(K^F_{l_p}\|K^e) = K^F_i$ (2)

So in this way all objects which have relationship with the object $OID_i$ can access it directly by generating the access key $K^F_i$ from their access key $K^F_{l_s}$ $(1 \leq s \leq p)$ for *"read"* and *"execute"* operations only, by using $h^F_i$. Note that in case of inheritance relationship $OID_{l_s}$ $(1 \leq s \leq p)$ are instances of subclasses of the object $OID_i$. Whereas in case of aggregation relationship $OID_{l_s}$ $(1 \leq s \leq p)$ are object-instances of ancestors of the object $OID_i$.

**Step 2.** Append the hash function $\{h^F_i\}$ in the H-FUNCTION of the object $OID_i$.

The Step 1 and Step 2 are completeed by the TM and the DBMS, respectively.

## 5.2 Granting Process

In this section, we propose a mechanism to propagate access privileges to users. The revocation of access rights is considered in Section 8.

Suppose that the grantor $U_j$ has the password $n_{j,i}$ for the object $OID_i$. $U_j$ can be the owner of the object $OID_i$ (or be sponsor of a group) and wants to give $m$ grantees, $U_{l_1}, U_{l_2}, \ldots, U_{l_m}$ with login passwords $PS_{l_1}, PS_{l_2}, \ldots, PS_{l_m}$ respectively, access rights to the object $OID_i$ by running the *grant command*. The following steps have to be completed by the DBMS.

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

**Step 2.** Select at random a hash function $h_{j,i} \in_R H_n$ such that

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

Step 2 ensures that all grantees who are involved in the mapping (the grantees are member of the group whose sponsor is the grantor $U_j$) can directly compute the $AOID_{j,i}$ of the grantor $U_j$ and access the authorization-instance related to the grantor $U_j$. After the above steps are completed by the TM, the TM also encrypts the DAC-INFO, $\{$ ("list-of-operations", "P/F", "yes/no", $h_{j,i}, n_{l_s,j,i})\}_{K_{db}}$. Then, the following steps are completed by the DBMS.

**Step 3.** Create the object $(U_{l_s}, OID_i, U_j,$ MEMBER-LIST, DAC-INFO) as an instances of the class AC for all $s = 1, \ldots, m$.

**Step 4.** Update the MEMBER-LIST of the authorization-instance related to the grantor $U_j$.

# 6 Validation of Access Requests

The processing of a user query starts by checking if the user has appropriate access rights to the objects specified in the query. This is done by the authorization system. In this section, we show how the validation of access requests is carried out in our authorization system.

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

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

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

  The syntax of the query is similar to a simple query. But before using "*" indicates that the hierarchy must be included in the evaluation of the query, i.e., the value of all properties of the object specified in the Target-clause and its relevant objects must be retrieved; in the case of inheritance hierarchy, the relevant objects are object-instances of superclasses which correspond to subclass; whereas in the case of composition (or aggregation) hierarchy, they are object-instances of descendants.

### 6.1 Access Validation

The access validation of a query is done into two phases. First, the legality of the user who issues the query is checked, i.e., it must be checked that the user has proper authorization rights to objects which are requested (**user validation phase**). Second, it must be ensured that the specified access rights is imposed to the object, special in case of hierarchical queries, or outsiders or illegal insiders cannot access the object (**access validation phase**).

In the following discussion, we assume without loss of generality that the user $U_l$ issues the query: **retrieve** $O_j$* **from** $O_i$;

**Phase 1. User validation.**

**Step 1.** Retrieve authorization instance related to the user $U_l$ for the objects $OID_j$ or $OID_i$.

**retrieve** AC **where**
(GRANTEE = $U_l$ **and** OBJECT-ID = $OID_j$ **and** PSWORD = $f_{PS_l}(GRANTOR \oplus OID_j)$)
**or**
(GRANTEE = $U_l$ **and** OBJECT-ID = $OID_i$ **and** PSWORD = $f_{PS_l}(GRANTOR \oplus OID_i)$);

The third part of above check is done by the TM, (PSWORD = $f_{PS_l}(GRANTOR \oplus OID_i)$), or (PSWORD = $f_{PS_l}(GRANTOR \oplus OID_j)$).

