Journal of Universal Computer Science, vol. 3, no. 6 (1997), 730-753
submitted: 28/2/96, accepted: 6/5/97, appeared: 28/6/97 © *Springer Pub. Co.*

Using Cryptographic Hash Functions for Discretionary Access Control in Object-Oriented Databases

Ahmad Baraani-Dastjerdi
(University of Wollongong, Australia
ahmadb@cs.uow.edu.au)

Josef Pieprzyk
(University of Wollongong, Australia
josef@cs.uow.edu.au)

Reihaneh Safavi-Naini
(University of Wollongong, Australia
rei@cs.uow.edu.au)

Janusz R. Getta
(University of Wollongong, Australia
jrg@cs.uow.edu.au)

Abstract: This is a discussion paper which presents a cryptographic solution for discretionary access control in object-oriented databases. Our approach is based on the use of pseudo-random functions and sibling intractable function families (SIFF). Each entity (object or class) in the object-oriented database model is associated with access keys that ensure secure access to that entity and all related entities. The main advantage of our approach is its ability to verify an access request during query processing. Pseudo-random functions and SIFF are applied in such a way that cryptographic keys can be generated from keys of related objects or users. The security of the system depends on the difficulty of predicting the output of pseudo-random functions and on finding extra collision for the sibling intractable function family. The authorization system supports ownership and granting/revoking of privileges.

Key Words: Data security, Database security, Object-oriented databases, Access control, Discretionary security policy, Application of cryptography.

Category: D.4.6 [**Software**]: Security and Protection; H.2.0 [**Database Management**]: General; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection.

1 Introduction

In an object-oriented database system model, aspects such as classes, inheritance, and composite data structures allow expression of rules for computing implicit authorizations from explicit ones. Hence, an access request to database objects may require applying authorization rules to explicit privileges to derive implicit authorizations [2, 8, 18]. One important question is whether implicit authorizations must be evaluated each time an access is requested, or whether they should be stored as redundant authorizations. If implicit authorizations are stored, the protection matrix gets very big. Consequently, the processing of access requests becomes inefficient. In this paper, we propose a solution to this problem, based on cryptographic hash functions.

Inheritance (*inclusion* relation) and composite data structures (*is part of* relation) create hierarchical structures in object-oriented database systems [23]. An interesting question is how conventional cryptographic solutions for hierarchical access control in multi-level systems can be extended to object-oriented database systems. Two such solutions are based on the RSA cryptosystem [1] and one-way hash functions [25].

The main drawback of the first solution is that it is limited to a fixed hierarchy, with no provision for possible changes to the hierarchy. Moreover, the integer values associated with the nodes of the hierarchical structure become extremely large when the number of nodes is large. We use the second solution, proposed by Zheng, Hardjono, and Pieprzyk [25], which is based on the sibling intractable function families (SIFF). We show how to develop a cryptographic solution for discretionary access control (DAC) in object-oriented database systems. The solution applies pseudo-random functions, SIFF, and an authorization class (instead of access control lists or a protection matrix). The desirable properties of our approach are as follows.

1. We employ pseudo-random functions and SIFF to produce a pair of unique and secure *access keys* and *passwords* for each database object (instances or classes) and its owner. Access keys and passwords for implicit authorizations may be derived from related database objects during query processing.
2. We use an authorization class (AC), 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 alteration of the membership of user groups requires manipulation of the AC only rather than checking all access control lists in the database. Moreover, because of data structure consistency, the database system operation can be used to manipulate the AC. Hence, an access request may be verified during query processing.
3. The security of the system relies on (i) the indistinguishability of pseudo-random functions from the truly random one; and (ii) the difficulty of finding collisions for SIFF, both of which are provably hard[1].
4. Operations such as grant, revoke, propagation of rights, and the required modifications due to the changes of both the user groups and the class structure are relatively easy to perform.
5. The existence of multiple owners of a database object (instances and classes) is possible.

2 Sibling Intractable Function Families (SIFF)

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

2.1 SIFF Definition

Zheng, Hardjono, and Pieprzyk [25] introduced the notion of sibling intractable function family (SIFF), which is a generalization of the concept of universal

[1] By the result of [13], the existence of one-way functions is sufficient for the construction of pseudo-random function families

one-way hash function defined by [17]. A universal one-way hash function family is a class of hash functions with the property that the number of functions that map any collection of r distinct input strings to the same hash value is fixed. SIFF is the universal one-way hash function family with the additional property that given a set of colliding sequences, it is computationally infeasible to find another sequence that collides with the initial set. This means that if a SIFF function h maps the bit strings $x_1, x_2, \ldots, x_i$ to the same hash value, then there is no polynomial time algorithm that can be used to compute some x' $(x' \neq x_j; j = 1, \ldots, i)$ such that

