An Efficient Key Assignment Scheme for Access Control in a Large Leaf Class Hierarchy

Jung-Wen Lo† Min-Shiang Hwang‡ Chia-Hsin Liu§

Department of Management Information Systems‡
National Chung Hsing University
250 Kuo Kuang Road, Taichung, Taiwan 402, R.O.C.
Email: mshwang@nchu.edu.tw
Fax: 886-4-22857173

Department of Information Management†
National Taichung Institute of Technology
129 Sec. 3, San-min Rd., Taichung, Taiwan 404, R.O.C

Department of Computer Science & Information Engineering§
Asia University
500, Lioufeng Rd., Wufeng, Taichung 41354, Taiwan, R.O.C.

October 25, 2012

Abstract

The employees of an organization are usually divided into different security classes to authorize the information retrieval, and the number of leaf classes is substantially larger than the number of non-leaf classes. Additionally, the alternations in leaf classes are more frequent than in non-leaf classes. We proposed a new key assignment scheme for controlling the access right in a large POSET (partially ordered set) hierarchy to reduce the required computation for key generation and

†Responsible for correspondence: Prof. Min-Shiang Hwang.
derivation with the storage amount of data decreased.

*Keywords:* Access control, cryptography, data security, hierarchy, key management.

# 1 Introduction

The access control of resources in a social organization is getting more and more attention in recent years. In most social organizations, such as companies, governments and militaries, the employees are divided into different security classes according to their positions and access authorities [3, 6, 7, 14, 15, 17, 19, 22, 29, 30]. The access control privilege is assigned into different privileged classes as a user tree-structure. In the structure of privileged classes, the users in the higher security classes can access the data in the lower security classes but not vice versa. Usually, the number and the frequency of alternation of bottom classes (leaf nodes) are much larger than the number of non-bottom classes (non-leaf nodes) in a user tree-structure. We need a very efficient scheme to handle the access control problem for this type of user structure.

Under such a user hierarchy, the entities including people, work and other items are organized into a number of disjointed sets of security classes [1]. Assigning users to a security class is called “security clearance”. Let \( C_1, C_2, \ldots, C_n \) be \( n \) disjointed security classes according to the significant levels and \( C \) is a partially ordered set (POSET) under a binary relation denoted by “\( \leq \)” where \( C = \{C_1, C_2, \ldots, C_n\} \). Note that \( C_i \leq C_j \) means that the privilege in \( C_j \) is equal to or higher than the one in \( C_i \) where \( C_i \in C \) and \( C_j \in C \). Figure 1 shows a typical case of a POSET in a user hierarchy. For example, \( C_2 \leq C_1 \) means that \( C_1 \) is a predecessor of \( C_2 \) and \( C_2 \) is a successor of \( C_1 \). If there is no \( C_k \) satisfying the relationship \( C_2 \leq C_k \leq C_1 \), then \( C_1 \) is called an immediate predecessor of \( C_2 \) and \( C_2 \) is an immediate successor of \( C_1 \).
1.1 Relative Works

Akl-Taylor first proposed a scheme for access control with a POSET hierarchy in 1983 [1]. The central authority (CA) assigns every security class $C_i$ a public parameter $PB_i$ and a secret key $K_i$. $C_i$ can encrypt any plaintext with $K_i$ by symmetric cryptosystem [12, 13], and $C_j$ can retrieve the plaintext by the secret key which can be calculated from $PB_i$, $PB_j$ and $K_j$ when $C_i \leq C_j$.

In the Akl-Taylor scheme, the access control problem is easily solved by a POSET hierarchy, but their scheme has some drawbacks. A large amount of storage is required to keep the public parameters when the user hierarchy is large, and all the keys in the system have to be redistributed when a new security class is added. In 1985, Mackinnon et al. proposed an improved scheme called “canonical assignment scheme”, which reduced the number of distinct primes in a user hierarchy [18]. Although this scheme can reduce the number of distinct primes, it still needs a large amount of storage and an optimal algorithm, which is not easy to find, for a complex or arbitrary user hierarchy.

Sandhu proposed a cryptographic implementation of a tree hierarchy in 1988 [20]. This method uses a one-way function and children’s identities to generate the secret keys without the extra public parameters. Unfortunately, Sandhu’s scheme could not provide a solution for the general case in a POSET hierarchy.

