

Key Assignment for Enforcing Access Control Policy Exceptions in Distributed Systems[★]

Jyh-haw Yeh^{a,*}, Randy Chow, Richard Newman^b

^a*Dept. of Computer Science, Boise State University, Boise, ID 83725, USA*

^b*Dept. of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611, USA*

Abstract

A cryptographic key assignment scheme is proposed to enforce access control policies in which antisymmetric and transitive exceptions are included, in addition to the policies with partial ordered set (POSet) properties. In current literature, all proposed cryptographic key assignment schemes assume a user hierarchy model which can only enforce policies with POSet properties. The POSet properties are suitable for hierarchical systems. However, there are many systems, especially distributed systems handling indirect remote accesses, that cannot be modeled as a strict hierarchy. A new access control model named user hierarchy-with-exception and its enforcing key assignment scheme are proposed for those systems.

There is only one key assigned to each user class in enforcing the user hierarchy model. The cost to achieve our more powerful scheme in the user hierarchy-with-exception model is one more key for each user class to memorize or one more step to access its own data.

Key words: Key Assignment, Key Derivation, Access Control Policy, Hierarchical with Exceptions

[★] The research is partially supported by NSA under contract number: MDA 904-98-C-A892

* Corresponding Author: Jyh-haw Yeh, MEC 302B, Dept. of Computer Science, Boise State University, 1910 University Drive, Boise, Idaho 83725.
Tel: (208) 426-3034, Fax: (208) 426-2470, Email:jhyeh@cs.boisestate.edu

1 Introduction

If system users are divided into distinct groups or classes, $C = \{C_1, C_2, \dots, C_n\}$, each class should have an encryption key to protect data owned by the users in this class. Each class in the system would have a different clearance and, consequently, a different accessible set. In such a system, the accessible set of a class C_i is the set of classes whose data can be accessed by C_i . An access control policy of a system defines the accessible sets of all classes. The objective of this paper is to design a key assignment scheme so that the access control policy can be enforced.

The most straightforward implementation of this problem requires classes to memorize all encryption keys of their accessible sets. This approach scales poorly. Akl and Taylor first proposed an elegant key assignment scheme [1] to overcome this dilemma, but their scheme only enforces policies in a hierarchical structure. An access control policy expressed as a user hierarchy defines the set C as a partially ordered set (POSet) on an accessible relation. We will describe the properties of a POSet in Section 2 and their key assignment scheme in Section 6.1. There are many papers [2–13] that address the key assignment for access control in a user hierarchy (POSet). But in practice, some systems may need more complex access policies which can not be expressed as a user hierarchy. These complex access control policies may violate the antisymmetric and transitive properties of POSet.

For example, suppose a system contains raw data owned by a class C_3 that should remain secret from ordinary users in a class C_1 . A special class C_2 exists of users who can pre-process the raw data and translate the data into another form before presenting them to class C_1 users. Namely, C_1 can access C_2 and C_2 can access C_3 , but C_1 cannot access C_3 . This transitive exception policy is very common and important for real systems. Moreover, the anti-symmetric property of POSets forces all classes in an accessible cycle to be equivalent, placing too many restrictions on a system's group/class partition. In Section 4, a distributed static database example is provided to show the importance of transitive and antisymmetric exceptions in real practice. Therefore, a policy model based on a hierarchical structure with exceptions, and its

policy-enforcing key assignment scheme are proposed in this paper. Compared to schemes for the user hierarchy model, the cost to achieve our more powerful scheme for user hierarchy-with-exception model is one more key for each user class to memorize or one more step to access its own data.

In most systems, the access rights usually include read and write accesses. As a result, each class may have different read and write accessible sets. All key assignment schemes proposed in the literature were designed to enforce read accesses only. None of them address how to enforce write accesses. Intuitively, write accesses differ from read accesses on the direction of information flow. That is, information is flowing from C_j to C_i if C_i reads C_j and is flowing on the opposite way, from C_i to C_j , if C_i writes C_j . We observe that write accesses could also be enforced by just using a different way to apply those schemes. Section 3 will address this issue in detail.

This paper is organized as follows: Section 2 shows the access control policies in a user hierarchy-with-exception model is a superset of those policies in a user hierarchy model. Section 3 describes how a scheme enforces read and write accesses simultaneously. Section 4 provides an interesting distributed static database example whose access control policy has antisymmetric and transitive exceptions. Section 5 discusses the related works with respect to several criteria. All of the existing schemes assume the user hierarchy model. Section 6 gives a brief description of Akl and Taylor's scheme. Section 7 describes the new cryptographic key assignment scheme in the user hierarchy-with-exception model. Finally, Section 8 concludes the paper.

2 Access Control Policies

Let an n -system be a user system C with n distinct user classes $C = \{C_1, C_2, \dots, C_n\}$. An access control policy for an n -system specifies rules that regulate the information flow among these n different user classes. In this section, we show that access control policies in the user hierarchy-with-exception model are a superset of those in a user hierarchy model. To describe the access control policies in both models, first some definitions are necessary.

Definition 2.1 *Three properties must hold for an n -system C to be a POSet on a relation R :*

1. *Reflexive property: C_iRC_i , where $C_i \in C$.*
2. *Antisymmetric property: C_iRC_j and $C_jRC_i \Rightarrow C_i = C_j$, where C_i and $C_j \in C$.*
3. *Transitive property: C_iRC_j and $C_jRC_k \Rightarrow C_iRC_k$, where C_i, C_j and $C_k \in C$.*

Definition 2.2 *An accessible set of a class C_i in an n -system C is the set $AS_i = \{C_j : C_i \text{ can access } C_j\}$.*

Definition 2.3 *An dominating set of a class C_i in an n -system C is the set $DS_i = \{C_j : C_j \text{ can access } C_i\} = \{C_j : C_i \in AS_j\}$.*

Definition 2.4 *An access control policy of an n -system C is the collection of all accessible sets $\{AS_1, AS_2, \dots, AS_n\}$.*

A user hierarchy model for an n -system preserves the three properties of POSet on the accessible relation. Thus, any access control policy $\{AS_1, AS_2, \dots, AS_n\}$ in the user hierarchy model has following properties: For $i, j, k = 1, 2, \dots, n$

1. $C_i \in AS_i$ (*reflexive property*)
2. If $C_j \in AS_i$ and $C_i \in AS_j \Rightarrow C_i = C_j$ (*antisymmetric property*)
3. If $C_j \in AS_i$ and $C_k \in AS_j \Rightarrow C_k \in AS_i$ (*transitive property*)

On the other hand, any access control policy in the user hierarchy-with-exception model for an n -system only needs to satisfy two properties:

1. $C_i \in AS_i$ (*reflexive property*)
2. If $AS_i = AS_j$ and $DS_i = DS_j \Rightarrow C_i = C_j$. (*equivalence property*)

The combination of reflexive and antisymmetric properties in the user hierarchy model also implies the equivalence property.

Proposition 2.1 *If an access control policy has the reflexive and antisymmetric properties, then it also has the equivalence property.*

Proof:

Assume that an access control policy P has no equivalence property. That is, $AS_i = AS_j$ and $DS_i = DS_j$, but $C_i \neq C_j$.

Since $C_i \in AS_i$ and $C_j \in AS_j$. (reflexive property)

We have $C_i \in AS_j$ and $C_j \in AS_i$, but $C_i \neq C_j$. Thus, P has no antisymmetric property.

Any access control policy in the user hierarchy model satisfies reflexive, anti-symmetric, and transitive properties. By Proposition 2.1, this access control policy also satisfies reflexive and equivalence properties and must belong to the user hierarchy-with-exception model. Therefore, the access control policies in the user hierarchy-with-exception model are a superset of the access control policies in the user hierarchy model.

Both the antisymmetric and equivalence properties are rules to decide the equivalence of two classes. In terms of the conditions for equivalence testing, the equivalence property is more relaxed than the antisymmetric property. This relaxation is because the absence of transitive property in the user hierarchy-with-exception model makes possible different accessible and dominating sets for two classes, though they could access to each other. The relaxation gives a system the opportunity to have a more flexible class/group partition, i.e., two classes that have the access right to each other are no longer necessarily equivalent. This antisymmetric exception is extremely useful for a system handling indirect remote accesses in a distributed environment.

In order to get a clearer view of these two models, a directed graph (DG) representation of access control policies in both models is discussed below. A node in a DG represents a class in a system. Two kinds of edges, positive and negative edges, are used in the DG. The positive edges indicate that the source nodes are allowed to access the destination nodes, whereas the negative edges indicate that the accesses are prohibited. If there is a conflict between these two kinds of edges, negative edges take precedence. The other difference between them is that positive edges have transitive property but negative edges do not. That is, a node A can access another node B if there exists a positive path (sequence of positive edges) from A to B . On the other hand,

the negative path (sequence of negative edges) from A to B does not mean A cannot access B if the path length is greater than one. Basically there are two conditions to decide the accessibility for node A to node B .