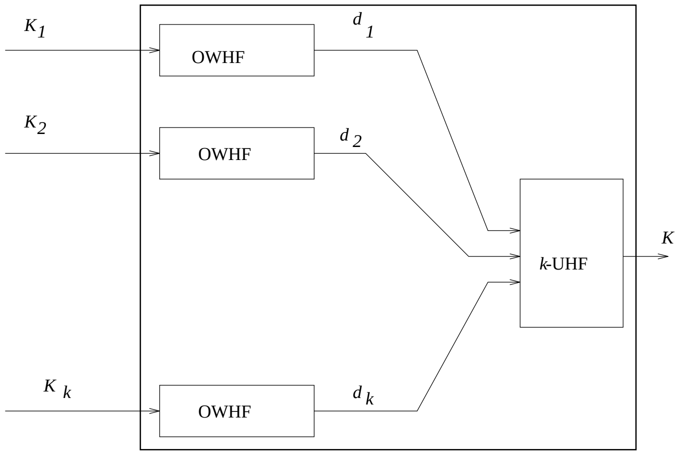
$$h(x') = h(x_1) = \cdots = h(x_i).$$

Let $H = \{H_n \mid n \in \mathcal{N}\}$ be an infinite family of functions, where $H_n = \{h \mid h : \Sigma^{l(n)} \to \Sigma^{m(n)}\}$. Note that a function $h \in H$ maps $l(n)$-bit inputs to $m(n)$-bit outputs. H is *polynomial time computable* if there is a polynomial time algorithm (in n) which computes outputs for given inputs for all $h \in H$. H is *samplable* if there is a probabilistic polynomial time algorithm that on input $n \in \mathcal{N}$ outputs randomly with uniform probability distribution a description of $h \in H_n$. Moreover, H has the *k-collision accessibility property*, or simply the collision accessibility property, if for all n and for all $1 \leq i \leq k$, given a set $X = \{x_1, x_2, \ldots, x_i\}$ of i initial strings in $\Sigma^{l(n)}$, it is possible in probabilistic polynomial time to select randomly and uniformly functions from H_n^X, where $H_n^X \subset H_n$ is the set all functions in H_n that maps $x_1, x_2, \ldots$, and x_i to the same strings in $\Sigma^{m(n)}$. Let $x \in_R X$ denote an element x which is randomly chosen from the set X with uniform probability. A sibling finder F is a probabilistic polynomial time algorithm that for a given input $X = \{x_1, x_2, \ldots, x_i\}$ and the description of h tries to compute a new collision. The finder F outputs either "?" ("I cannot find") or a string $x' \in \Sigma^{l(n)}$ such that $x' \notin X$ and $h(x') = h(x_1) = \cdots = h(x_i)$. The following definition of SIFF is taken from [25]. For more details, the reader is referred to [11] and [25].

Definition 1. [25]. Let $k(n)$ be a polynomial with $k(n) \geq 1$ and $H = \{H_n \mid n \in N\}$ be a family of functions that are computable in polynomial time and samplable. Moreover they have the collision accessibility property and map $l(n)$-bit input into $m(n)$-bit output strings $H_n = \{h|h : \Sigma^{l(n)} \to \Sigma^{m(n)}\}$. Assume that $X = \{x_1, x_2, \ldots, x_i\}$ is a set of i initial strings, where $1 \leq i \leq k(n)$. H is a k-sibling intractable function family (or k-SIFF) if for all $1 \leq i \leq k(n)$, any sibling finder F, any polynomial $Q(n) > 0$, and for all sufficiently large n,

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

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



OWHF: One Way Hash Function

k-UHF: k-Universal Hash Function

Figure 1: A sketch of a construction of k-SIFF hash function.

2.2 Sketch of Construction of SIFF

As mentioned in [25], SIFF can be constructed from any universal one-way hash function family (*OWHF*). Figure 1 illustrates an example construction of a k-SIFF hash function (see Definition 1).

First each string K_i $(i = 1, \ldots, k)$ is hashed by a one-way function. Then the output (digest) is mapped to the value K by a k-universal hash function,

$$k\text{-UHF}(\text{OWHF}(K_1)) = k\text{-UHF}(\text{OWHF}(K_2)) = \cdots$$
$$= k\text{-UHF}(\text{OWHF}(K_k)) = K.$$

The OWHF can be any one-way hash function such as MD4, MD5 [20] [21], or HAVAL [26] for which a fast hardware implementation is available. Note that MD4 is already considered to be insecure [5]. Also the recent work by Dobbertin [6] casts some doubts about the security of MD5. HAVAL seems a preferred choice for a fast and secure hashing algorithm. As stated in [24, 25], a possible candidate for a k-universal hash function family (*k-UHF*) with the collision accessibility property can be obtained from polynomials over finite fields. Let P_n be the collection of all polynomials over $GF(2^{l(n)})$ with degree less than k, that is,

$$P_n = \{a_0 + a_1x + \cdots + a_{k-1}x^{k-1} \mid a_0, a_1, \ldots, a_{k-1} \in GF(2^{l(n)})\}.$$

For each $p \in P_n$, let u_p be the function obtained from p by chopping the first $(l(n) - m(n))$ bits of the output of p whenever $l(n) \geq m(n)$, or by appending a fixed $(m(n) - l(n))$ bits to the output of p whenever $l(n) < m(n)$. Let $\text{UHF}_n = \{u_p \mid p \in P_n\}$, and UHF$= \bigcup_n \text{UHF}_n$. Then UHF is a k-universal hash function family, which maps $l(n)$-bit input into $m(n)$-bit output strings with the collision accessibility property.

3 Object-Oriented Model Concepts

Here, we only describe those concepts that are relevant to our discussion.

In a general-purpose object-oriented database system, all *real world* entities are modelled as objects. Every object encapsulates *a state* and *a behavior.* The state of an object is implemented by *properties* (or instance variables), and the behavior of an object is encapsulated in *methods* that operate on the state of the object. The state and behavior are collectively called *facets of object* [16]. 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 facets forms a class. An object belongs to only one class and is an instance of that *class.* Each class is given a unique name.

Classes can be organized into hierarchies of classes. There are two different types of hierarchies: the *class-composition hierarchy* and *class-inheritance hierarchy* [14]. The class-composition hierarchy captures the *is-part-of* relationship, whereas a class-inheritance hierarchy represents the *is-a* relationship.

A mechanism for providing authorization for object-oriented database systems needs to address both the class-composition and the class-inheritance hierarchies.

4 Security Policy

The specification of access control may involve a range of policy choices. The choice of security policies is important because it influences the flexibility, usability, and performance of the system [7]. In this paper, our considerations are restricted to discretionary access control (DAC).

4.1 General Policies

In the authorization system for object-oriented database systems, the granularity of the control, i.e., the smallest unit of authorization, may be a class, an object-instance, and/or a property (or instance variable) [18]. We choose the units of authorization to be classes and instances of classes. This means that one user may be granted access to a complete class, while another user may be granted access to its instance. Properties and methods are excluded from our consideration to simplify the model. We will use the term *entity* to refer to either a class or an instance of a class.

An authorization system works with a specific collection of access privileges. We assume the following set of privileges in the authorization system: *read-definition, read, write, delete, execute*, and *create. Read* and *read-definition* privileges are used to read the instances of a class and read the definition of a class, respectively. An *execute* privilege is used to perform the methods associated with a class. In other words, the *execute* privilege can be considered as an *invoker* that can call methods associated with a class. *Write* and *create* privileges are used to modify and to create an instance of a class, respectively. A *delete* privilege is used to delete an instance of a class. Note that in order to exercise some privileges, a user must have other privilege as well. For instance, if a user wants to delete an object the user must first access it and later remove it. This implies that the user must have *read* and *read-definition* privileges as well. Hence to simplify the authorization system, we assume that the privileges are partially ordered such that authorization to access privileges of higher order implies authorization to access privileges of lower order. The assumed order is:

write > *execute* > *read* > *read-definition*,
create > *execute* > *read* > *read-definition*, and
delete > *read* > *read-definition*.