If there is no such instance of authorization class AC then the system rejects the request and exits. Suppose that the query has been successful and the extracted instance is

$(U_l, OID_k, U_{l'},$ MEMBER-LIST, $\{$ ("list-of-operation", "F", "yes"/"no", "yes"/"no", $h_{l',k}, n_{l,l',k})\}_{K_{db}})$ where $k$ is $j$ or $i$.

**Step 2.** Derive the password of the owner of the object $OID_k$, say $U_w$, i.e.

**retrieve**
$h_{l',k}(PSWORD)$ **where** GRANTOR $\neq$ GRANTEE **and** OWNERSHIP $\neq$ "yes";

Each time with new values for the hash function, and PSWORD, the TM first obtains $AOID_{l',k}$, the authorization-instance identifier of the GRANTOR, by $AOID_{l',k} = h_{l',k}(PSWORD)$. Then, the query is evaluated by the DBMS until the check (GRANTOR= GRANTEE or OWNERSHIP="yes") is satisfied. Suppose that the derived password and hash functions for the owner $U_w$ of the object $OID_k$ are $n_{w,k}$ and $h_k^P$, respectively ($k$ is $j$ or $i$). Then we next enter the access validation.

**Phase 2. Access validation.**

**Step 1.** Retrieve the object $OID_j$.

**retrieve** $OID_j$ **where**
$(h_j^P(n_{w,j} \| K^r) = K_j^P)$ **or**
$(h_i^P(n_{w,i} \| K^r) = K_i^P$ **and** AUTH-TYPE ="F" **and** $h_j^F(K_i^F \| K^r) = K_j^F)$;

**Step 2.** Retrieve the objects which are in relation with the object $OID_j$ (hierarchical relationship). Let $OIDID_s$ denote such object.

**retrieve** $OID_s$ **where** (AUTH-TYPE = "F" **and** $h_s^F(K_j^F \| K^r) = K_s^F$);

After all $OID_s$ have been retrieved, the process will finish. In the above steps, all checks must be evaluated by the TM.

Note that since the access key of instances of descendants in the case of the composite object, or the instances of superclasses in case of the inheritance hierarchy, can only be derived by the the access key of the object, then it is ensured that the access to the instances of ancestors or subclasses of the object is not occurred. Furthermore, in the case of the partial authorization, because the checks for mapping in Steps 1 and 2 are not satisfied, the request for indirect access will fail.

# 7 Object Restructuring

In an object-oriented database system, objects or relationships might be deleted, added, or modified. In this section, we consider the effect of these changes to our authorization system.

## 7.1 Deletion of Objects

There are two ways of deleting an object from database data model. Firstly, by altering database schema and secondly by deleting an object-instance. In the first case, all objects of database system has to reorganize according to the new schema.

In the second case, we consider three possibilities: deletion from a leaf node, deletion from an intermediate node, and/or deletion of a relationship.

Deletion of an object from a leaf node requires authorization-instances corresponding to the deleted object, be deleted from the AC.

Deletion of an intermediate object which is part of a composite object results in the descendants of the deleted object to become the descendent of the parent of the deleted object. This requires producing a new SIFF that satisfies Equation (2) and replacing it for the old one. If the deleted object is an instance of a subclass of an inheritance hierarchy, because of the specialization relationship between object-instance of subclasses with superclasses, it is logical to delete all object-instances of the lower subclasses of the deleted object. Otherwise the object-instances of the subclass of the deleted object become the object-instances of the superclass of the deleted object and all subclasses of the object-instances of superclasses of the deleted object are changed. New SIFFs which satisfy Equation (2) must be produced for all object-instances 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 be deleted from the AC.

In a fully object-oriented database, there are three types of relationship: (i) "is-part-of" relationship; (ii) "is-a" relationship, and (iii) "normal" relationship such as "teaches", "is-taught-by", "supplies", "is-supplied-by", etc. It must be noted that when a data model is changed, the deletion of the first two types of relationship might occur. The last one might occur when an object does not participate in relationships with other objects.

Deletion of a "is-part-of" relation may affect the composite object in two ways. First, the deletion causes a part of the component object to be deleted. In this case, if the deleted part does not exist in the database schema any more, the AC must also be updated. Otherwise due to the changes in the structure of the deleted part, new SIFFs for all objects of the deleted part must be produced and replace the old ones. The AC is left intact. Second, the deletion causes the changes in the ancestors of the descendants. In this case, new SIFFs for all the object-instances of the descendants must be produced and replace the old ones.