Condition 1: If there exists a positive path (a sequence of positive edges) from A to B .

Condition 2: If there exists a negative edge from A to B .

If Condition 1 is true, but not Condition 2, then A can access B . On the other hand, if Condition 1 is false or Condition 2 is true, then A can not access B . Having this kind of DG in mind, it is possible to discuss the policy models. For the user hierarchy model, the DGs must obey the three properties of POSets.

- 1. Reflexive Property: Each node should have a positive edge to itself. It means that each class can access its own data.*
- 2. Transitive Property: A node can access another node if there exists a positive path to that node.*
- 3. Antisymmetric Property: Two nodes are equivalent if both of them have a positive path to the other. In general, a positive path cycle in a DG means all nodes in this cycle can access each other and should be the same class. Thus, for an n -system, the n node DG must be acyclic.*

Obviously, the negative edges do not exist in the user hierarchy model, since their purpose is to kill transitive property. It is used only in the user hierarchy-with-exception model. For user hierarchy-with-exception model, the DGs have two properties.

- 1. Reflexive Property: Each node should have a positive edge to itself.*
- 2. Equivalence Property: Two nodes are equivalent if both of them have a positive path to each other and have the same immediate predecessors and successors for the negative edges.*

Note that in the user hierarchy-with-exception model, the usage of negative edges is to break the transitive property of positive edges. Therefore, a policy with transitive exceptions can be included.

Two access control policy examples that can be handled in a user hierarchy -with-exception model, but not in a user hierarchy model are given in Figure 1 and Figure 2. For clarity, self-pointing edges for the reflexive property in the DG representation are omitted for rest of the paper.

In Figure 2, there is a positive cycle $C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow C_1$. These three classes are not equivalent in the user hierarchy-with-exception model because $AS_i \neq AS_j$ and $DS_i \neq DS_j$ for $i \neq j$ and $1 \leq i, j \leq 3$. Two transitive exceptions also exist in this example. One is $C_2 \rightarrow C_3$ and $C_3 \rightarrow C_1$, but $C_2 \not\rightarrow C_1$. The other is $C_3 \rightarrow C_1$ and $C_1 \rightarrow C_2$, but $C_3 \not\rightarrow C_2$. These two transitive exceptions allow different accessible sets and dominating sets for these three classes. Therefore, an access control policy with the antisymmetric exceptions should also have the transitive exceptions. Note that for any two classes which can access each other, transitive exceptions may cause different accessible sets with the same dominating sets for these two classes, or the other way around. In Figures 3 and 4, classes C_1 and C_2 are not equivalent because of $AS_1 \neq AS_2$ but $DS_1 = DS_2$, and $DS_1 \neq DS_2$ but $AS_1 = AS_2$, respectively.

An access control policy $\{AS_1, AS_2, \dots, AS_n\}$ of an n -system can be represented as an $n \times n$ matrix A , where

$$A_{ij} = \begin{cases} 1 & \text{if } C_j \in AS_i \\ 0 & \text{otherwise} \end{cases}$$

For example, the access control policy in Figure 2 can be represented as Table 1. This policy matrix A is different from the access control matrix (ACM) of HRU [16] model. The ACM is a matrix correlating the subject, object and the authorizations owned by each subject on each object. The rows of ACM correspond to subjects and the columns to the objects. Each element $ACM[i, j]$ contains the access modes for which the subject i is authorized on object j . However, the policy matrix in the user hierarchy or in the user hierarchy-with-exception models merges subjects and objects with the same security level into a class. Both the rows and columns of the matrix A correspond to the classes. The element A_{ij} of matrix A denotes the access authorization between classes C_i and C_j . $A_{ij} = 1$ indicates that the class C_i has the privilege to access the

class C_j , $A_{ij} = 0$ otherwise.

3 Read and Write Accesses

In a real system, there may have different access control policies for read and write accesses. That is, two different policy matrices A^r and A^w for read and write accesses respectively. The typical example is a system using the Bell-La Padula model [17] to regulate information flow. In this model, There are two properties characterizing the secure flow of information.

Simple Security Property: A subject in a class C_i has the read access to an object in another class C_j only if C_i 's security level is higher then or equal to C_j 's.

***-Property:** A subject in a class C_i has the write access to an object in another class C_j only if C_i 's security level is lower than or equal to C_j 's.

Based on the above properties, any subject in the system can not read-up and write-down to forbid sensitive information leaking into lower security levels. Obviously, such systems must have separated and different policy matrices for read and write.

All key assignment schemes assign keys to users. keys may be derivable from other keys based on the accessible relationship among their owners. Most of the schemes assign keys in a way that a class C_j 's key can derive another class C_i 's key if C_j is allowed to access C_i . Therefore, access control policies can be enforced by a sequence of encryption, decryption and derivation of these keys. In order to enforce two different policy matrices for read and write, two keys are necessary for each class. One key denoted as K^r is for read and the other key denoted as K^w is for write. Each class C_i uses the read key K_i^r to encrypt its own data. Every other class C_j having the read access right on C_i retrieves data from C_i by the following sequence of actions.

1. C_j makes a read request to C_i .
2. C_i sends back the encrypted data for C_j to read.
3. C_j derives C_i 's read key K_i^r from its own key K_j^r .

4. C_j uses the key K_i^r to decrypt the received data.

To perform write accesses, the write key K^w should be used. If C_j has the write access to C_i , then C_j initiates a write access to C_i by the following sequence of actions.

1. C_j makes a write request which contains the data to be written to C_i , where the data appended with C_j 's ID is encrypted by C_j 's write key K_j^w . C_j also sends its ID in cleartext in the packet. The purpose of sending ID both in cleartext and cyphertext is to avoid replay attacks.
2. C_i derives C_j 's write key K_j^w from its own key K_i^w .
3. C_i uses the key K_j^w to decrypt the received data.
4. C_i re-encrypts the data by its own read key K_i^r and perform write operation.

The proposed scheme in this paper uses the same scenario as other schemes to assign keys so that the read and write access policies can be enforced simultaneously. The difference is that our scheme is the first one capable of enforcing policies with exceptions. Section 7 will describe the new scheme in detail. But first, a distributed static database system is presented in the next section to show the importance of policy exceptions. After the discussion of read and write in this section, there should have no ambiguity to simply use the term “access” to represent either read or write or both for the rest of the paper.

4 Policy Exception Example: A Distributed Static Database System

Consider a static database that contains some tables with confidential records. Users can only make a request to a specific query processor to retrieve some aggregate information in these tables. Table 2 EMPLOYEE gives an example of confidential records in a static database system. A user in this system can not directly read the confidential records. Instead he or she can make requests such as “How many employees are ranked 1 ?” or “How many employees have salaries of \$60,000 or higher ?” to the query processor. The query processor

goes over the EMPLOYEE table and answers the user: “There are two employees are ranked 1” and “There are two employees with salaries 60,000 or higher”.

In this example, there are three classes: C_1 for users, C_2 for the query processor, and C_3 for the EMPLOYEE table. The EMPLOYEE table is encrypted to prevent direct access from C_1 . C_1 makes a request to C_2 , and C_2 derives the encryption key of C_3 to decrypt the EMPLOYEE table. C_2 sends the result encrypted by its own encryption key to C_1 . Then C_1 decrypts it and gets the result. A transitive exception exists among these three classes. That is, C_1 can access C_2 , C_2 can access C_3 , but C_1 cannot access C_3 . Encryption of the result is necessary whenever there are several classes of users and each user class only can retrieve the static information over some confidential tables through a different query processor. For simplicity, we assume there is only one user class and one query processor.

Next, the example is extended to a distributed environment to show that the antisymmetric exception is also important. Suppose a company has distributed sites connected by a network. Each site has its own confidential tables and three classes as above. A user who wants remote access to the static information from another site is not allowed to make requests directly to the remote query processor. The request must go to the local query processor first, which decides whether to forward the request to the remote query processor. If the request is forwarded, the remote query processor prepares and encrypts the result and sends it back to the local query processor. The local query processor first decrypts the answer, then re-encrypts the answer so that the user can decrypt it. The two-site example is depicted in Figure 5 (Eight negative edges representing the transitive exceptions are omitted here for clarity). The corresponding access control policy for this example is shown in Table 3.

In this example, both query processors in sites A and B can access each other, but they are not equivalent. There are eight transitive exceptions and one antisymmetric exception in this example. We will show that this example can be handled by our key assignment scheme in Section 7, but not by other schemes in the literature.