This means that the holder of an access privilege of a higher order possesses privileges of the lower order. For instance *execute* implies that its holder (a user) has both *read* and *read-definition* privileges because the user must be able to read the values and definitions associated with the parameters of a method in order to execute the method. The user can access the result as well. The state of the object will not change. In order to change the state of an object, the *write* privilege is required.

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

4.2 Administrative policies

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

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

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

A group is defined as a set of users or a collection of smaller groups. Groups are not necessarily disjoint. This means that a user may be a member of more than one group. Groups may be members of other groups provided they do not belong (directly or indirectly) to any of its members. The resulting group hierarchy has to be a directed acyclic graph. Figure 2 shows an example of a group hierarchy.

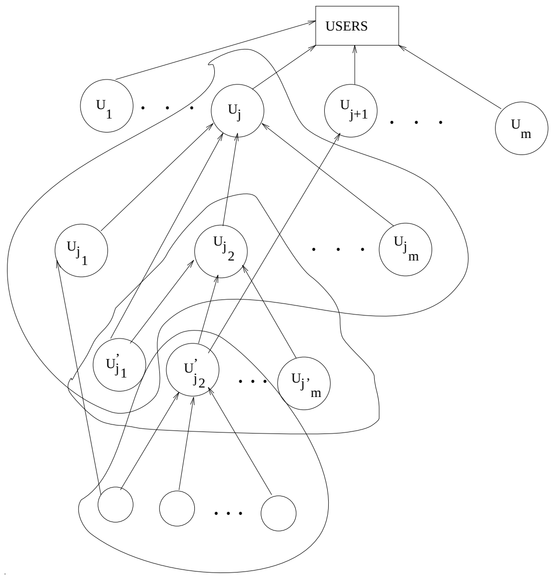


Figure 2: User groups hierarchy.

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

4.3 Implicit policies

There are two different types of object hierarchies in object-oriented database systems: *class-composition* and *class-inheritance hierarchy* [14]. To access the full information regarding an entity, a user is required to have the proper authorizations along the hierarchies. There are two policies: *visibility from above* and *visibility from below* that define how an explicit authorization may propagate along the hierarchies [15].

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

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

In order to indicate how privileges are propagated along the hierarchy, different types of authorization should be identified. Two possible types of authorization are: *partial*, and *full authorization* [18].

In the object-composition hierarchy, a user with the *full authorization* for a set of privileges (such as *read-definition, read* and/or *execute*) over an entity has the same rights to the entity and all its components. In the case of *partial authorization,* access to an entity does not extend to its components.

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

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

5 Notation, Assumptions, and Definitions

5.1 Notation

- O_i and OID_i are the names of the i-th object and the i-th object identifier, respectively. C_i is the name of the i-th class. E_i is the name of the i-th entity (object or class) which can be either O_i or C_i.
- Every user has a login-name and a corresponding login password. U_j denotes the login-name of the j-th user. PS_j denotes the login password of user U_j. The password is chosen by U_j, and is kept secret. Also PS_j is assumed to be long enough.
- $\|$ and $\oplus$ denote concatenation and exclusive-or (XOR), respectively.
- TM and DBMS denote a tamper-proof module and a database management system, respectively.
- AC denotes the authorization class which contains authorization information of the system. For a detailed definition, see Definition 7. An instance of AC is denoted by $ACID_{j,i,k}$, where the subscripts indicate that the user U_j has been granted authorization to the entity E_i by the user U_k.
- K_{db} is the database cryptographic key. The TM only can access the K_{db}. The TM uses the key to encrypt the authorization information in the class AC. $\{x\}_{K_{db}}$ denotes the ciphertext of x generated using the key K_{db}.

Note that we use n-bit strings to represent $OID_i, O_i, C_i, U_j, PS_j$, and $ACID_{j,i,k}$.

5.2 Assumptions

1. $F = \{F_n \mid n \in \mathcal{N}\}$ is a pseudo-random function family, where $F_n = \{f_K \mid f_K : \Sigma^n \to \Sigma^n,\ K \in \Sigma^n\}$. A pseudo-random function family can be constructed from pseudo-random string generators, and can be a one-way function. For a formal definition of a pseudo-random function family, the reader is directed to [25]

2. $H = \{H_n \mid n \in \mathcal{N}\}$, where $H_n = \{h \mid h : \Sigma^{2n} \rightarrow \Sigma^n\}$ is a k-SIFF mapping $2n$-bit inputs to n-bit output strings. k is a parameter which is chosen in such a way that no database entity has more than k relevant entities and no group has more than k users.
3. Random n-bit strings $K^{rd}, K^r, K^w, K^e, K^d, K^c$ correspond to *read-definition*, *read*, *write*, *execute*, *delete*, and *create*, respectively. These are stored in a protected memory, and are available to the TM only.

5.3 Definitions

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

Definition 2. A class C is represented by a tuple:
(CNAME, PNAME, "class-struct", "method-list", SECURITY-INFO).
Here *CNAME* is a unique name of C given by its creator. *PNAME* is the parent name of C. The *"class-struct"* is its structure, and *"method-list"* is the list of methods that can be executed by users if they have the *execute* privilege. *SECURITY-INFO* specifies class authorization information which is an aggregation of the *CKEYS-LIST*, and *H-FUNCTION*. *CKEYS-LIST* is a pair of access keys (K_i^P, K_i^F) corresponding to partial and full authorization. *H-FUNCTION* describes the hash function that must be used by the related classes to derive the access key K_i^F. □

Definition 3. The *class-struct* is $[P_1 : \rho_1(T_1), \ldots, P_i : \rho_i(T_i), \ldots]$, where P_i is a name of property, T_i is a type name of the respective property, and ρ_i is an optional type constructor, e.g. set-of, collection-of, array-of, ordered list, etc. The set of type names includes the names of atomic data types like integer, real, string, etc. as well as the names of classes that have been pre-defined. □

Definition 4. An object O is a tuple:
(OID, ONAME, CNAME, "state", SECURITY-INFO). Here *OID* is the identifier of the object and created by the DBMS. *ONAME* is the name of the object given by its creator. *CNAME* indicates the name of the class to which O belongs. *"state"* is the associated state of the object. *SECURITY-INFO* specifies object authorization information which is an aggregation of the *OKEYS-LIST*, and *H-FUNCTION*. *OKEYS-LIST* is a pair of access keys (K_i^P, K_i^F) corresponding to partial and full authorization. *H-FUNCTION* indicates the hash function that must be used by the related classes or objects to derive the access key K_i^F of the object O. □

The definitions of superclass and ancestor are as follows.