The deletion of a relationship may affect the class/subclass object in two ways. First, deletion causes superclasses of the low-level subclasses to be changed. This requires that new SIFFs for all object-instances of the superclasses be produced and replace the old ones. Second, the deletion causes a subhierarchy of object to be deleted and hence the authorization-instance of the deleted object must be deleted from the AC. Otherwise, because of the changes in the subclasses of superclasses of the sub-hierarchy and the original one, new SIFFs for object-instances of the associated superclasses must be produced and replace the old ones. The AC is left intact.

Finally if a relationship with one or more objects is removed. New SIFFs with one degree less collision accessibility property must be selected and replace the old ones for the object.

### 7.2 Addition of Objects

Object might be added to the database in two forms. First, a new object-instance of an existing class of the database schema is created. Second, an object-instance for a class newly added to the database schema. In the former case, it is sufficient to complete the process described in Subsection 5.1. In the latter case, we distinguish three possibilities: (i) new classes are added at leaves; (ii) new classes are added at intermediate levels, and (iii) new relationships are introduced.

If a class is added as a new leaf, then for all object-instances of the new class, a process similar to the one described in Subsection 5.1 must be completed. Moreover, since the subclasses of the superclasses change, new SIFFs for all object instances of superclasses must be produced and replace the old ones.

If a class is added at intermediate nodes, firstly, the process described in Subsection 5.1 must be completed for object-instances of the new classes and then because of the changes in the hierarchy of classes, new SIFFs must be produced (see Step 3 of Subsection 5.1) and replace the old ones.

In the case of the addition of new relationships, for all object-instances of descendants below or superclasses above new SIFFs, as described in Step 3 of Subsection 5.1, must be chosen and placed instead of the all old ones.

### 7.3 Replacing and/or Transferring the Ownership

There are two possibilities for replacing the ownership: the owner of an object is replaced by a new one or the login password of the owner has been changed. In both cases, the process described in Subsection 5.1 must be completed.

In our authorization system, it is also possible that the ownership be granted to other users. That is, it is possible for an object to have several owners who can act independently. Suppose that the previous owner of the object $OID_i$ is $U_j$ and (s)he wants to grant the ownership of the object $OID_i$ to user $U_r$ and $U_s$. This requires the following steps to be completed.

> **Step 1.** For all new owners, passwords according to Step1 in Subsection 5.1 are chosen: $n_{r,i} = f_{PS_r}(U_r \oplus OID_i)$ and $n_{s,i} = f_{PS_s}(U_s \oplus OID_i)$
>
> **Step 2.** A new SIFF is selected such that
>
> $$h_i^P(n_{j,i}\|K^r) = h_i^P(n_{j,i}\|K^e) = h_i^P(n_{j,i}\|K^w) = h_i^P(n_{j,i}\|K^d) = h_i^P(n_{j,i}\|K^c) = h_i^P(n_{r,i}\|K^r) = h_i^P(n_{r,i}\|K^e) = h_i^P(n_{r,i}\|K^w) = h_i^P(n_{r,i}\|K^d) = h_i^P(n_{r,i}\|K^c) = h_i^P(n_{s,i}\|K^r) = h_i^P(n_{s,i}\|K^e) = h_i^P(n_{s,i}\|K^w) = h_i^P(n_{s,i}\|K^d) = h_i^P(n_{s,i}\|K^c) = K_i^P$$
>
> **Step 3.** Instance of the user $U_j$ in the AC be updated and new instances for users $U_r$ and $U_s$ are created.

# 8 Grouping and Group Updating

When a user has the grant authorization, (s)he can create a user group and become the sponsor of the group. He/she can give his/her rights, partially or totally, to the members of the group by running the *grant commands* (see Subsection 5.2). A member of a group with the grant option can propagate privileges to other users and have a user group hierarchy similar to the one shown in Figure 1 is created. An important issue in the group hierarchy is group updating. In the following, we suggest a way of handling this problem.

We distinguish three possible cases:

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

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

### 8.1 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.

First, the deleted user is just a member of the group. Then, it is required that the associated authorization-instance is removed from the AC, and a new SIFF, with one degree less collision accessibility property is selected. This new SIFF replaces the old one in the authorization-instance associated with usres in the MEMBER-LIST of the sponsor. The MEMBER-LIST must be updated too.

Second, the deleted user is the sponsor of a group. In such case, not only the associated authorization-instance of the user must be deleted from the AC but also all the subsequent members and groups, granted by that user must be removed from the AC. 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).