5 Related Works

The cryptographic key assignment problem for a system to enforce the access control policy was first investigated by Akl and Taylor [1]. They proposed an elegant solution to enforce access policies modeled as a partial order set. Their scheme is simple with respect to the key generation and derivation procedures, but suffers high memory overhead, weak appendability and restricted (hierarchy-based) set of policies. The appendability weakness means that the keys for existing classes need to be re-computed when a new security class is added. For a decade, researches on this key assignment problem concentrate on finding solutions to reduce memory overhead and increase the appendability. As the computer technology evolving toward distributed computing, the user hierarchy model becomes even restricted to model the needs for distributed systems. This paper intends to develop a key assignment scheme which can enforce a richer set of policies for distributed systems.

Mackinnon et al. [2] presented an improved scheme using canonical assignment to reduce the memory overhead. Harn and Lin [4] proposed another scheme in which the key assignment is handled by a bottom-up approach, while [1] and [2] are top-down approaches. The appendability weakness is addressed by Sandhu [3], Harn, Chien and Kiesler [5], Liaw [7], Hwang et al. [8], Wu et al. [9], Tsai et al. [10]. This appendability problem is difficult to solve and none of existing schemes has achieved complete appendability. Some solutions to reduce the complexity are to limit the access control model to a tree structure, which is a special case of the hierarchy model. Others reduces the re-computation to a smaller set of existing keys with different degrees of compensation. In [3], an ID-based scheme is presented by iteratively applying one-way functions to solve the problem only for tree like structure. Harn, Chien, and Kiesler [5] presented a scheme by dividing a system into several subgroups to reduce the affected keys while a new class is added. Each subgroup has a root class responsible for replying a request from higher subgroup. Liaw's [7] scheme which is based on RSA public key strategy increases the appendability, but it still suffers the key re-computation overhead when the added class has a high security level. Hwang et al [8] presented a scheme in

which the addition/deletion of a class only needs to modify the keys of immediate higher classes. In their scheme, a class with r immediate lower classes must assign a unique integer from 1 to r to each immediate lower classes. In order to derive the immediate lower classes' keys, each class needs to memorize the integer identifiers of them. Wu et al [9] scheme based on the Chinese Remainder theorem achieves the appendability without changing other existing keys, but a set of public parameters needs to be modified. Tsai et al [10] proposed a scheme based on the Rabin's public key system and the concept of the Chinese Remainder theorem and Newton's interpolating method. Their scheme still requires to update the secret interpolating polynomials of those security classes which have the access privilege on a new class, though no existing keys need to be modified.

The other criterion to evaluate a scheme is the efficiency of key derivation. [1], [2], [4], [7], [9], [10] derive a subordinate key directly, but [3], [5], [8] need iterative derivation.

Although the new scheme proposed in this paper has the same advantages and disadvantages as in [1], a new and important capability for our scheme which broadens the access control from a user hierarchical structure to a user hierarchy structure with exceptions has never been addressed before.

6 Akl and Taylor's Key Assignment Scheme

6.1 Key Assignment and Derivation

In Akl and Taylor's key assignment scheme, there is a central authority C_0 . In an n -system, C_0 must generate a set of encryption keys K_1, K_2, \dots, K_n and send K_i to each class C_i securely. Each class C_i can use its own key K_i and some public information T_i and T_j to derive the class C_j 's key K_j if C_i is allowed to access C_j . C_0 proceeds the following steps:

1. Generate a secret number K_0 .
2. Compute the product M of two large prime numbers and make it public.

For $i, j = 1, 2, \dots, n$

3. Assign a distinct prime number P_i for each class C_i .
4. Compute public information $T_i = \prod_{A_{ij}=0} P_j$, where matrix A is defined in Section 2.
5. Assign each $K_i = K_0^{T_i} \bmod M$.

The class C_i uses the key derivation function,

$$f(K_i, T_i, T_j) = K_i^{T_j/T_i} \bmod M,$$

to derive C_j 's key K_j , where

$$K_j = K_0^{T_j} = (K_0^{T_i})^{T_j/T_i} = K_i^{T_j/T_i} \bmod M.$$

The derivation function $f(K_i, T_i, T_j)$ is feasible to compute if and only if T_j/T_i is an integer (see proof in next section). This statement relies on the belief [15] that computing r^{th} roots (mod M) for integral $r > 1$ is as difficult as factoring M . The step 4 in their scheme assigns all public information T 's in such a way that T_j/T_i is an integer if and only if the policy allows C_i to access C_j .

For example, the policy of a 5-system in Table 4 (the corresponding DG policy representation in Figure 6) can be enforced by assigning keys:

$$\begin{aligned} K_1 &= K_0 \bmod M \\ K_2 &= K_0^2 \bmod M \\ K_3 &= K_0^{2 \cdot 3 \cdot 7} \bmod M \\ K_4 &= K_0^{2 \cdot 3 \cdot 5} \bmod M \\ K_5 &= K_0^{2 \cdot 3 \cdot 5 \cdot 7} \bmod M. \end{aligned}$$

Note that there is no transitive or antisymmetric exception in this example. Their scheme can only enforce policies in the user hierarchy model.

6.2 Correctness

A correct key assignment scheme should prohibit all illegal key derivations. There are two kinds of illegal key derivations in the user hierarchy model:

1. A class C_i derives a class C_j 's key, but the policy forbids C_i from accessing C_j .
2. A set of classes $H \subset C$ collusively derive a class C_j 's key, but the policy does not allow any class in H to access C_j .

In Akl and Taylor's paper, they give a theorem (below) and several propositions (6.1, 6.4, 6.5, and 6.6) to show the correctness of their scheme. We add two propositions (6.2 and 6.3) here that make the proof of correctness more complete. (The notation $T_i \mid T_j$ below means that T_i is a factor of T_j and $T_i \nmid T_j$ otherwise).

Theorem 6.1 *Let t and t_1, \dots, t_n be given integers and suppose there is a feasibly computable function G for which*

$$K^t = G(K^{t_1}, K^{t_2}, \dots, K^{t_n}) \pmod{M}$$

for every K in Z_M^X , the group of units mod M . Let

$$d = \gcd\{t_i\}, \quad e = \gcd\{t, d\}, \quad \text{and } r = d/e.$$

Then we can feasibly compute r -th roots in Z_M^X .

Proof:

Taking any H in Z_M^X , we will compute $H^{1/r} \pmod{M}$. Let $K = H^{1/d}$ (we cannot necessarily compute K) and $r_i = t_i/d$. Choose a and b so that $e = at + bd$. Then

$$\begin{aligned} H^{1/r} &= H^{e/d} = K^e = K^{bd} K^{at} = H^b G(K^{t_1}, \dots, K^{t_n})^a \\ &= H^b G(K^{dr_1}, \dots, K^{dr_n})^a \\ &= H^b G(H^{r_1}, \dots, H^{r_n})^a \pmod{M}, \end{aligned}$$

and this can feasibly be computed.

The argument relies on the current belief [15] that computing r^{th} roots \pmod{M} for integral $r > 1$ is as difficult as factoring M . The case $r = 2$ is proved in [14]. The method therein generalizes to the case $r \mid \Phi(M)$, where $\Phi(M)$ is the Euler totient function. In Akl and Taylor's paper, Proposition 6.1 only has the "only if" part. Because the value T_j/T_i can be very large, the function $f = K_i^{T_j/T_i} \pmod{M}$ is not feasible to compute. In order to make the "if" part of this proposition also true, we assume that there exists an upper

bound U for the value T_j/T_i so that f is feasible to compute.

Proposition 6.1 *The key derivation function $f(K_i, T_i, T_j)$ is feasible to compute if and only if $T_i \mid T_j$.*

Proof:

Let $G(K_i) = K_i^{T_j/T_i}$, then $f(K_i, T_i, T_j) \equiv G(K_i) \pmod{M}$

If $f(K_i, T_i, T_j)$ is feasible to compute, then K_j can be computed from $f(K_i, T_i, T_j)$, so as $G(K_i) \pmod{M}$.

That is, $K_j = K_0^{T_j} = G(K_0^{T_i}) \pmod{M}$ for every K_0 .

By the Theorem 6.1 above, $r = 1$ and hence $T_i \mid T_j$. Otherwise we could compute nontrivial roots in Z_M^X .

Proposition 6.2 *K_j can be derived from K_i if and only if $f(K_i, T_i, T_j) = (K_i)^{T_j/T_i} = K_j$ is feasible to compute.*

Proof:

If $f(K_i, T_i, T_j)$ is feasible to compute, then it is obvious that K_j can be derived from K_i .