Harn and Lin proposed a new key assignment scheme with a bottom-up
key generation scheme in 1990 [8]. The security of Harn-Lin’s scheme is based on the complexity of factoring out a big number. With this bottom-up key generation method, the storage space of public parameters is noticeably reduced. However, Harn-Lin’s scheme still requires a large amount of storage space for a large user hierarchy [9].

In 1998, Yeh et al. proposed a key assignment scheme with a matrix model which is called “YCN scheme” [26]. By employing a derivation key and an encrypted data key, their scheme can prevent the illegal access and allow only the authorized classes to derive the encrypted key. However, Hwang pointed out that several classes can easily collaborate to derive the two keys in a special case [10, 16]. Later, in 2003, Yeh et al. proposed an improved scheme to fix the collusive attack problem [27].

Tzeng proposed a time-bound cryptographic key assignment scheme in a partially ordered hierarchy in 2002 [23]. The cryptographic keys of security classes are different in distinct periods. Each user can use the legal key during the authorized period only. When the time has expired, the user in the security class can not access any data of the descendent classes. However, Yi and Ye showed that Tzeng’s scheme is insecure because any three users can collaborate to obtain the secret keys of some classes [28].

Hwang and Yang proposed an efficient access control scheme for those large partially ordered hierarchies in 2003 [11]. Hwang-Yang’s scheme is based on both Akl-Taylor’s and Harn-Lin’s schemes with the mathematical concepts of combination to reduce the number of selected prime numbers and to support a dynamic POSET hierarchy. However, the leaf group is difficult to form in a general application and Hwang-Yang’s scheme is insecure against the collusion attack [24].

Yang and Li proposed an efficient access control using one-way hash functions in 2004 [25]. When a class is eliminated from the hierarchy, the order of
the sibling would become a problem. Chen and Huang proposed a very efficient key management scheme for dynamic access control in 2005 [4]. Atallah et al. also proposed a pretty efficient access control scheme in 2005, but their scheme is not suitable for a deep tree or a tree with complex relationships [2]. Santis et al. proposed a time-bound and hierarchical key assignment scheme in 2006 [21] and Chung et al. proposed an access control scheme based on elliptic curve cryptosystem in 2008 [5]. However, all of the above schemes are not suitable for a large leaf class hierarchy.

1.2 Requirement of a Large Leaf Class Hierarchy

Most studies are designed for a general tree structure instead of a large leaf tree structure. The solution scheme should possess the following requirements.

1. The smallest space for storing the public parameters.

2. The guarantee of computation security.

3. The better efficiency of alternation occurring in lower-level classes.

4. Minimum influence on the whole structure when adding or deleting a class.

The number of prime numbers directly affects the space of the storage and causes the problem of computing load. Therefore, we have carefully considered that the amount of prime numbers used in the system should be reduced and proposed an efficient scheme for the dynamic key management.

1.3 Organization

This article is constructed as follows. The introduction is stated in Section 1. In Section 2, the efficient key assignment scheme for access control is proposed. The security and comparisons among schemes are discussed in Section 3. The last section is our conclusion.
2 The Proposed Scheme

Basically, the number of the selected prime numbers in the system affects the storage space of data, the key generation time and key derivation time. Our scheme has a relatively small amount of the prime numbers.

2.1 Key Generation Phase

In the key generation phase, there is a CA in the system, and the security classes have the authorized relationships shown in Figure 2. The CA executes the following steps:

Step 1. CA chooses two large primes $p$ and $q$, and then computes the public parameter $m = p \cdot q$, where $p$ and $q$ must be kept in secret.

Step 2. CA generates a random number $K_0$ where the $K_0$ and $m$ are relative primes, and the condition $2 < K_0 < (m - 1)$ should be satisfied.

Step 3. For every class $C_i$, which is a non-leaf security class or a leaf security class with two or more immediate ancestors in the hierarchy, CA selects a prime number $e_i$ and computes the corresponding multiplicative inverse $d_i$ to satisfy $d_i = e_i^{-1} \mod \phi(m)$, and then assigns the pair $(e_i, d_i)$ to $C_i$.

Step 4. For every leaf security class $C_i$, which has only one immediate ancestor in the hierarchy, CA randomly generates a secret key $K_i$ and calculates a public parameter $PB_i = K_i \oplus H(K_j, C_i, C_j)$, where $H(\cdot)$ denotes a one-way function and $C_j$ is the immediate
ancestor of $C_i$.
For every class $C_k$, which is a non-leaf security class or a leaf security class with two or more immediate ancestors in the hierarchy, CA calculates a secret key $K_k = K_0^{(d_k \cdot \prod_{t \in C_k} d_t \mod \phi(m))} \mod m$ and a public parameter $PB_k = e_k \cdot \prod_{t \in C_k} e_t$, where $(e_t, d_t)$ is the key pair given by CA for the class $C_t$. Here, $C_t$ is the successor of $C_k$ and $C_t$ is not a leaf class with one predecessor.