### 8.2 Addition of New Memberships

Adding a new membership, means that a user, who has sponsorship right, gives partial, or total, authorization to a new user. The new user can be an individual user or the sponsor of a group. In each cases, it is required that the process described in Subsection 5.2 be completed.

### 8.3 Replacing

A member in a group can be replaced by a new member. The member which was replaced can be the sponsor of the group, or can be a normal user. In either cases, it is required that the associated authorization-instance of the replaced member is updayed with the user identifier and password of the new user. Also, a new SIFF for the group which the user is member of must be selected, and replaces the old ones.

## 9 Security of the Authorization System

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

**Proposition 1** *Assume that the tamper-proof module (TM) is secure and is only run by DBMS, 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, which is used for encryption is insecure.*

Sketch of the Proof:

Theproposed authorization system is secure if it is computationally difficult for an intruder to access necessary information required for request validity check in Step 1 of Phase 2 in Section 6.

More precisely, an intruder can access a protected object in the system if (s)he can provide the required security information. We consider the following possible ways.

1. The intruder generates valid access key and associated SIFF, and operation key for object $O_j$, $K_j^P, h_j^P$, and $K^r$ respectively. This means that the ability of intruder is equivalent to predicating outputs of a pseudo-random function, and to finding a collision for the SIFF. This is a contradiction to the definition of pseudo-random function families and SIFF, and to our assumption.

2. The intruder can forge the login password of the owner or one of grantees of object $O_j$, say $U_l$, to obtain $n_{l,j} = f_{PS_l}(U_l \oplus OID_j)$ and consequently to access authorization-instance associated with user $U_l$. This means to break the user authentication system of underlying operating system. This contradicts our assumption.

3. The intruder modifies his (her) access rights or other's, for example, (s)he modifies partial authorization to full authorization, change ownership, etc. If it happens so, then the ability of the intruder is equivalent to breaking the cryptosystem which is used for encryption of DAC-INFO. This is a contradiction to our assumption.

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

# 10 Implementation Consideration

In this section, we briefly consider the difficulty we may encounter when we implement the proposal, and complexity of the system.

## 10.1 Sketch of Implementation

The TM in addition to checking and encryption/decryption, the crucial unit to which must be facilitated, is a k-SIFF hash function and a pseudo-random generator. Figure 3 in Appendix B illustrates one possible way of constructing a k-SIFF hash function generator where OWHF is any one-way hash function, k-UHF is a k-universal hash function.

Because the hardware products of MD4 and MD5 [22, 23] are available in the market, and they are fast enough, they can be good candidates for the OWHF. Of course, it is worth noting that the security of MD4 and MD5 has not been proved formally yet, however they have not been proved that they are insecure either. As state in [3, 25], the possible candidates to create a k-universal hash function families (k-UHF) with collision accessibility property are polynomials over finite fields. For detailed description, the reader is directed to [3, 25].

## 10.2 Complexity of The System

As discussed before, there are two hierarchical organization in an object-oriented data model, *inheritance hierarchy* and *composite hierarchy.* In order that the DBMS evaluates the capabilities along the object hierarchies, the most efficient way is to employ the proposed hierarchical access control for organizing capabilities in a hierarchical manner. 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 [4], was the first, proposed a solution to the *hierarchical access control problem based on the RSA cryptosystem.* There are several problems with this scheme related to its efficiency. Firstly, the scheme is established on a rigid hierarchical organization. Secondly, since the integer values associated with nodes of hierarchy are integers of arbitrary length, so their average size becomes extremely large when the number of nodes is large, and it consequently needs large memory space. Thirdly, the entire system must be predefined by the trusted central authority, and there is no way to expand or modify an existing hierarchy. Later some other solutions to overcome these problems have been proposed [11, 15]. A common drawback with them is still that they are based on a difficulty of breaking the RSA cryptosystem, and make heavy use of the underlying algebraic properties of the crypto-function.

The k-SIFF solution which we have chosen, has several features. First, each node in the hierarchical structure needs to keep only one key of the length $n$ ($n = 128$ bits) which the required storage is low. Second, the SIFF construction is based on the assumption of the existence of one-way function. So SIFF can be based on MD4 (or MD5) instead of very slow RSA system. In the following, we give a sketch comparison of the time complexity and space complexity of the system using SIFF with using RSA approach.