Conversely, if $f(K_i, T_i, T_j) = (K_i)^{T_j/T_i} = K_j$ is infeasible to compute $\Rightarrow T_j/T_i$ is not an integer. (by Proposition 6.1)

Suppose that K_j can be derived from K_i by using other public information X and Y so that $f(K_i, X, Y) = K_j$ is feasible to compute $\Rightarrow Y/X$ is an integer. (by Proposition 6.1)

Since $f(K_i, X, Y) = ((K_0)^{T_i})^{Y/X} = K_j \Rightarrow T_i \cdot Y/X = T_j \Rightarrow Y/X = T_j/T_i$ - contradiction.

Thus K_j cannot be derived from K_i .

Proposition 6.3 *K_j can be derived from K_i if and only if $T_i \mid T_j$*

Proof:

By Propositions 6.1 and 6.2, the result follows.

Proposition 6.4 *Under the assignment scheme in Section 6.1, $T_i \mid T_j$ if and only if $A_{ij} = 1$.*

Proof:

If $A_{ij} = 1 \Rightarrow C_j \in AS_i$. For any $C_k \in AS_j$, we have $C_k \in AS_i$ (transitive property) $\Rightarrow AS_j \subseteq AS_i$.

Since $T_i = \prod_{A_{ik}=0} P_k = \prod_{C_k \notin AS_i} P_k$, $T_j = \prod_{A_{jk}=0} P_k = \prod_{C_k \notin AS_j} P_k$, and $AS_j \subseteq AS_i \Rightarrow T_i \mid T_j$.

Conversely if $A_{ij} = 0 \Rightarrow P_j \mid T_i$. Since $A_{jj} = 1$ (reflexive property) $\Rightarrow P_j \nmid T_j \Rightarrow T_i \nmid T_j$.

Proposition 6.5 *Under the assignment scheme in Section 6.1, a key K_j can be feasibly computed from a set of keys $\{K_i : C_i \in H\}$ if and only if $\gcd(T_i : C_i \in H) \mid T_j$, where $H \subset C$.*

Proof:

If $g = \gcd(T_i : C_i \in H)$, then we can choose integers a_i such that $g = \sum_H a_i T_i$. If $T_j = gm$ for some integer m , then $K_j = (K_0)^{T_j} = (K_0)^{gm} = \prod_H (K_0)^{T_i a_i m} = \prod_H (K_i)^{a_i m}$ and K_j can be computed from K_i 's.

Conversely, if there exists a feasibly computable function for obtaining $K_j = (K_0)^{T_j} \pmod{M}$ from the K_i 's for every K_0 . By the Theorem 6.1 above, $r = 1$ and $\gcd(T_i : C_i \in H) \mid T_j$.

Proposition 6.6 *Under the assignment scheme in Section 6.1, a set of classes $H \subset C$ can not collusively derive a class C_j 's key if $A_{ij} = 0$ for all $T_i \in H$.*

Proof:

$P_j \mid T_i$ whenever $A_{ij} = 0$, therefore, $P_j \mid \gcd(T_i : A_{ij} = 0)$.

But $A_{jj} = 1$ (reflexive property) $\Rightarrow (P_j \nmid T_j) \Rightarrow \gcd(T_i : A_{ij} = 0) \nmid T_j$.

By Proposition 6.5, the result follows.

From Propositions 6.3 and 6.4, K_j can be derived from K_i if and only if the policy allows C_i to access C_j . Thus, the first illegal key derivation is prohibited. From Propositions 6.5 and 6.6, the second illegal key derivation is also prohibited.

7 The New Key Assignment Scheme

In the new key assignment scheme, there are two keys (K^e and K^d) assigned to each class: K^e for data encryption and K^d for key derivation. If a class C_i would like to access the data of a class C_j , C_i uses its own derivation key K_i^d and some public information to derive C_j 's encryption key K_j^e . The purpose of assigning two keys for each class is to enforce transitive exception policies. For example, a transitive exception policy for a 3-system $\{C_1, C_2, C_3\}$ defines that C_1 can access C_2 and C_2 can access C_3 , but C_1 cannot access C_3 as in Figure 7. If only one key is assigned to each class, there is no way to enforce the transitive exception because C_1 can always derive C_3 's key by deriving C_2 's key first. With two keys for each class in our assignment scheme, this problem can be solved. The idea of enforcing transitive exceptions is shown in Figure 7 (here, an arrow in the key derivation rule means the source key can derive the destination key). Furthermore, two keys for each class also makes it possible to enforce antisymmetric exceptions. Therefore, two classes can access each other's information without being equivalent. Figure 8 shows the key derivation rule for enforcing antisymmetric exceptions. The key assignment scheme proposed in next section assigns keys in a way that follows the key derivation rules so that the transitive and antisymmetric exceptions can be enforced.

7.1 Key Assignment and Derivation

The access control policy in a matrix A , described in Section 2, only indicates which class can access another. Transitive exception information is embedded in A . For any pair of classes C_i and C_j , two kinds of exception information are important to our assignment scheme. They are

1. Whether C_i has a transitive exception to C_j , and
2. If there is a transitive exception, which other classes are intermediate for this exception.

In order to have the above information, two algorithms are applied to the matrix A . Algorithm 7.1 translates an $n \times n$ matrix A to another $n \times n$ matrix A' which explicitly indicates whether there are transitive exceptions. Then, the matrix A' is fed into Algorithm 7.2 to generate another $n \times n$ matrix B to further include the second exception information. Before presenting these two algorithms, the definitions of matrix A' and B are given below.

$$A'_{ij} = \begin{cases} 1 & \text{if } A_{ij} = 1 \\ -1 & \text{if } \exists 1 \leq k_1, k_2, \dots, k_m \leq n, \text{ where } 1 \leq m \leq n-2, \\ & \text{such that } A_{ik_1} = A_{k_1k_2} = \dots = A_{k_mj} = 1 \text{ and } A_{ij} = 0 \\ & \text{(transitive exception from class } C_i \text{ to class } C_j) \\ 0 & \text{otherwise} \end{cases}$$

$$B_{ij} = \begin{cases} 2 & \text{if } A'_{ij} = 1 \text{ and } \exists k \neq i, j \text{ such that } A'_{jk} = 1, \text{ but } A'_{ik} = -1 \\ & \text{(class } C_j \text{ is an intermediate class for a transitive exception)} \\ A'_{ij} & \text{otherwise} \end{cases}$$

Intuitively, a class C_i can access another class C_j if $B_{ij} = 1$ or 2 in which the latter case indicates that C_j is an intermediate class for transitive exceptions from C_i to some other classes. On the other hand, C_i can not access C_j if $B_{ij} = -1$ or 0 in which the former case indicates that there is a transitive exception from C_i to C_j .

Algorithm 7.1 computes the transitive closure A^* of A and compares A^* and A to find out the transitive exceptions.

Algorithm 7.1 *Transformation from A to A'*

$$A' = A - (A^* - A) = 2A - A^*.$$

Algorithm 7.2 uses a nested loop to find out which classes are intermediate for transitive exceptions.

Algorithm 7.2 *Transformation from A to B*

```

for  $i \leftarrow 1$  to  $n$ 
  for  $j \leftarrow 1$  to  $n$ 
    {
       $B_{ij} = A_{ij}$ 
      if  $A_{ij} = 1$ 
        for  $k \leftarrow 1$  to  $n$ 
          if  $A_{jk} = 1$  and  $A_{ik} = -1$ 
            {
               $B_{ij} = 2$ 
              break
            }
    }
  }

```

We demonstrate the transformation of algorithms 7.1 and 7.2 with the distributed static database example in Table 3. Table 5 is the resulting matrix B . With the transformed matrix B , the central authority C_0 proceeds as follows.

1. Generate a secret number K_0 .
2. Compute the product M of two large prime numbers and make it public.

For $i, j, k = 1, 2, \dots, n$

3. Assign a distinct prime number P_i to each row B_i of B and a number P_i' to each row B_i of B , where

$$P_i' = \begin{cases} \text{another distinct prime number} & \text{if } \exists k \text{ such that } B_{ki} = 2 \\ 1 & \text{otherwise} \end{cases}$$

Compute a product X_i for each row B_i of B , where

$$X_i = \begin{cases} (\prod_{B_{ij}=1} P_j) \cdot (\prod_{B_{ij}=2} P_j') & \text{if } \exists k \text{ such that } B_{ki} = 2 \\ 1 & \text{otherwise} \end{cases}$$

4. Compute the public information

$$T_i^d = \left(\prod_{B_{ij} \neq 1} P_j \right) \cdot \left(\prod_{B_{ij}=0 \text{ or } -1} P_j \right)$$

$$T_i^e = T_i^d \cdot X_i$$

5. Assign keys

$$K_i^d = (K_0)^{T_i^d} \bmod M$$

$$K_i^e = (K_0)^{T_i^e} \bmod M$$

to each class C_i and send them securely.