Step 5. CA passes the secret key $K_i$ to every class $C_i$ through a secure channel individually and publishes all public parameters and authorized relationships.

### 2.2 Key Derivation Phase

Assume that $C_i$ and $C_j$ are in the POSET hierarchy with relationship $C_i \leq C_j$, where $C_i$ is an immediate successor of $C_j$. $K_i$ and $K_j$ are the secret keys of $C_i$ and $C_j$, respectively. When a user $u_j$ is assigned to $C_j$, $u_j$ can derive the key $K_i$ in $C_i$ from the following formula:

$$K_i = \begin{cases} 
PB_i \oplus H(K_j, C_i, C_j) & \text{if } C_i \text{ is a leaf class with only one immediate ancestor } C_j, \\
PB_i \oplus H(K_j^{PB_j/PB_k}, C_i, C_j) & \text{if } C_i \text{ is a leaf class with only one immediate ancestor } C_k \\
K_j^{(PB_j/PB_i) \mod m} & \text{otherwise}
\end{cases}$$

Without knowing the $\phi(m)$, the user $u_j$ in $C_j$ can efficiently deduce the secret key $K_i$ of $C_i$ with its secret key $K_j$ and the public parameters, $PB_i$ and $PB_j$.

**Theorem 1.** For two secure classes $C_i$ and $C_j$ with relationship $C_i \leq C_j$, $C_j$ can derive the secret key $K_i$ of $C_i$ from the above formula.
Proof. We have the public information \((C_1, \cdots, C_n), (PB_1, \cdots, PB_n)\) and one-way hash function \(H(\cdot)\) in an \(n\) node hierarchy.

(1) \(C_i\) is a leaf class with only one predecessor.

a. \(C_j\) is the immediate predecessor of \(C_i\). It is trivial to get the key from the equation \(K_i = PB_i \oplus H(K_j, C_i, C_j)\).

b. There is a \(C_k\) to satisfy the relationship of \(C_i \leq C_k \leq C_j\) where \(C_k\) is the immediate predecessor of \(C_i\). Because we do not know the secret key \(K_k\), we shall process the following steps. The \(C_t\) is a subset of \(C_j\) and \(C_i\) does not include the leaf class with only one immediate predecessor.

\[
K_i = PB_i \oplus H(K_k, C_i, C_k)
= PB_i \oplus H(K_0^{(d_k \mod \phi(m))} \mod m, C_i, C_k)
= PB_i \oplus H(K_0^{((d_j \cdot \prod_{all \ C_t} d_t) \cdot (e_j \cdot \prod_{all \ C_t} e_t / e_k) \mod \phi(m))} \mod m, C_i, C_k)
= PB_i \oplus H(K_j^{(e_j \cdot \prod_{all \ C_t} e_t / e_k) \mod \phi(m))} \mod m, C_i, C_k)
= PB_i \oplus H(K_j^{PB_j/PB_k \mod m, C_i, C_k})
\]

(2) \(C_i\) is a non-leaf class or a leaf class with multiple predecessors.

The \(C_t\) is a subset of \(C_j\) which does not include the leaf class with only one immediate predecessor, and the \(C_r\) is a subset of \(C_i\) and does not include the leaf class with only one immediate predecessor.

\[
K_i = K_0^{((d_i \cdot \prod_{all \ C_r} d_r) \mod \phi(m))} \mod m
= K_0^{((d_j \cdot \prod_{all \ C_t} d_t) \cdot (e_j \cdot \prod_{all \ C_t} e_t / e_i \cdot \prod_{all \ C_r} e_r) \mod \phi(m))} \mod m
= K_j^{(e_j \cdot \prod_{all \ C_t} e_t / (e_i \cdot \prod_{all \ C_r} e_r)) \mod \phi(m))} \mod m
= K_j^{PB_j/PB_i \mod m}
\]

\(\Box\)
2.3 Examples of Key Derivation

Assume that five hundred security classes \((C_1, C_2, \cdots, C_{500})\) are in the POSET hierarchy structured shown in Figure 2. Table 1 shows how the secret keys and public parameters are assigned in our proposed method.