Let $k$ be the number of objects in the hierarchical structure. Let $n$ be the size of the keys. Furthermore, let MD5 and polynomials of degree $k$ over finite fields $GF(2^n)$ ($n = 128$) be used to construct a k-SIFF hash function. The system based on SIFF requires $O(\log k)$ modular multiplications of 128 bits long for key derivation (see Appendix A). Note that the time computation of the pseudo-random function and MD5 is negligible, it is ignored. If the

RSA approach is used for capability derivation and generation, the time complexity of the system will then be $O((k+1)\log k)$ modular multiplications of $n$ bit long integer; $n$ must be at least 512 bits long, four times more. If the computation time of integers associated with objects is added up, then the time complexity will be much higher.

Let $m$ be the entire number of objects in the database. Each object in the proposed system holds a key and a hash function. Hence, the propsed system requires $O(2mn)$ total space. Since $n$ bit string is compressed by the MD5, then it is sufficient that $n$ is chosen near 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 $O(3nm\log k)$ which as mentioned before $n$ must at least 512 bits long.

Finally, in order to increase the efficiency of the system and to get benefit of the order operations; the set of operation is partially ordered such that the lower operation can be inferred from higher operations. The operation keys $(K^r, K^e, \ldots, K^c)$ can be chosen such that lower keys are computable from higher keys, for example, to select keys the Akl-Taylor's method can be used. Therefore, this results in having a shorter list of access rights, and SIFFS with smaller degree of collision accessibility property. For example, the Equation (1) in Subsection 5.1 will simplify as follows

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

## 11 Conclusion and Remarks

This paper proposes a cryptographic mechanism for discretionary access controls at object-instance level in a fully object-oriented database system. The mechanism is based on creating unique and secure access keys for each object, and unique passwords for owners and user groups, using pseudo-random functions and SIFFs, such that they can be derived by the objects which have relationship with or by the user who are a member of the group. We use an authorization class to specify security and then use the 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, both of which are known to be difficult.

Object-instances based authorization system presents a finer granularity than class based authorization system and enables the control to be imposed on individual objects. We note that as the numbers of users and objects in the system grow, the number of instances in the authorization class will increase and enforcing security and manipulating the object structure become resource expensive. To alleviate this problem we may group the objects according to their owners and then use the groups as the units of authorization in the system.

## References


[1] F. Rabitti and E. Bertino and W. Kim and D. Woelk. A Model of Authorization for Next-Generation Database Systems. *ACM Transactions on Database Systems*, 16:88–131, March 1991.

[2] 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.

[3] Mark N. Wegman and J. Lawrance Carter . New Hash Functions and Their Use in Authentication and Set Equality. *Journal of Computer and System Sciences* , 22:265–279, 1981.

[4] 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.

[5] T. F. Keefe and W. T. Tsai. Security Model Consistency in Secure Object-Oriented Systems. In *5th Annual Computer Security Applications Conference Tucson, Arizona*, pages 290–298. IEEE Computer Security Press, December 1989.

[6] T. Hardjono and 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)*, pages 334–342, 1992.

[7] A. Baraani-Dastjerdi and J. R. Getta and 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)*, volume 17(2), pages 36–45. Australian Computer Science Communications, Ron Sacks and Justin Zobel (Editors), January 1995.

[8] 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.

[9] 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*, pages 184–194. Oakland, CA, 1987.

[10] E. B. Fernandez and R. C. Summers and C. Wood. *Database Security and Integrity*. Addison-Wesley Publishing Company, 1981.

[11] G. C. Chick and S. E. Tavares. Flexible Access Control With Master Keys. In *Advances in Cryptology Proceedings of CRYPTO'89, G. Brassard, Ed.*, pages 316–322. Springer-Verlag, 1990.

[12] 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.*, pages 260–272. IEEE, May 1992.

[13] John M. Smith and Diane C. P. Smith. Database Abstraction: Aggregation and Generalization. *ACM Transactions Database Systems* , 2(2):105–133, June 1977.

[14] K. R. Dittrich and M. Hartig and H. Pfefferle. Discretionary Access Control In Structurally Object-Oriented Database Systems. In *Database Security II: Status and Prospects, C. E. Landwehr (Editor)*, pages 105–121. Elsevier Science Publishers B. V. (North-Holland), IFIP, 1989.

[15] L. Harn and 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, May 1990.