The class C_i can use the key derivation function

$$f(K_i^d, T_i^d, T_j^e) = (K_i^d)^{T_j^e/T_i^d} \bmod M$$

to derive the encryption key K_j^e of C_j , where

$$\begin{aligned} K_j^e &= (K_0)^{T_j^e} \bmod M \\ &= ((K_0)^{T_i^d})^{T_j^e/T_i^d} \bmod M \\ &= (K_i^d)^{T_j^e/T_i^d} \bmod M \end{aligned}$$

The way to assign public information T 's in step 4 ensures that T_j^e/T_i^d is an integer if and only if the policy allows C_i to access C_j . Since the key derivation function $f(K_i^d, T_i^d, T_j^e)$ is feasible to compute if and only if T_j^e/T_i^d is an integer (Proposition 6.1). Therefore, C_i can access C_j 's data by deriving C_j 's encryption key using the key derivation function $f(K_i^d, T_i^d, T_j^e)$. The correctness of the assignment scheme and the derivation function is formally proved in next section. Our scheme is secure against collusive attacks using the Euclidean algorithm. The Euclidean collusive attack is launched by a set H of classes $\{C_{i_1}, C_{i_2}, \dots, C_{i_j}\}$ to collusively apply Euclidean algorithm to derive a key which is not supposed to be known to all classes in H . This kind of collusive attacks is described as follows. Let $Y = \gcd\{T_{i_k} \mid k = 1, 2, \dots, j\}$, then $\gcd\{T_{i_k}/Y \mid k = 1, 2, \dots, j\} = 1$. This means that all T_{i_k}/Y , for all $C_{i_k} \in H$, are relatively prime to each other. Therefore, all $C_{i_k} \in H$ can conspire to compute a value $K_0^Y \bmod M$ using the Euclidean Algorithm. The algorithm makes all $C_{i_k} \in H$ possible to find integers U_{i_k} such that $\sum_{k=1}^j U_{i_k} \cdot T_{i_k}/Y = 1$. All $C_{i_k} \in H$ raise the power U_{i_k} to their own key and multiply the results to

get the value $K_0^Y \bmod M$. That is,

$$(K_{i_1})^{U_{i_1}} = ((K_0)^{T_{i_1}})^{U_{i_1}} = (K_0)^{U_{i_1} \cdot T_{i_1}} \bmod M$$

$$(K_{i_2})^{U_{i_2}} = ((K_0)^{T_{i_2}})^{U_{i_2}} = (K_0)^{U_{i_2} \cdot T_{i_2}} \bmod M$$

.....

$$(K_{i_j})^{U_{i_j}} = ((K_0)^{T_{i_j}})^{U_{i_j}} = (K_0)^{U_{i_j} \cdot T_{i_j}} \bmod M$$

$$\prod_{k=1}^j (K_{i_k})^{U_{i_k}} = \prod_{k=1}^j ((K_0)^{T_{i_k}})^{U_{i_k}} = (K_0)^{Y \cdot \sum_{k=1}^j U_{i_k} \cdot T_{i_k} / Y} = K_0^Y \bmod M$$

Base on the value $K_0^Y \bmod M$, the classes in H can derive the key of another class $C_l \notin H$ if $Y \mid T_l$. Our scheme is secure against this attack because the way of assigning public information T 's in step 4 makes all T_l not multiple to Y . Lemma 7.9 in the next section shows this.

For the access control policy in Table 5, Figure 9 shows the corresponding key derivation rule and Table 6 illustrates a possible assignment of the prime numbers and public information. The key assignments using the new scheme are:

$$\begin{aligned} K_1^d &= K_0^{3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 19} \bmod M & K_1^e &= K_0^{3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 19} \bmod M \\ K_2^d &= K_0^{2 \cdot 7 \cdot 11 \cdot 13} \bmod M & K_2^e &= K_0^{2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 19} \bmod M \\ K_3^d &= K_0^{2 \cdot 3 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19} \bmod M & K_3^e &= K_0^{2 \cdot 3 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19} \bmod M \\ K_4^d &= K_0^{2 \cdot 3 \cdot 5 \cdot 11 \cdot 13 \cdot 17} \bmod M & K_4^e &= K_0^{2 \cdot 3 \cdot 5 \cdot 11 \cdot 13 \cdot 17} \bmod M \\ K_5^d &= K_0^{2 \cdot 3 \cdot 5 \cdot 7} \bmod M & K_5^e &= K_0^{2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17} \bmod M \\ K_6^d &= K_0^{2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 17 \cdot 19} \bmod M & K_6^e &= K_0^{2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 17 \cdot 19} \bmod M \end{aligned}$$

7.2 Correctness

There are three possible illegal key derivations in the user hierarchy-with-exception model:

1. A class C_a derives class C_b 's keys by using the derivation function f , but the policy does not allow C_a to access C_b .
2. A class C_a derives class C_c 's keys by iteratively applying derivation function f to derive another classes C_{b_1} 's, C_{b_2} 's, ..., C_{b_m} 's keys, where the

policy allows C_a to access C_{b_1} , C_{b_1} to access C_{b_2} , \dots , C_{b_m} to access C_c , but not C_a to access C_c .

3. A set of classes $H \subset C$ collusively derive class C_b 's keys by using the derivation function f , but the policy does not allow any class in H to access C_b .

We use indices $a, b, c \in \{1, 2, \dots, n\}$ along with $i, j, k \in \{1, 2, \dots, n\}$ (used in assignment definitions in Section 7.1) to avoid confusion. The above three illegal key derivations are infeasible under our key assignment scheme. Lemmas 7.7, 7.8 and 7.9 (below) prove the first, second, and third illegal key derivations respectively are infeasible.

First, we give a proposition showing the key derivation relationship between keys.

Proposition 7.1 *Under the key assignment scheme in Section 7.1, if a key K_a^x can derive a key K_b^x (for $x = d$ or e) by using the key derivation function $f(K_a^x, T_a^x, T_b^x) = K_b^x$, then (i) any key can be derived from K_b^x , it also can be derived from K_a^x ; (ii) any key can derive K_a^x , it also can derive K_b^x .*

Proof:

(i) Let a set of keys $R = \{K_c^x : K_c^x \text{ can be derived from } K_a^x\}$ and a set of keys $S = \{K_c^x : K_c^x \text{ can be derived from } K_b^x\}$.

By Proposition 6.3, we can rewrite $R = \{K_c^x : T_a^x \mid T_c^x\}$ and $S = \{K_c^x : T_b^x \mid T_c^x\}$.

Since K_a^x can derive K_b^x , then $T_a^x \mid T_b^x$ (by Proposition 6.3).

This implies $R \supseteq S$.

(ii) Let a set of keys $R = \{K_c^x : K_c^x \text{ can derive } K_a^x\}$ and a set of keys $S = \{K_c^x : K_c^x \text{ can derive } K_b^x\}$.

By Proposition 6.3, we can rewrite $R = \{K_c^x : T_c^x \mid T_a^x\}$ and $S = \{K_c^x : T_c^x \mid T_b^x\}$.

Since K_a^x can derive K_b^x , then $T_a^x \mid T_b^x$ (by Proposition 6.3).

This implies $R \subseteq S$.

In the key assignment scheme in Section 7.1, each class can use its own derivation key to derive its own encryption key. Because of this property, each class

only needs to memorize a derivation key, but one more key derivation would be necessary whenever it wants to access its own data. The Proposition 7.2 shows this property.

Proposition 7.2 *Under the key assignment scheme in Section 7.1, K_a^d can always derive K_a^e by using the derivation function $f(K_a^d, T_a^d, T_a^e) = K_a^e$.*

Proof:

By Proposition 6.3, we only need to show $T_a^d \mid T_a^e$ is always true.

$$T_a^e / T_a^d = \frac{T_a^d \cdot X_i}{T_a^d} = X_i$$

Since X_i is either 1 or a product of primes.

Therefore, T_a^e / T_a^d is always an integer and we have the result.

Proposition 7.3 *Under the key assignment scheme in Section 7.1, (i) if a key can be derived from K_a^e , it also can be derived from K_a^d ; (ii) if a key can derive K_a^d , it also can derive K_a^e .*

Proof:

By Propositions 7.1 and 7.2, the result follows.

Proposition 7.4 *If $B_{ab} = 1$, then $B_{aj} \neq 1$ implies $B_{bj} \neq 1$, and also $B_{aj} = 0$ or -1 implies $B_{bj} = 0$ or -1 .*

Proof:

Part 1: Prove if $B_{ab} = 1$, then $B_{aj} \neq 1$ implies $B_{bj} \neq 1$.