Table 1: An example of using the proposed scheme

<table>
<thead>
<tr>
<th>Security classes ((C_i))</th>
<th>Public parameters ((PB_i))</th>
<th>Secret keys ((K_i))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C_1)</td>
<td>(e_1 e_2 e_3 e_4 e_6 e_7 e_{10})</td>
<td>(K_0^{d_1 d_2 d_3 d_4 d_6 d_7 d_{10}} \mod \phi(m) \mod m)</td>
</tr>
<tr>
<td>(C_2)</td>
<td>(e_2 e_3 e_{10})</td>
<td>(K_0^{d_2 d_3 d_4 d_{10}} \mod \phi(m) \mod m)</td>
</tr>
<tr>
<td>(C_3)</td>
<td>(e_3 e_6 e_7 e_{10})</td>
<td>(K_0^{d_3 d_4 d_6 d_{10}} \mod \phi(m) \mod m)</td>
</tr>
<tr>
<td>(C_4)</td>
<td>(e_4)</td>
<td>(K_0^{d_4} \mod m)</td>
</tr>
<tr>
<td>(C_5)</td>
<td>(e_5 e_{10})</td>
<td>(K_0^{d_5 d_{10}} \mod \phi(m) \mod m)</td>
</tr>
<tr>
<td>(C_6)</td>
<td>(e_6 e_{10})</td>
<td>(K_0^{d_6 d_{10}} \mod \phi(m) \mod m)</td>
</tr>
<tr>
<td>(C_7)</td>
<td>(e_7)</td>
<td>(K_0^{d_7} \mod m)</td>
</tr>
<tr>
<td>(C_8)</td>
<td>(K_8 \oplus H(K_4, C_8, C_4))</td>
<td>(K_8)</td>
</tr>
<tr>
<td>(C_9)</td>
<td>(K_9 \oplus H(K_4, C_8, C_4))</td>
<td>(K_9)</td>
</tr>
<tr>
<td>(C_{10})</td>
<td>(e_{10})</td>
<td>(K_0^{d_{10}} \mod m)</td>
</tr>
<tr>
<td>(C_{11})</td>
<td>(K_{11} \oplus H(K_7, C_{11}, C_7))</td>
<td>(K_{11})</td>
</tr>
<tr>
<td>(\vdots)</td>
<td>(\vdots)</td>
<td>(\vdots)</td>
</tr>
<tr>
<td>(C_{500})</td>
<td>(K_{500} \oplus H(K_7, C_{500}, C_7))</td>
<td>(K_{500})</td>
</tr>
</tbody>
</table>

Case 1: Deriving the key of a leaf class which has only one immediate predecessor

A user \(u_1\) in \(C_1\) wishes to access information of user \(u_{500}\) in \(C_{500}\), where \(C_{500} \preceq C_1\). The user \(u_1\) can derive \(K_{500}\) with its secret key \(K_1\) and the public parameter of the immediate predecessor of user \(u_{500}\).

\[
K_{500} = PB_{500} \oplus H(K_7, C_{500}, C_7)
\]

\[
= PB_{500} \oplus H(K_1^{PB_1/PB_7} \mod m, C_{500}, C_7)
\]
The proof is stated as follows.

\[
K_{500} = K_{500} \oplus H(K_7, C_{500}, C_7) \oplus H(K_7, C_{500}, C_7)
\]
\[
= PB_{500} \oplus H(K_7, C_{500}, C_7)
\]
\[
= PB_{500} \oplus H(K_0^{d_7 \mod m}, C_{500}, C_7)
\]
\[
= PB_{500} \oplus H((K_0^{d_1d_2d_3d_4d_5d_6d_7^{d_{10}}} e_1e_2e_3e_4e_5e_6^{e_{10}} \mod m, C_{500}, C_7)
\]
\[
= PB_{500} \oplus H((K_1^{e_1e_2e_3e_4e_5e_6^{e_{10}}/e_7} \mod m, C_{500}, C_7)
\]
\[
= PB_{500} \oplus H(K_1^{PB_1/PB_{10}} \mod m, C_{500}, C_7)
\]

**Case 2:** Deriving the key of a non-leaf successor or the key of a leaf class which has more than one immediate predecessor

A user \(u_1\) in \(C_1\) wishes to access the information of user \(u_{10}\) in \(C_{10}\), where \(C_{10} \leq C_1\). The user \(u_1\) can derive \(K_{10}\) with its secret key \(K_1\), its public parameter \(PB_1\) and the public parameter of the user \(u_{10}\).