[16] M. Atkinson and D. DeWitt and D. Maier and F. Bancilhon and K. Dittrich. The Object-Oriented Database System Manifesto. In *Proceeding of First International Conference on DOOD89, Kyoto, Research Park, Kyoto, Japan*, pages 223–240. Published by Elsevier Science Publishers B. V. North-Holland, 1990, December 1989.

[17] M. B. Thuraisingham. Mandatory Security in object-oriented database Systems. In *Proceedings International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 203–210. New Orleans, October 1989.

[18] M. M. Larrondo-Petrie and E. Guides and 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.

[19] 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.

[20] M. S. Olivier and S. H. Von Solms. A Taxonomy for Secure Object-Oriented Databases. *ACM Transactions on Database Systems*, 19(1):3–46, March 1993.

[21] P. P. Griffiths and B. W. Wade. An Authorization mechanism for a Relational

Database System. *ACM Transactions on Database Systems*, 1(3):242–253, September 1976.

[22] Ronald L. Rivest. The MD4 Message Digest Algorithm. In *Advances in Cryptology, Proceedings of CRYPTO'90*, pages 281–291. Springer-Verlag, 1990.

[23] Ronald L. Rivest. The MD5 Message-Digest Algorithm . MIT Laboratory for Computer Science and RSA Data Security, Inc., Request for Comments (RFC), 1992.

[24] Won Kim. Object-Oriented Databases: Definition and Research Directions. *IEEE Transactions on Knowledge and Data Engineering*, 2(3):327–341, September 1990.

[25] Y. Zheng and 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.

[26] Yair Wand. A Proposal for a Formal Model of Objects. In *Object-Oriented Concepts, Databases, and Applications, W. Kim and F. H. Lochovsky ( Eds.)*, pages 537–559. Addison-Wesley, Reading, Massachusetts, ACM Press, 1989.

# A Appendix

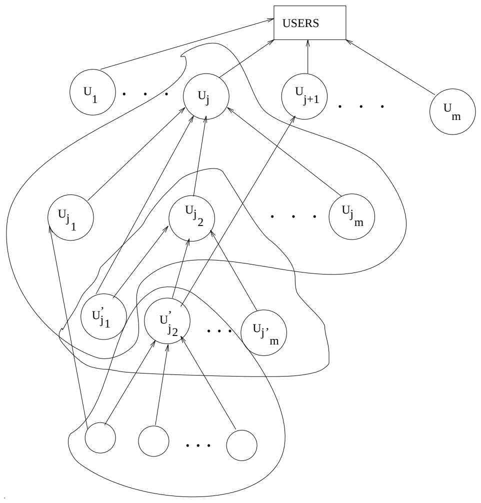


Figure 1: User groups Hierarchy.

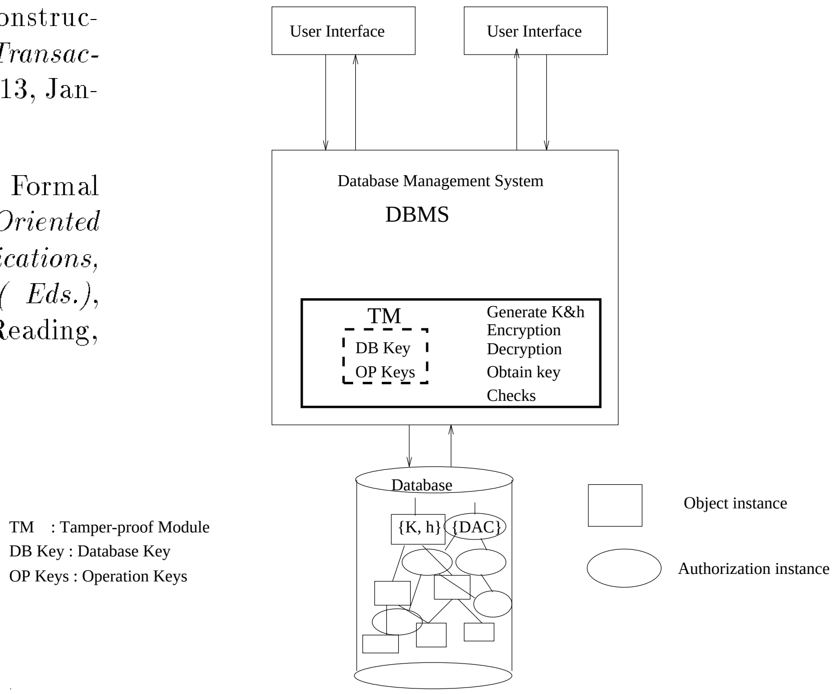


Figure 2: One possible implementation of the TM.