Definition 5. An ancestor of class C_i is any class C_k such that either
(1) $(C_k, PNAME, [P_1 : \rho_1(C_1), \ldots, P_i : \rho_i(C_i), \ldots], \ldots)$, or
(2) $(C_k, PNAME, [P_1 : \rho_1(C_1), \ldots, P_j : \rho_j(C_j), \ldots], \ldots)$ and C_j is ancestor of C_i. □

Definition 6. A superclass of C_i is any class C_k such that either
(1) $(C_i, C_k, \ldots)$, or
(2) $(C_k, C_j, \ldots)$ and C_j is a superclass of C_i. □

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

Definition 7. An Authorization Class (AC) is a tuple:
(GRANTEE, ENAME, GRANTOR, MEMBER-LIST, DAC-INFO).
Here *GRANTEE* indicates the user who is authorized to access the entity. *ENAME* specifies the entity which can be a class name or an object identifier. *GRANTOR* names the user who has authorized the *GRANTEE* to access the entity. *MEMBER-LIST* is the list of users who are the members of the group whose sponsor is the *GRANTEE. DAC-INFO* specifies DAC information and has the form:
(OP-RIGHTS, AUTH-TYPE, SPONSORSHIP, OWNERSHIP, H-FUNCTION, PSWORD). Here *OP-RIGHTS* indicates the list of privileges which the *GRANTEE* has on the entity (it could be read-definition, read, write, execute, delete, create, and all; the word *"all"* is used to indicate all possible access privileges). *AUTH-TYPE* (F or P) specifies full or partial authorization. *SPONSORSHIP* (YES or NO) indicates if the *GRANTEE* can be the sponsor of a group (or groups) (indicated by the *GRANTOR*), and is able to propagate his/her privileges to the group members. *OWNERSHIP* (YES or NO) specifies whether *GRANTEE* has ownership privilege. *H-FUNCTION* indicates the hash function that must be used to derive the grantor's password. *PSWORD* stores the user password. □

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

6 Proposed Solution

Our main goal is to design a cryptographic mechanism for discretionary access control in object-oriented database systems. Thus we will not consider other security issues such as authentication and secrecy of stored data. To enforce authentication and secrecy, the scheme proposed by Hardjono, Zheng and Seberry [9, 10] for database authentication based on SIFF can be applied. We assume that the user authentication is done by the underlying operating system, and is secure. Also, we use a tamper-proof module (TM) to perform all necessary cryptographic operations, to generate the needed cryptographic elements, and to verify the validity of access attempts. The security of the TM relies on the security of the underlying operating system and the DBMS. The TM can be an interface between the user and the database system, or between the database and physical layer, or a separate function in the database system. Figure 3 shows the position of the TM when it is a separate function in the DBMS.

The DBMS provides essential authorization information such as the entity identifier and access privileges in plain form and user password, access key, and coefficients of polynomial in SIFF in encrypted form to the TM. Then the TM evaluates the request according to the algorithms described in this section and Section 7, and passes the result to the DBMS.

To protect an entity against unauthorized access, the authorization system needs to know the user's authorization rights. There are two possible approaches

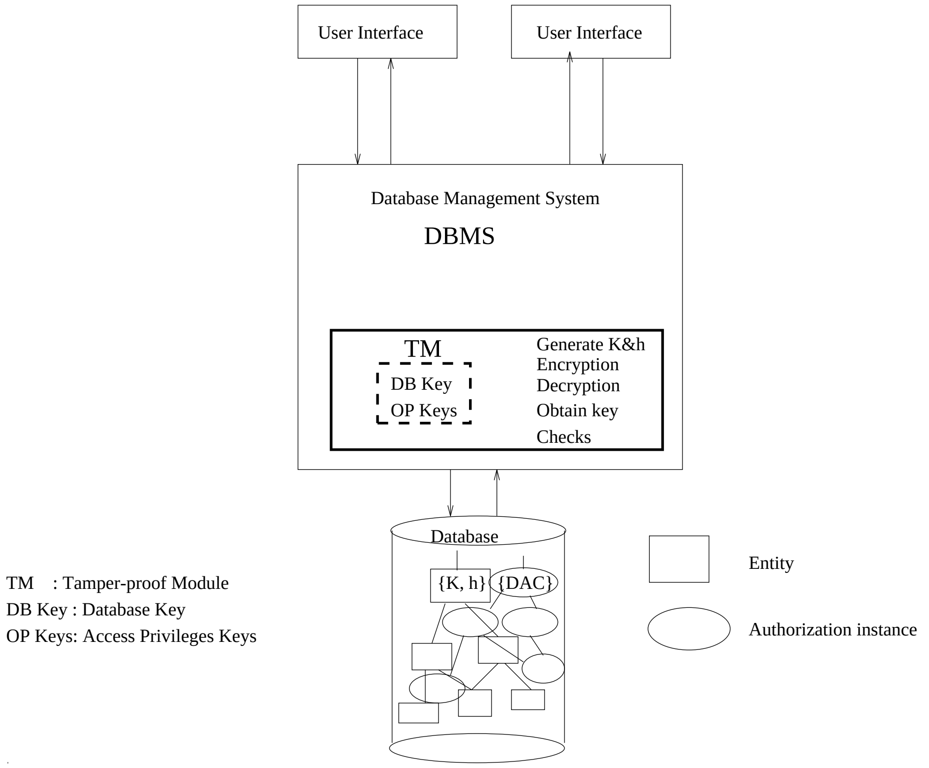


Figure 3: A possible implementation of the TM.

to accomplish this. In the first one, all authorizations, both explicit and implicit are stored. In the second one, only explicit authorizations are stored, and implicit ones are derived each time the access request is processed. The first approach is inefficient and time consuming when the number of object instances is large. The second approach is even worse if we use access control lists to store explicit authorizations. Here, we propose a cryptographic mechanism using SIFF to derive implicit authorizations from explicit authorizations which are stored in the authorization class AC.