The above statement is equivalent to the following statement.

“If $B_{ab} = 1$, then $B_{bj} = 1$ implies $B_{aj} = 1$ ”.

The proof of the equivalent statement is as follows.

If $B_{ab} = 1$ and $B_{bj} = 1$,

$\implies A_{ab} = 1$ in which C_b is not an intermediate class for any exception from C_a , and $A_{bj} = 1$ in which C_j is not an intermediate class for any exception from C_b ,

$\implies A_{aj} = 1$ in which C_j is not an intermediate class for any exception from C_a ,

$\implies B_{aj} = 1$.

Part 2: Prove if $B_{ab} = 1$, then $B_{aj} = 0$ or -1 implies $B_{bj} = 0$ or -1 .

The above statement is equivalent to the following statement.

“If $B_{ab} = 1$, then $B_{bj} = 1$ or 2 implies $B_{aj} = 1$ or 2 ”.

The proof of the equivalent statement is as follows.

If $B_{ab} = 1$ and $B_{bj} = 1$ or 2 ,

$\implies A_{ab} = 1$ and $A_{bj} = 1$,

$\implies A_{aj} = 1$,

$\implies B_{aj} = 1$ or 2 .

Proposition 7.5 *If $B_{ab} = 2$, then $B_{aj} = 0$ or -1 implies $B_{bj} \neq 1$.*

Proof:

The above statement is equivalent to the following statement.

“If $B_{ab} = 2$, then $B_{bj} = 1$ implies $B_{aj} = 1$ or 2 ”.

The proof of the equivalent statement is as follows.

If $B_{ab} = 2$ and $B_{bj} = 1$,

$\implies A_{ab} = 1$ and $A_{bj} = 1$,

$\implies A_{aj} = 1$,

$\implies B_{aj} = 1$ or 2 .

Proposition 7.6 *Under the key assignment scheme in Section 7.1, $T_a^d \mid T_b^e$ if and only if $B_{ab} = 1$ or 2 .*

Proof:

By Proposition 7.4, if $B_{ab} = 1$,

then $(B_{aj} \neq 1 \Rightarrow B_{bj} \neq 0)$ and $(B_{aj} = 0$ or $-1 \Rightarrow B_{bj} = 0$ or $-1)$.

Thus, let $I = \frac{\prod_{B_{bj} \neq 1} P_j}{\prod_{B_{aj} \neq 1} P_j}$ and $II = \frac{\prod_{B_{bj}=0 \text{ or } -1} P_j^l}{\prod_{B_{aj}=0 \text{ or } -1} P_j^l}$, then both I and II should be positive integers.

$$\begin{aligned} T_b^e / T_a^d &= \frac{(\prod_{B_{bj} \neq 1} P_j) \cdot (\prod_{B_{bj}=0 \text{ or } -1} P_j^l) \cdot X_b}{(\prod_{B_{aj} \neq 1} P_j) \cdot (\prod_{B_{aj}=0 \text{ or } -1} P_j^l)} \\ &= I \cdot II \cdot X_b \end{aligned}$$

I , II and X_b are all positive integers.

Thus, $T_a^d \mid T_b^e$ if $B_{ab} = 1$.

By Proposition 7.5, if $B_{ab} = 2$,

then $(B_{aj} = 0 \text{ or } -1 \Rightarrow B_{bj} \neq 1)$.

Thus, let $III = \frac{\prod_{B_{bj} \neq 1} P_j^t}{\prod_{B_{aj}=0 \text{ or } -1} P_j^t}$, then III should be a positive integer.

Let $IV = \frac{\prod_{\forall j} P_j}{\prod_{B_{aj} \neq 1} P_j}$, then IV should also be a positive integer.

Since $B_{ab} = 2$, then $X_b = (\prod_{B_{bj}=1} P_j) \cdot (\prod_{B_{bj}=2} P_j^t)$.

$$\begin{aligned} T_b^e / T_a^d &= \frac{(\prod_{B_{bj} \neq 1} P_j) \cdot (\prod_{B_{bj}=0 \text{ or } -1} P_j^t) \cdot (\prod_{B_{bj}=1} P_j) \cdot (\prod_{B_{bj}=2} P_j^t)}{(\prod_{B_{aj} \neq 1} P_j) \cdot (\prod_{B_{aj}=0 \text{ or } -1} P_j^t)} \\ &= \frac{(\prod_{\forall j} P_j) \cdot (\prod_{B_{bj} \neq 1} P_j^t)}{(\prod_{B_{aj} \neq 1} P_j) \cdot (\prod_{B_{aj}=0 \text{ or } -1} P_j^t)} \\ &= III \cdot IV \end{aligned}$$

III and IV are both positive integers.

Thus, $T_a^d \mid T_b^e$ if $B_{ab} = 2$.

Conversely, if $B_{ab} \neq 1$ and $B_{ab} \neq 2 \Rightarrow B_{ab} = 0 \text{ or } -1$.

Since $T_a^d = (\prod_{B_{aj} \neq 1} P_j) \cdot (\prod_{B_{aj}=0 \text{ or } -1} P_j^t)$,

then $P_b \mid T_a^d$ and $P_b^t \mid T_a^d$.

Two cases are necessary to show $T_a^d \nmid T_b^e$ as follows.

Case 1: if $\exists k$ such that $B_{kb} = 2$,

then $P_b^t \neq 1$ and $T_b^e = (\prod_{B_{bj} \neq 1} P_j) \cdot (\prod_{B_{bj}=0 \text{ or } -1} P_j^t) \cdot (\prod_{B_{bj}=1} P_j) \cdot (\prod_{B_{bj}=2} P_j^t)$.

Since $B_{bb} = 1$, we have $P_b^t \nmid T_b^e$.

This implies $T_a^d \nmid T_b^e$.

Case 2: if $\nexists k$ such that $B_{kb} = 2$,

then $T_b^e = (\prod_{B_{bj} \neq 1} P_j) \cdot (\prod_{B_{bj}=0 \text{ or } -1} P_j^t)$.

Since $B_{bb} = 1$, we have $P_b \nmid T_b^e$, where $P_b \neq 1$ (by definition).

This implies $T_a^d \nmid T_b^e$.

Lemma 7.7 *If the policy does not allow C_a to access C_b , then it is infeasible for C_a to derive C_b 's keys by using the derivation function f under the key assignment scheme in Section 7.1.*

Proof:

The available information for C_a are K_a^d , K_a^e , and all public information T_i^d , and T_i^e for all i .

By Proposition 6.2, only four possible inputs (K_a^d, T_a^d, T_b^d) , (K_a^d, T_a^d, T_b^e) , (K_a^e, T_a^e, T_b^d) , and (K_a^e, T_a^e, T_b^e) to function f need to be considered.

By Proposition 7.3, we only need to show one input (K_a^d, T_a^d, T_b^e) to f is not feasible to compute.

Since C_a is not allowed to access $C_b \Rightarrow T_a^d \not\perp T_b^e$ (by Proposition 7.6) $\Rightarrow f(K_a^d, T_a^d, T_b^e)$ is not feasible to compute. (by Proposition 6.1)

If an access control policy of a system contains transitive exceptions, the possibility of the second illegal key derivation becomes a threat. Fortunately, our key assignment scheme is immune from this threat.

Lemma 7.8 *If the policy does not allow C_a to access C_c , but C_a can access C_{b_1} , C_{b_1} can access C_{b_2} , \dots , C_{b_m} can access C_c for $1 \leq m \leq n - 2$, then it is infeasible for C_a to iteratively apply the derivation function f to derive C_c 's keys by deriving C_{b_1} 's, C_{b_2} 's, \dots , C_{b_m} 's keys under the key assignment scheme in Section 7.1.*

Proof:

By Lemma 7.7, if the policy does not allow C_a to access C_c , then K_a^x can not derive K_c^x , where $X = d$ or e .

So, basically we want to show that there does not exist any key derivation chain from K_a^x to K_c^x if C_a can not access C_c .

Suppose there exists a key derivation chain $K_a^x \rightarrow K_{b_1}^x \rightarrow K_{b_2}^x \rightarrow \dots \rightarrow K_{b_m}^x \rightarrow K_c^x$.

Since K_a^x can derive $K_{b_1}^x$ and $K_{b_1}^x$ can derive $K_{b_2}^x \Rightarrow K_a^x$ can derive $K_{b_2}^x$ (by Proposition 7.1)

By applying Proposition 7.1 m times, we have the result that K_a^x can derive K_c^x . It is a contradiction.

Thus, there does not exist any key derivation chain from K_a^x to K_c^x if the policy does not allow C_a to access C_c .