# B Appendix

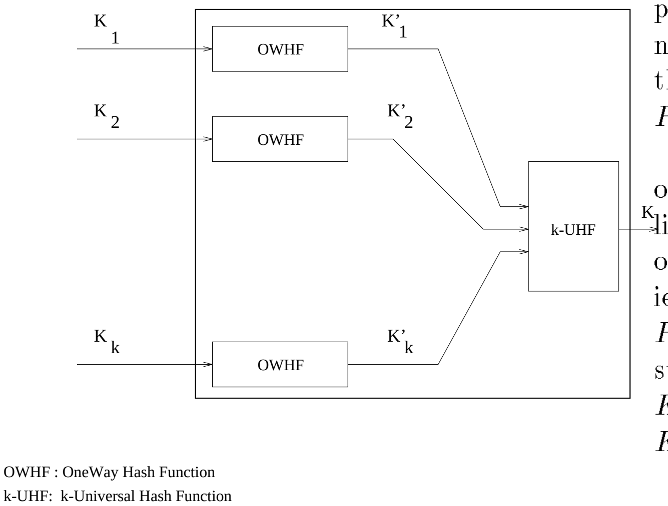


Figure 3: A sketch of Implementation of k-SIFF hash function.

The OWHF can be any one-way hash function such as MD4 or MD5, and the k-UHF is a k-universal hash function with collision accessibility property which can polynomials over finite fields.

A polynomial $P(x)$ of degree $k$ over finite $GF(2^n)$ has $k$ Colliding points.
$P(x) = a_0 + a_1x + \ldots + a_{k-1}x^{k-1} =$
$(x + b_0)(x + b_1)\ldots(x + b_{k-1})$ where $a_0, a_1, \ldots, a_{k-1}, b_0, b_1, \ldots, b_{k-1} \in GF(2^n)$.

When it is used for derivation it cost $k$ modular multiplications, $O(k)$. In order to convert $P(x)$ that the number modular multiplications is reduced, one possible implementation of $P(x)$ is shown in Figure 4.

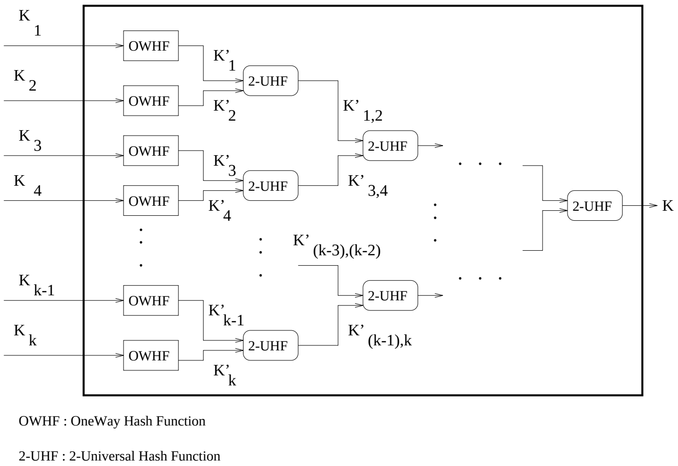


Figure 4: Some improvement of implementation of k-SIFF.

In the first layer, polynomials of degree two are defined or in other words they have two collisions. It is then required to calculate $k/2$ poly-
nomials $P_{1,2}(x), P_{3,4}(x), \ldots, P_{(k-1),(k)}(x)$ such that $P_{1,2}(K'_1) = P_{1,2}(K'_2) = K'_{1,2}$, $P_{3,4}(K'_3) = P_{3,4}(K'_4) = K'_{3,4}, \ldots,$

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, ie., 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 $P_{1,2,3,4}(K'_{1,2}) = P_{1,2,3,4}(K'_{3,4}) = K'_{1,2,3,4}$, $P_{5,6,7,8}(K'_{5,6}) = P_{5,6,7,8}(K'_{7,8}) = K'_{5,6,7,8}, \ldots$

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 multiplication).

There is however one problem to be solved; as there are many different polynomials and only one "path" is used, the sytem 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.