\[
K_{10} = K_1^{PB_1/PB_{10}} \mod m
\]

The proof is stated as follows.

\[
K_{10} = K_0^{d_{10}} \mod m
\]
\[
= K_0^{(d_{10}d_1d_2d_3d_4d_5d_6d_7)(e_1e_2e_3e_4e_5e_6^{e_7})} \mod m
\]
\[
= K_1^{(e_1e_2e_3e_4e_5e_6^{e_7})} \mod m
\]
\[
= K_1^{(e_{10}e_1e_2e_3e_4e_5e_6^{e_7})/e_{10}} \mod m
\]
\[
= K_1^{PB_1/PB_{10}} \mod m
\]

### 2.4 Dynamic Key Management

In a realistic environment, a key assignment scheme must have the dynamic management ability for adding and deleting security classes.

#### 2.4.1 Adding a security class

When adding a new class, the condition being considered is to check if the new class is a leaf class and whether it has only one immediate predecessor.
Adding a class with only one immediate predecessor

When adding a leaf class $C_n$ under one security class $C_r$. CA only assigns a random secret key $K_n$ and computes a public parameter $PB_n$ to $C_n$ from Step 4 of the key generation phase. Almost all ancestors of the new security class $C_n$ are unaffected. For example, let us add a new class $C_{502}$ under the class $C_7$ in Figure 2. CA assigns a random secret key $K_{501}$ and a public parameter $PB_{501} = K_{501} \oplus H(K_7, C_{501}, C_7)$ to $C_{501}$. While other 500 classes do not change their secret keys and public parameters.

Only if the new immediate predecessor class is a leaf class before the new class $C_n$ is added, then its predecessors should be updated. In this case, the CA needs to assign a pair $(e_n, d_n)$ to $C_n$ and to recompute the public parameters and secret keys of all ancestors along the path to the root class. For example, let us add a new class $C_{502}$ under the class $C_{500}$ in Figure 2. CA assigns a $(e_{500}, d_{500})$ to class $C_{500}$ and recomputes the $PB_{500} = e_{500}$ and $K_{500} = K_0^{d_0} \mod m$. Then CA chooses a random secret key $K_{502}$ to $C_{502}$ and computes the public parameter $PB_{502} = K_{502} \oplus H(K_{500}, C_{502}, C_{500})$, plus the predecessors of $C_{500}$ should be updated as shown in Table 2 while other 496 classes remain unchanged.

Table 2: The new data for predecessors of the new adding class $C_{500}$

<table>
<thead>
<tr>
<th>Class</th>
<th>Secret Key</th>
<th>Public parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_7$</td>
<td>$K_7 = K_0^{d_7d_{500} \mod \phi(m)} \mod m$</td>
<td>$PB_7 = e_7e_{500}$</td>
</tr>
<tr>
<td>$C_3$</td>
<td>$K_3 = K_0^{d_3d_4d_7d_{10}d_{500} \mod \phi(m)} \mod m$</td>
<td>$PB_3 = e_3e_6e_7e_{10}e_{500}$</td>
</tr>
<tr>
<td>$C_1$</td>
<td>$K_1 = K_0^{d_1d_2d_3d_4d_5d_7d_{10}d_{500} \mod \phi(m)} \mod m$</td>
<td>$PB_1 = e_1e_3e_4e_5e_6e_7e_{10}e_{500}$</td>
</tr>
</tbody>
</table>

Adding a class with multiple immediate predecessors

When a new security class $C_n$ is added under two or more immediate ances-
tors or $C_n$ is a non-leaf class, CA assigns a pair $(e_n, d_n)$ to $C_n$ and computes the $K_n$ with the formula in Step 4 of the key generation phase. Next, CA computes the public parameter $PB_n$ by multiplying the new prime $e_n$ with the value of $e$ given by CA of all descendants of $C_n$ except the leaf class with one predecessor. Along the path to the root, CA modifies the public parameters and secret keys of the ancestors of $C_n$ with the steps in the key generation phase. In this case, only the classes along the path from $C_n$ to root should be recomputed to their public parameters and secret keys. Most descendants are still not affected unless the new class is added above a leaf class, and then the leaf class should be recomputed with the public parameter. For example, let us add a new class $C_{503}$ under the classes $C_4$ and $C_5$ in Figure 2. CA assigns key pair $(e_{503}, d_{503})$ to class $C_{503}$ and generates the secret key $K_{503} = K_0^{d_{503}} \mod m$ and a public parameter $PB_{503} = e_{503}$ for $C_{503}$. Next, CA regenerates the secret keys and public parameters by Step 4 of the key generation phase for new ancestor classes, $C_4$, $C_5$, $C_2$ and $C_1$ while other 496 classes remain unchanged.