To allow access to an entity i (object or class), we must be able to produce access keys K_i^P and K_i^F for the entity. Keys K_i^P and K_i^F correspond to the partial and full authorizations, respectively. K_i^F can be derived from the access key of the related objects. The relationship can be either the inheritance *(is-a)* or the aggregation *(is-part-of)*. In the case of inheritance, the access key K_i^F can be derived from the access keys of the instances of subclasses of the entity i. Whereas in the case of the aggregation, the key K_i^F can be computed from the access keys of the objects of ancestors of the entity i. In other words, the access key K_i^P guarantees secure explicit authorization access. The key K_i^F ensures implicit authorization rights along the *inheritance*, and *composite* hierarchies.

Every time a user requests access to a specific entity i either K_i^F or K_i^P is computed and compared with the stored one by the TM. If they match, the access is permitted, otherwise denied.

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

6.1 Creating

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

6.1.1 Case 1. Partial authorization.

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

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

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

6.1.2 Case 2. Full authorization.

Suppose that objects $O_{l_1}, O_{l_2}, \ldots, O_{l_p}$ with access keys $K_{l_1}^F, K_{l_2}^F, \ldots, K_{l_p}^F$ are related to the object O_i (via either inheritance or aggregation).
Step 1. The TM selects, at random, the access key K_i^F for the object O_i for full authorization.
Step 2. The TM selects, at random, a SIFF hash function $h_i^F \in_R H_n$ for the full authorization. The function should map the access keys of the respective objects for the *read-definition, read,* and *execute* privileges to the access key of the O_i, that is,

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

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

Step 3. The DBMS appends the hash function $\{h_i^F\}$ to the *H-FUNCTION* of the object O_i.

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

6.2 Authorization Administration

To be complete, an authorization system must include *grant, revoke,* and *ownership transfer* operations.

6.2.1 Granting

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

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

A polynomial of degree m with random coefficients should be selected. This step ensures that all grantees $U_{l_1}, U_{l_2}, \ldots, U_{l_m}$ (the grantees are members of the group whose sponsor is the grantor U_j) can directly compute the $ACID_{j,i,k}$ of the U_j and access the authorization-instance related to the U_j for the entity E_i granted by U_k.
Step 3. The TM encrypts the *DAC-INFO*, $\{($"access privileges", "P/F", "yes/no", "yes/no", $h_{j,i}, n_{l_s,i,j})\}_{K_{db}}$.
Step 4. The DBMS creates the object (U_{l_s}, E_i, U_j, *MEMBER-LIST, DAC-INFO*) as an instance of the class AC ($ACID_{l_s,i,j}$) for $s = 1, \ldots, m$.
Step 5. The DBMS updates the *MEMBER-LIST* of the authorization-instance related to the grantor U_j.

6.2.2 Revoking

If a grantor U_k revokes the privilege of U_j over the entity E_i, the following steps have to be completed.
Step 1. The DBMS deletes the associated authorization-instance $ACID_{j,i,k}$ from the AC.
Step 2. The TM selects a new SIFF with one less collision for the group whose sponsor is U_k (the user U_j no longer belongs to the group).
Step 3. The TM replaces the old SIFF in the authorization-instance associated with users in the *MEMBER-LIST* of the sponsor with the new one.
Step 4. The DBMS updates the *MEMBER-LIST* associated with U_k.

Section 9 discusses in detail the impact of the group updating on the authorization system.

6.2.3 Ownership Transfer

An entity (class or object) can have several owners who may act independently. Suppose that the creator of E_i is U_j and U_j wants to grant the ownership of E_i to users U_r and U_s by executing the *transfer-own* command. The following steps have to be completed.

Step 1. The TM computes passwords $n_{r,i} = f_{PS_r}(U_r \oplus E_i)$ and $n_{s,i} = f_{PS_s}(U_s \oplus E_i)$ for new owners.

Step 2. The TM selects a new SIFF hash function with the following collisions: $h_i^P(n_{j,i}\|K^{rd}) = h_i^P(n_{j,i}\|K^r) = h_i^P(n_{j,i}\|K^e) = h_i^P(n_{j,i}\|K^w) = h_i^P(n_{j,i}\|K^d) = h_i^P(n_{j,i}\|K^c) = h_i^P(n_{r,i}\|K^{rd}) = h_i^P(n_{r,i}\|K^r) = h_i^P(n_{r,i}\|K^e) = h_i^P(n_{r,i}\|K^w) = h_i^P(n_{r,i}\|K^d) = h_i^P(n_{r,i}\|K^c) = h_i^P(n_{s,i}\|K^{rd}) = h_i^P(n_{s,i}\|K^r) = h_i^P(n_{s,i}\|K^e) = h_i^P(n_{s,i}\|K^w) = h_i^P(n_{s,i}\|K^d) = h_i^P(n_{s,i}\|K^c) = K_i^P$

Step 3. The DBMS updates the instance in the AC for U_j and creates new instances for U_r and U_s.

Note that if the creator of E_i revokes the ownership privilege from the user U_s (by executing the *revoke-own* command), a new SIFF hash function with the following collisions have to be selected by the TM.
$h_i^P(n_{j,i}\|K^{rd}) = h_i^P(n_{j,i}\|K^r) = h_i^P(n_{j,i}\|K^e) = h_i^P(n_{j,i}\|K^w) = h_i^P(n_{j,i}\|K^d) = h_i^P(n_{j,i}\|K^c) = h_i^P(n_{r,i}\|K^{rd}) = h_i^P(n_{r,i}\|K^r) = h_i^P(n_{r,i}\|K^e) = h_i^P(n_{r,i}\|K^w) = h_i^P(n_{r,i}\|K^d) = h_i^P(n_{r,i}\|K^c) = K_i^P$
As a result, all privileges granted by U_s to other users will be deleted as well. If U_j is indicated as an owner of the E_i and both *GRANTEE* and *GRANTOR* are U_j, then U_j is considered the creator of the E_i.

Note that the polynomial (which is a part of SIFF) in Step 2 should be selected at least of degree 18, and the polynomial in Step 3 should be at least of degree 12.

7 Validation of Access Requests

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

In object-oriented database systems, as in relational databases, there is an SQL-like (structural query language) query language to retrieve data from the database system. Unlike relational databases, in object-oriented database systems, the hierarchical structure of an entity may or may not be included in the evaluation of the query. Hence, there are two forms of queries: *simple queries* and *hierarchical queries* [3, Chapter 3]. A *simple query* has the following form:

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

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

- **select** Target-clause* [**from** Entry-clause] [**where** Qualification-clause]; The syntax of the query is similar to a simple query. "*" indicates that the hierarchy must be included in the evaluation of the query, that is, the value of all properties (or objects) of the entity specified in the Target-clause and its relevant entities must be retrieved.