Finally, lemma 7.9 shows our key assignment scheme is infeasible for the collusive attack. That is, a set of malicious classes combines their information to derive a key that the policy does not allow them to derive.

Lemma 7.9 *If the policy does not allow any class in a set of classes $H \subset C$ to access C_b , then it is infeasible for classes in H to collusively derive C_b 's keys by using the derivation function f under the key assignment scheme in*

Section 7.1.

Proof:

By Proposition 7.3, we only need to show a set of derivation keys $\{K_a^d : C_a \in H\}$ can not collusively derive C_b 's encryption key K_b^e , where $B_{ab} = 0$ or -1 .

Two cases are necessary for the proof.

Case 1: if $\exists k$ such that $B_{kb} = 2$, then $P_b \nmid 1$ and $T_a^d = (\prod_{B_{aj} \neq 1} P_j) \cdot (\prod_{B_{aj}=0 \text{ or } -1} P_j)$ and $T_b^e = (\prod_{B_{bj} \neq 1} P_j) \cdot (\prod_{B_{bj}=0 \text{ or } -1} P_j) \cdot (\prod_{B_{bj}=1} P_j) \cdot (\prod_{B_{bj}=2} P_j)$.

If $B_{ab} = 0$ or -1 , then $P_b \nmid T_a^d$.

Therefore, $P_b \nmid \gcd(T_a^d : B_{ab} = 0 \text{ or } -1)$.

Since $B_{bb} = 1$, so $P_b \nmid T_b^e$.

Thus, $\gcd(T_a^d : B_{ab} = 0 \text{ or } -1) \nmid T_b^e$.

Case 2: if $\nexists k$ such that $B_{kb} = 2$, then $T_a^d = (\prod_{B_{aj} \neq 1} P_j) \cdot (\prod_{B_{aj}=0 \text{ or } -1} P_j)$ and $T_b^e = (\prod_{B_{bj} \neq 1} P_j) \cdot (\prod_{B_{bj}=0 \text{ or } -1} P_j)$.

If $B_{ab} = 0$ or -1 , then $P_b \mid T_a^d$, where $P_b \neq 1$ (by definition).

Therefore, $P_b \mid \gcd(T_a^d : B_{ab} = 0 \text{ or } -1)$.

Since $B_{bb} = 1$, so $P_b \nmid T_b^e$.

Thus, $\gcd(T_a^d : B_{ab} = 0 \text{ or } -1) \nmid T_b^e$.

By case 1 and case 2 and by Proposition 6.5, we have the result.

We have proved correctness of the proposed key assignment scheme by showing that the three possible illegal key derivations, listed at the beginning of the section, are not feasible.

8 Conclusion

This paper gives a cryptographic key assignment scheme for controlling access to data in an organization where the read/write access control policies for this organization can be any policy as a user hierarchy-with-exception. The user hierarchy-with-exception model can handle not only the access control policies in the user hierarchy model but also more complex policies with antisymmetric or transitive exceptions. Transitive exceptions usually occur in a system that is not hierarchy-based. Both antisymmetric and transitive exceptions are quite common for a distributed system as the example shown in Section 4. The cost

of this new cryptographic key assignment scheme, compared to the Akl and Taylor's scheme, is one more key for each class to memorize or one more step to access its own data.

There are two keys (derivation and encryption keys) assigned to each class for each different access right in our cryptographic key assignment scheme. The read encryption key of a class encrypts its own data to protect against illegal read accesses from other classes. The read derivation key of a class could derive the read encryption keys of other classes in order to decrypt and access their data. On the other hand, the write encryption key of a class could be derived by other classes to encrypt the data being written to the class. The write derivation key of a class enables the class to verify write requests and decrypt the data to be written from other classes. The key assignment scheme assures that a class's derivation key cannot derive another class's encryption key in violation of the policy. We have formally shown the correctness of the key assignment scheme.

Acknowledgments

The authors wish to thank many colleagues for their suggestions. Part of this research was supported by the National Security Agency under contract number: MDA 904-98-C-A892.

References

- [1] S. G. Akl, P. D. Taylor, Cryptographic Solution to a Problem of Access Control in a Hierarchy, *ACM Transactions on Computer Systems* 1(1983) 239–248.
- [2] S. J. Mackinnon, P. D. Taylor, H. Meijer, S. G. Akl, An Optimal Algorithm for Assigning Cryptographic Keys to Control Access in a Hierarchy, *IEEE Transactions on Computers* c-34(1985) 797–802.
- [3] R. S. Sandhu, Cryptographic Implementation of a Tree Hierarchy for Access Control, *Information Processing Letters* 27(1988) 95–98.
- [4] L. Harn, H.-Y. Lin, A Cryptographic Key Generation Scheme for Multilevel Data Security, *Computers and Security* 9(1990) 539–546.

- [5] L. Harn, Y.-R. Chien, T. Kiesler, An Extended Cryptographic Key Generation Scheme for Multilevel Data Security, Fifth Annual Computer Security Application Conference, 1989, pp. 254–262.
- [6] B.-M. Shao, J.-J. Hwang, P. Wang, Distributed Assignment of Cryptographic Keys for Access Control in a Hierarchy, *Computers and Security* 13(1994) 79–84.
- [7] H.-T. Liaw, A Dynamic Cryptographic Key Generation and Information Broadcasting Scheme in Information Systems, *Computers and Security* 13(1994) 601–610.
- [8] M.-S. Hwang, W.-P. Yang, A New Dynamic Cryptographic Key Generation Scheme for a Hierarchy, 1994 IEEE Region 10's Ninth Annual International Conference, 1994, pp. 465–468.
- [9] T.-C. Wu, T.-S. Wu, W.-H. He, Dynamic Access Control Scheme Based on the Chinese Remainder Theorem, *Computer Systems Science and Engineering* 10(1995) 92–99.
- [10] H.-M. Tsai, C.-C. Chang, A Cryptographic Implementation for Dynamic Access Control in a User Hierarchy, *Computers and Security* 14(1995) 159–166.
- [11] S.-J. Wang, J.-F. Chang, A Hierarchical and Dynamic Group-Oriented Cryptographic Scheme, *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences* E79-A(1996) 76–85.
- [12] F.-K. Tu, C.-S. Lai, W.-C. Kuo, Cryptanalysis of a New Cryptographic Solution for Dynamic Access Control in a Hierarchy, 1997 Information Security Conference (INFOSEC '97), 1997, pp. 104–108.
- [13] Y.-M. Tseng, J.-K. Jan, A Scheme for Authorization Inheritance in a User Hierarchy, 1997 Information Security Conference (INFOSEC '97), 1997, pp. 109–115.
- [14] M. O. Rabin, Digitalized Signatures and Public-key Functions as Intractable as Factorization, Technical Report MIT/LCS/TR-212, Laboratory for Computer science, Massachusetts Institute of Technology, Cambridge, Mass., 1979.
- [15] R. L. Rivest, A. Shamir, L. Adleman, A Method for Obtaining Digital Signatures and Public-key Cryptosystems, *Communications ACM* 21(1978) 120–126.
- [16] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman, Protection in Operating Systems, *Communications ACM* 19(1976) 461-471.
- [17] D. Bell and L. La Padula, Secure Computer Systems: Mathematical Foundations and Model, MITRE Report, MTR2547, 2(1973).

Table 1
Matrix A – an access policy in a 3-system

	C_1	C_2	C_3
C_1	1	1	1
C_2	0	1	1
C_3	1	0	1

Table 2
Confidential records for EMPLOYEE

Name	Rank	Sex	Age	Salary
Joe	1	M	50	60,000
Sam	2	M	45	55,000
Mary	1	F	46	62,000
John	3	M	36	50,000

Table 3

Matrix A – the access policy for a two-site distributed static database system

	C_1	C_2	C_3	C_4	C_5	C_6
C_1	1	1	0	0	0	0
C_2	0	1	1	0	1	0
C_3	0	0	1	0	0	0
C_4	0	0	0	1	1	0
C_5	0	1	0	0	1	1
C_6	0	0	0	0	0	1

Table 4
 Akl and Taylor's scheme for an access policy in a 5-system

	C_1	C_2	C_3	C_4	C_5	P_i	T_i
C_1	1	1	1	1	1	2	1
C_2	0	1	1	1	1	3	2
C_3	0	0	1	0	1	5	$2 \cdot 3 \cdot 7$
C_4	0	0	0	1	1	7	$2 \cdot 3 \cdot 5$
C_5	0	0	0	0	1	11	$2 \cdot 3 \cdot 5 \cdot 7$

Table 5

Matrix B – an access policy in a 6-system with explicit transitive exception information