7.1 Access Validation

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

Without loss of generality, we assume that the user U_l issues the query: **select** E_j* **from** E_i;

7.1.1 Phase 1. User validation.

Step 1. In order to verify *read* privilege of the user U_l to E_j or E_i, the instance of the AC corresponding to either E_j or E_i for the user U_l should be retrieved. To search for such an instance whose value of the property $GRANTEE$ is U_l and $ENAME$ is E_j or E_i, the following query is issued by the DBMS.
- **select** AC **where**
$(GRANTEE = U_l$ **and** $ENAME = E_j$ **and** $PSWORD = f_{PS_l}(E_j \oplus GRANTOR))$
or
$(GRANTEE = U_l$ **and** $ENAME = E_i$ **and** $PSWORD = f_{PS_l}(E_i \oplus GRANTOR))$;
The verification that
$(PSWORD = f_{PS_l}(E_i \oplus GRANTOR))$, or $(PSWORD = f_{PS_l}(E_j \oplus GRANTOR))$ is done by the TM. If there is no such instance of the class AC, then the DBMS rejects the request, otherwise the corresponding instance is retrieved. Suppose the retrieved instance is: $(U_l, E_k, U_{l'}$, *MEMBER-LIST*,
$\{($"access privileges", "F", "yes/no", "yes/no", $h_{l',k}, n_{l,l',k})\}_{K_{db}})$, where k is either j or i.
Step 2. To access the entity E_k, the password associated with the owner of the entity has to be derived. The access key associated with the entity can be computed if the password of its owner exists (see Section 6.1 for details). So, the DBMS issues the following query to retrieve the password of the owner of the entity E_k:
- **while**$(OWNERSHIP \neq$ "yes"$)$**do select** $h_{l',k}(PSWORD)$;
Suppose that the derived password and SIFF for the owner U_w of E_k are $n_{w,k}$ and h^P_k, respectively (k is either j or i). Then we enter the access validation phase.

7.1.2 Phase 2. Access validation.

Assume that

h^{P^C} is SIFF hash function associated with a class,
h^{P^O} and h^{F^O} are SIFF hash functions used for an object (partial and full),
K^{P^C} and K^{F^C} are access keys used for a class (partial and full), and
K^{P^O} and K^{F^O} are access keys associated with an object (partial and full).

To retrieve the entity E_k, K_k^P or K_k^F corresponding to the partial and the full authorization must be computed. Then if the computed values are matched to the stored values in the entity, the access is permitted, otherwise is denied. To do so, the following steps are executed by the DBMS.

Step 1. One of the following queries depending on the type of the entity is executed by the DBMS.

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

Step 2. Retrieve objects which are in relation with the entity E_j (via either inheritance or aggregation). Let O_s denote such an object.
repeat
select O_s **where**
$(h_j^{P^O}(n_{w,j}\|K^r) = K_j^{P^O}$ **and** $AUTH\text{-}TYPE$ = "F" **and** $h_s^{F^O}(K_j^{F^O}\|K^r) = K_s^{F^O})$
or
$(h_j^{P^C}(n_{w,j}\|K^r) = K_j^{P^C}$ **and** $AUTH\text{-}TYPE$ = "F" **and** $h_s^{F^O}(K_j^{F^O}\|K^r) = K_s^{F^O})$;
until(there is no O_s);

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

8 Object Restructuring

In an object-oriented database system, objects or relationships might be deleted, added, or modified. In this section, we consider the impact of such operations on our authorization system.

8.1 Deletion of Objects

Objects in object-oriented database systems can be deleted indirectly by altering the database schema, or directly by using the *delete* privileges. For the indirect deletion, all objects of the database system need to be reorganized according to the new schema. For direct deletion, we consider three possibilities: deletion from a leaf node, deletion from an intermediate node, and deletion of a relationship. Deletion of an object from a leaf node requires the authorization-instances corresponding to the deleted object to be deleted from the AC. If an intermediate object which is a part of the composite object is deleted, then the descendants of the deleted object become the descendants of the parent of the deleted object. This requires the generation of a new SIFF function that satisfies Equation (2) of Section 6.1 and replaces the old SIFF function. If the deleted object is an instance of a subclass in the inheritance hierarchy, all objects of the lower subclasses of the deleted object are deleted. New SIFF hash function which satisfy Equation (2) of Section 6.1 must be produced for all objects of the superclasses of the deleted object and replace the old ones. In both cases, it is also necessary that the authorization-instances which correspond to the deleted object are deleted from the AC.

In an object-oriented database system, there are three types of relationships: (i) *aggregation* relationship; (ii) *generalization* (or *is-a*) relationship, and (iii) *association* relationship such as *teaches*, *is-taught-by*, *supplies*, *is-supplied-by*, etc. [22]. Note that the modification of the data model may cause deletion of relationships *aggregation* and *generalization*. The deletion of the *association* relationship may occur if the object is not associated with any objects.

Deletion of an *aggregation* relation may affect the composite object in two ways. First, a part of the component has been removed. In this case, the AC must be updated if the deleted part no longer exists in the database schema. Otherwise, due to the changes in the structure of the deleted part, new SIFF functions for all objects of the deleted part must be regenerated. The AC is left intact. Second, the changes in the hierarchy of ancestors. In this case, new SIFF functions for all the objects of the descendants must be reproduced.

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

8.2 Addition of Objects

An object might be added to the database as an object of an existing class, or as a new object of a new class. In the case of a new object of the old class, it is sufficient to complete the process described in Section 6.1. In the case of a new object of a new class, we can distinguish the following three possibilities: (i) a new class is added to a leaf, (ii) a new class is added to an intermediate level, and (iii) a new relationship is created.

If a class is added as a new leaf, then a process similar to the one described in Section 6.1 must be completed for all objects of the new class. Moreover, since the subclasses of the superclasses change, new SIFF functions for all object instances of the superclasses must be regenerated.

If a class is added to an intermediate node, first, the process described in Section 6.1 must be completed for objects of the new classes. Next, new SIFF functions must be regenerated (see Step 2 of Phase 2 in Section 6.1).

In the case of the addition of new relationships, a new SIFF function as described in Step 2 of Phase 2 of Section 6.1, must be regenerated for all objects of descendants or superclasses.

9 Grouping and Group Updating

When users have the grant authorization (specified by *SPONSORSHIP*), they can create user groups and become their sponsors. They can give the privileges to the members of the group by running the *grant command* (see Section 6.2.1). Members of a group with the grant option (i.e., if *SPONSORSHIP* is on) can propagate privileges to other users. They have a user group hierarchy similar to the one shown in Figure 2. An important issue in the group hierarchy is that of group updating.

We can distinguish three possible cases:

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

The impact of each modification on the group organization and the necessary updates are as follows.

9.1 Deletion of Memberships

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

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

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

9.2 Addition of New Memberships

A user who has the sponsorship privilege gives his/her privileges to a new user. The new user can be an individual user or the sponsor of a group. In every case, it is required that the process described in Section 6.2.1 be completed.

9.3 Replacing

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

10 Security of the Authorization System

The authorization based on SIFF is a complex system whose proof of security is a difficult (or perhaps impossible) task. We are going to give some plausible arguments to support our claim that the proposed authorization system is "secure".

There are two classes of users in an authorization system. The first one consists of users who have been admitted by the security officer as users of the system. They are called insiders. The second one includes all users who are not part of the system. They are called outsiders. The authorization system is secure if any insider who wants to access entities outside their privileges, may succeed with a very small probability (say 2^{-64}).

Claim 1 *Assume that the tamper-proof module (TM) is accessible to the DBMS only, and the computational power of an insider is polynomially bounded. If the SIFF scheme, the pseudo-random functions, the user authentication scheme, and the cryptosystem used for encryption are all secure, then the authorization system is secure as well.*

Justification:

There are two possibilities for an insider to have an unauthorized access. The first one is impersonation. This means that the intruder can guess or disclose the login password of the owner or one of the grantees of entity j, say U_l, so the intruder can compute $n_{l,j} = f_{PS_l}(U_l \oplus EID_j)$ and access authorization-instance associated with user U_l. This contradicts that the user authentication system is secure. The second possibility is either to access unauthorized objects or to exercise nonexistent privileges. In the first case, this means that either the intruder generates a valid access key and an associated SIFF function, and an access privilege key for entity j, K_j^P, h_j^P, and K^{op}, or the intruder who is authorized to access a component of the object hierarchy, accesses high-level objects. This means that the intruder is able to predict the output of the pseudo-random function and to find collisions for the SIFF function. This contradicts the assumption that the pseudo-random function and SIFF are secure. In the second case, this means that the intruder is able to modify his/her privileges or someone else's. For example, the intruder modifies partial authorization to full authorization, changes ownership, changes the *read* privilege to *write* privilege, etc. If this happens, the ability of the intruder is equivalent to breaking the cryptosystem which is used for encryption of *DAC-INFO*. This again contradicts our assumption.

For an outsider, there are two possibilities as well. The intruder finds a valid password, and enters the system as a legal user. If this happens, this means that the intruder has broken the user authentication system, which contradicts our assumption. The second one is the intruder bypasses the database management system, and accesses the data file directly. This means that either the operating system is not secure or that the access control to the data files fails.

11 Complexity of The System

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

In 1982, Akl and Taylor [1] were the first to propose a solution to the *hierarchical access control problem.* Their solution was based on the RSA cryptosystem [19]. There are several problems with this scheme. The scheme can work only with a rigid hierarchical structure and cannot be used in object-oriented database systems where the database schema may evolve. Each node stores two integers. The first integer is a prime assigned to the node and the second integer is product of primes associated with other nodes that are not descendants of the given node. The average length of the second component is large and hence expensive in terms of storage. Moreover, the entire system must be predefined by a trusted central authority, and there is no way to expand or modify it according to changes of the hierarchy. Some other solutions to overcome these problems have been proposed [4, 12]. A common drawback of these solutions is that they are based on the difficulty of breaking the RSA cryptosystem, and make heavy use of the underlying algebraic properties of the crypto-function.

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

Let us give a brief comparison of the time and space complexity of the SIFF construction with the RSA one.

Let k be the number of objects in the hierarchical structure. Let n be the length of keys. Assume that MD5 and polynomials of degree k over finite fields $GF(2^n)$ ($n = 128$) are used to construct the SIFF hash function. The system based on the SIFF requires $O(\log k)$ modular multiplications of 128 bits long for key derivation (see Appendix A). Note that, because computation time of the pseudo-random function and MD5 is negligible, it is ignored. If the RSA approach is used for authorization derivation and generation, the time complexity of the system will be $O((k+1)\log k)$ modular multiplications of n bit long integer; n must be at least 512 bits long, that is four times more. If the computation time of integers associated with objects is added up, then the time complexity will be much higher.

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

Finally, in order to increase the efficiency of the system and to get benefit of the order access privileges, the set of access privileges is partially ordered such that the lower access privilege can be inferred from higher access privileges. The access privilege keys $(K^{rd}, K^r, \ldots, K^c)$ can be chosen so that lower keys are computable from higher keys. This results in having a shorter list of privileges, and the SIFF function has smaller number of collisions. For example, the Equation (1) in Section 6.1 can be simplified as:

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

12 Conclusion and Remarks

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

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

Some aspects of the presented solution in this paper need further investigation. There is a need to implement a prototype of our proposal in order to investigate the applicability, efficiency, and performance of the access control mechanism. This will give a clearer picture on how complex the administration becomes in real life. Moreover, insights may be gained into what other requirements are essential for a successful cryptographic mechanism. We assumed that there are two types of access: partial and full. In the full authorization, a lower-node is accessed by all higher-nodes of hierarchical structure. In the partial authorization, a specific node is only accessed. There is also a need for further investigation of a situation where a lower-node in a hierarchical structure may be accessed only by some higher-nodes.