	C_1	C_2	C_3	C_4	C_5	C_6
C_1	1	2	-1	0	-1	-1
C_2	0	1	1	0	2	-1
C_3	0	0	1	0	0	0
C_4	0	-1	-1	1	2	-1
C_5	0	2	-1	0	1	1
C_6	0	0	0	0	0	1

Table 6
 Prime numbers and public information for the policy in Table 5

	P_i	P_i'	X_i	T_i^d	T_i^e
C_1	$P_1 = 2$	$P_1' = 1$	1	$3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 19$	$3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 19$
C_2	$P_2 = 3$	$P_2' = 17$	$3 \cdot 5 \cdot 19$	$2 \cdot 7 \cdot 11 \cdot 13$	$2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 19$
C_3	$P_3 = 5$	$P_3' = 1$	1	$2 \cdot 3 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$	$2 \cdot 3 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$
C_4	$P_4 = 7$	$P_4' = 1$	1	$2 \cdot 3 \cdot 5 \cdot 11 \cdot 13 \cdot 17$	$2 \cdot 3 \cdot 5 \cdot 11 \cdot 13 \cdot 17$
C_5	$P_5 = 11$	$P_5' = 19$	$11 \cdot 13 \cdot 17$	$2 \cdot 3 \cdot 5 \cdot 7$	$2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17$
C_6	$P_6 = 13$	$P_6' = 1$	1	$2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 17 \cdot 19$	$2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 17 \cdot 19$

class	accessible set	dominating set
C_1	C_1, C_2	C_1
C_2	C_2, C_3	C_1, C_2
C_3	C_3	C_2, C_3

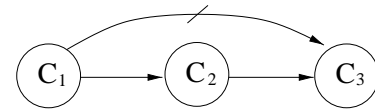


Fig. 1. An access policy in a 3-system that only violates the transitive property

class	accessible set	dominating set
C_1	C_1, C_2, C_3	C_1, C_3
C_2	C_2, C_3	C_1, C_2
C_3	C_1, C_3	C_1, C_2, C_3

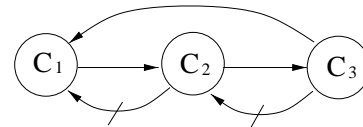


Fig. 2. An access policy in a 3-system that violates both antisymmetric and transitive properties

class	accessible set	dominating set
C_1	C_1, C_2	C_1, C_2
C_2	C_1, C_2, C_3	C_1, C_2
C_3	C_3	C_2, C_3

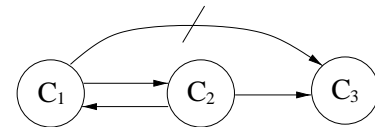


Fig. 3. A transitive exception makes $AS_1 \neq AS_2$ but $DS_1 = DS_2$

class	accessible set	dominating set
C_1	C_1, C_2	C_1, C_2
C_2	C_1, C_2	C_1, C_2, C_3
C_3	C_2, C_3	C_3

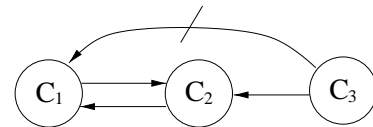


Fig. 4. A transitive exception makes $DS_1 \neq DS_2$ but $AS_1 = AS_2$

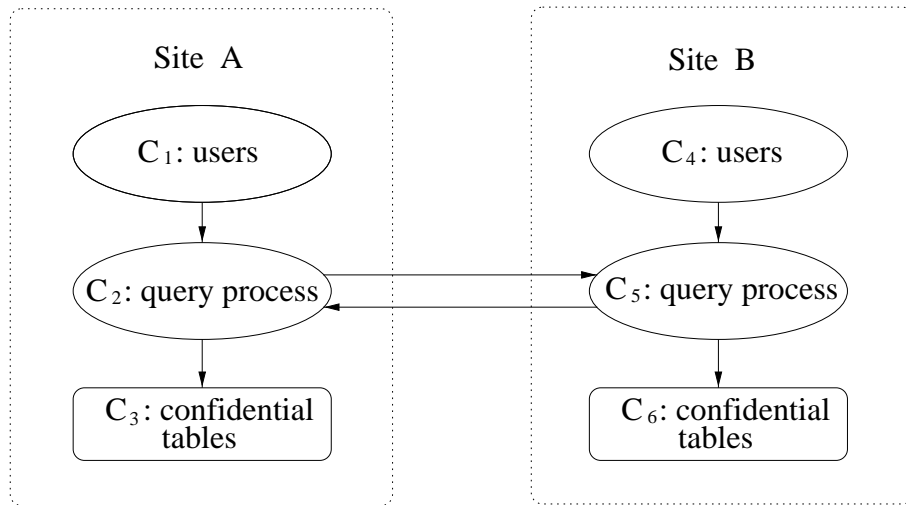


Fig. 5. A two-site distributed static database system

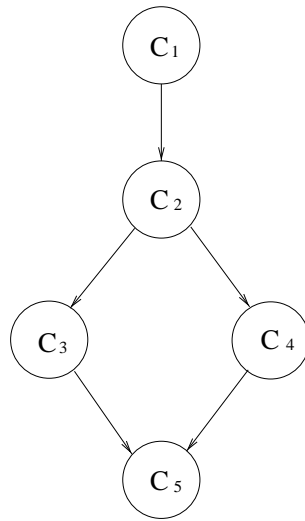


Fig. 6. DG representation of an access policy in a 5-system

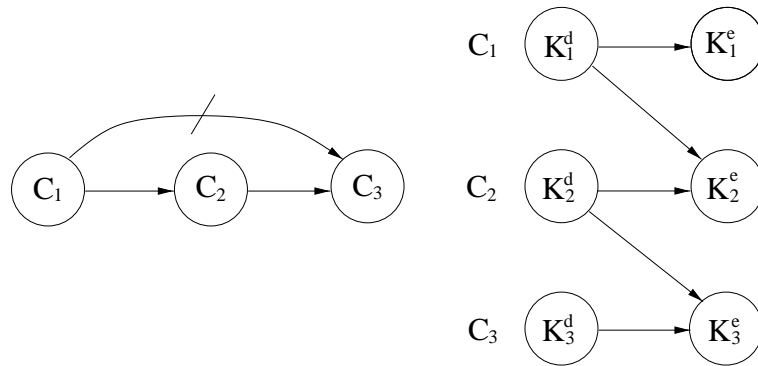


Fig. 7. A 3-system with a transitive exception and the corresponding key derivation rule

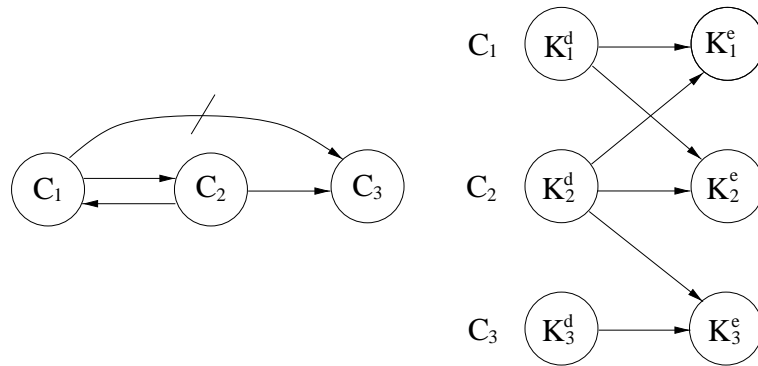


Fig. 8. A 3-system with an antisymmetric exception and the corresponding key derivation rule

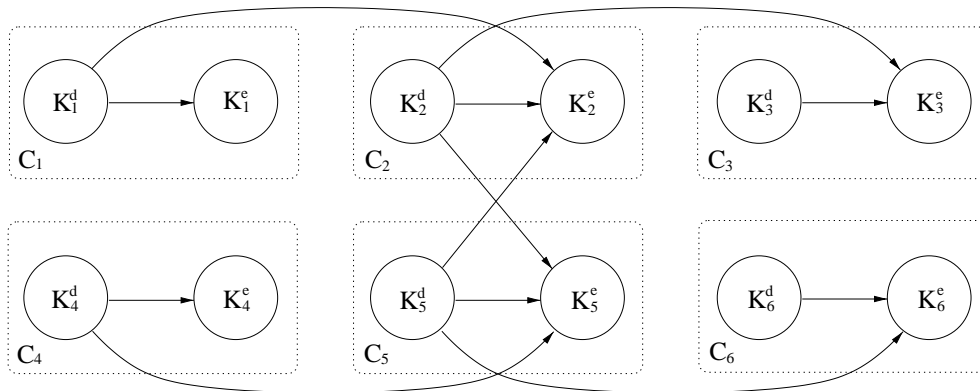


Fig. 9. The key derivation rule for the policy in Table 5