APPENDIX A. An Improvement of the Construction of k-SIFF

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

$$P(x) = a_0 + a_1x + \ldots + a_{k-1}x^{k-1} = (x + b_0)(x + b_1)\ldots(x + b_{k-1})$$

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

When it is evaluated, the evaluation costs k modular multiplications, $O(k)$. The number of modular multiplications can be reduced if the evaluation of the polynomial is done as shown in Figure 4. The OWHF is any one-way hash function such as MD4, MD5 [20, 21], or HAVAL [26]. The UHF is any universal hash function with the collision accessibility - this is our polynomial $P(x)$

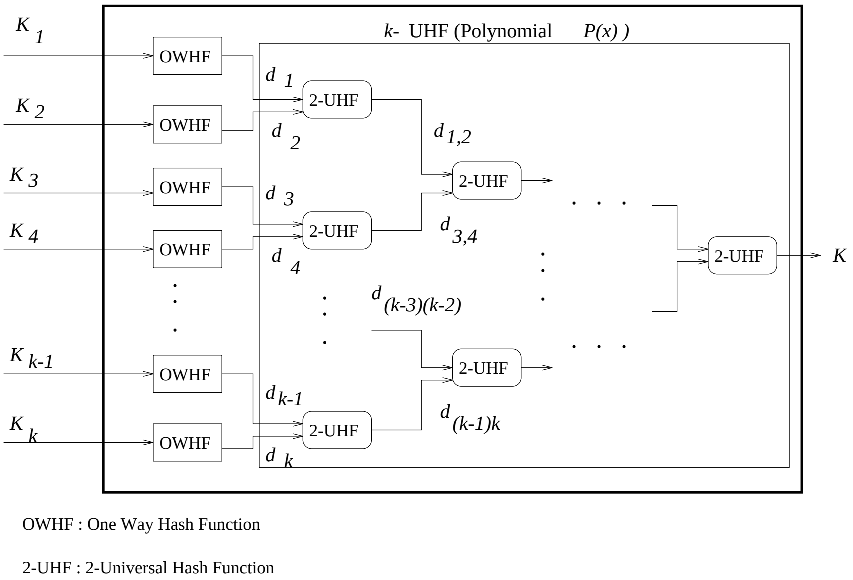


Figure 4: Some improvement of implementation of k-SIFF.

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

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

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

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

References

1. S. G. Akl and P. D. Taylor. Cryptographic Solution To A Multilevel Security Problem. In D. Chaum, L. Rivest, and A. T. Sherman, editors, *Advances in Cryptology Proceedings of CRYPTO'82*, pages 237–250. Plenum Press, NY, August 1982.
2. E. Bertino, F. Origgi, and P. Samarati. A New Authorization Model for Object-Oriented Databases. In J. Biskup, M. Morgenstern, and C. E. Landwehr, editors, *Database Security VIII (A-60)*, pages 199–222. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1994.
3. Elisa Bertino and Lorenzo Martino. *Object-Oriented Database Systems: Concepts and Architectures.* International computer science series. Addison-Wesley, 1993.
4. G. C. Chick and S. E. Tavares. Flexible Access Control With Master Keys. In G. Brassard, editor, *Advances in Cryptology Proceedings of CRYPTO'89*, pages 316–322. Springer-Verlag, 1990.
5. Hans Dobbertin. Cryptanalysis of MD4. In D. Gollmann , editor, *Fast Software Encryption*, volume 1039 of *Lecture Notes in Computer Science*, pages 53–69. Springer-Verlag , 1996.
6. Hans Dobbertin. Cryptanalysis of MD5 Compress. Annoucement, May 1996.
7. E. B. Fernandez, R. C. Summers, and C. Wood. *Database Security and Integrity.* Addison-Wesley Publishing Company, 1981.
8. E. Gudes, H. Song, and E. B. Fernandez. Evaluation of Negative, Predicate, and Instance-based Authorization in Object-Oriented Databases. In S. Jajodia and C.E. Lanwehr, editors, *Database Security IV*, pages 85–98. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1991.
9. T. Hardjono, Y. Zheng, and J. Seberry. A New Approach to Database Authentication. In *Research and Practical Issues in Databases: Proceedings of the Third Australian Database Conference (Database'92)*, pages 334–342, 1992.
10. T. Hardjono, Y. Zheng, and J. Seberry. Database authentication revisited. *Computers & Security*, 13(7):573–580, 1994.
11. Thomas Hardjono. *Applications of Cryptography for the Security of Database and Distributed Database Systems.* PhD thesis, University College, University of NSW, Sydney, Australia, 1991.
12. L. Harn, Y.-R Chien, and T. Kiesler. An Extended Cryptographic Key Generation Scheme For Multilevel Data Security. In *Proceedings of the IEEE Computer Society Symposium on Security and Privacy*, Oakland, CA., May 1990. IEEE Computer Society Press.

13. R. Impagliazzo, L. Levin, and M. Luby. Pseudo-Random Generation from One-Way Functions. In *Proceedings of the 21st ACM Symposium on Theory of Computing*, pages 12–24. , 1989.
14. Won Kim. Object-Oriented Databases: Definition and Research Directions. *IEEE Transactions on Knowledge and Data Engineering*, 2(3):327–341, September 1990.
15. M. M. Larrondo-Petrie, E. Guides, H. Song, and E. B. Fernandez. Security Policies in Object-Oriented Databases. In D. L. Spooner and Landwehr, editors, *Database Security III*, pages 257–269. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1990.
16. T. F. Lunt. Multilevel Security for Object-Oriented Database Systems. In D. L. Spooner and Landwehr, editors, *Database Security III*, pages 199–209. Elsevier Science Publishers B. V. (North-Holland) IFIP, 1990.
17. M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the 21st ACM Symposium on Theory of Computing*, pages 33–43. ACM Press, 1989.
18. F. Rabitti, E. Bertino, W. Kim, and D. Woelk. A Model of Authorization for Next-Generation Database Systems. *ACM Transactions on Database Systems*, 16(1):88–131, March 1991.
19. R. L. Rivest, A. Shamir, and L. Adleman. A Method For Obtaining Digital Signatures And Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–128, 1978.
20. Ronald L. Rivest. The MD4 Message Digest Algorithm. In *Advances in Cryptology, Proceedings of CRYPTO'90*, pages 281–291. Springer-Verlag, 1990.
21. Ronald L. Rivest. The MD5 Message Digest Algorithm. MIT Laboratory for Computer Science and RSA Data Security, Inc., Request for Comments (RFC), 1992.
22. J. Rumbaugh, M. Blaba, W. Premerlani, F. Eddy, and W. Larensen. *Object-Oriented Modeling and Design.* Printice Hall Inc., 1991.
23. Yair Wand. A Proposal for a Formal Model of Objects. In W. Kim and F. H. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, pages 537–559. Addison-Wesley, Reading, Massachusetts, ACM Press, 1989.
24. 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.
25. 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.
26. Y. Zheng, J. Pieprzyk, and J. Seberry. HAVAL- A One-Way Hashing Algorithm with Variable Lenght of Output (Extended Abstract). In *Advances in Cryptology, Proceedings of AUSCRYPT'92*, volume 718 of *Lecture Notes in Computer Science*, pages 83–104. Springer-Verlag, 1992.