2.4.2 Cancelling a security class

When removing a security class, the condition being considered is to check whether the class is a leaf class with only one immediate predecessor.

*Removing a class with only one immediate predecessor*

CA simply voids the public parameter and secret key of the removed class while eliminating the class, and almost all classes are not affected. For example, let us remove the class $C_8$ in Figure 2. CA eliminates $PB_8$ and $K_8$ and all 499 public parameters and secret keys of the system remain unchanged.

If the immediate successor becomes a leaf class after the cancelling process is completed, CA chooses a random secret key for the new leaf class and computes the public parameter for the class, as described in the key generation phase. Let us remove the class $C_9$ following the above example. CA eliminates
and deletes the class $C_9$. Next, CA selects a new random secret key $K_4$ for $C_4$ and computes the public parameter $PB_4 = K_4 \oplus H(K_2, C_4, C_2)$ because $C_4$ is now a leaf class. Also, the secret keys and public parameters of classes $C_2$ and $C_1$ should be recomputed while other 495 public parameters and 499 secret keys of the classes in the system remain unchanged.

Removing a non-leaf class

When removing a non-leaf class, CA voids the public parameter and the secret key of the eliminated class when deleting the security class. Then, CA reorganizes the structure and recomputes the public parameters and secret keys for all ancestors of the eliminated class, and most of the successors are not affected. For example, let us remove the class $C_2$ in Figure 2. CA eliminates $PB_2$ and $K_2$ and deletes the class $C_2$ from the system structure then a new structure is organized. Class $C_4$ and class $C_5$ become the immediate successors of the class $C_1$. Therefore, $K_1$ and $PB_1$ are recomputed while other 498 public parameters and secret keys of the system remain unchanged.

If the immediate successor of the eliminated class is a leaf class, the class should be recomputed after the cancelling process has been done. For example, let us remove the class $C_4$ in Figure 2. CA either keeps the old secret keys of the class $C_8$ and class $C_9$, or regenerates new secret keys for $C_8$ and $C_9$, and the CA recomputes the public parameters $PB_8 = K_8 \oplus H(K_2, C_8, C_2)$ and $PB_9 = K_9 \oplus H(K_2, C_9, C_2)$ because the new immediate predecessor is class $C_2$ while other 496 public parameters and secret keys of the system remain unchanged.

Removing a leaf class with multiple immediate predecessors

When removing a leaf class with multiple immediate predecessors, CA voids the public parameter of the eliminated class and the secret key of the class
while deleting the security class. Then, CA recomputes the public parameters and secret keys for all ancestors of the eliminated class along the path to the root class, and most of the successors are not affected. However, if the immediate successor is a leaf class after the cancelling process is finished, most of the successors would be affected. For example, let us remove the class $C_{10}$ in Figure 2. CA eliminates $PB_{10}$ and $K_{10}$ when deleting the class $C_{10}$ from the system. Then CA recomputes or regenerates the secret keys and public parameters of the classes $C_5$, $C_6$, $C_3$, $C_2$, and $C_1$. Note that $C_5$ and $C_6$ only became leaf classes after $C_{10}$ is removed, so they should each be assigned a new secret key and a new public parameter. Other 494 public parameters and secret keys of the classes in the system remain unchanged.

3 Security and Efficiency Comparison

In this section, we will present the security analysis of the proposed scheme in a large leaf class hierarchy. Also, we will examine the required storage and computational complexity.

3.1 Security Analysis

The security analysis of our proposed scheme in possible attacks is described as follows.

1. Modular $m$ attack:
   Any adversary can easily obtain public parameters $e_i$ and the modular $m$, but he could not mathematically prove that the multiplicative inverse $d_i$ can be derived from the known $e_i$ and $m$. Basically, the security of our scheme is similar to that of RSA cryptosystem, where its security is based on the difficulty of factoring $m$ into $m$’s two prime factors. Many factorizing methods are similar to the brute-force attack which would further result in time-consuming efforts. Hence, by using the current
factoring algorithms, it would be inefficient to factor a product of two large primes, especially the strong primes.

2. Contrary attack:
Assume that there are $C_i$ and $C_j$ in a POSET hierarchy and $C_i \leq C_j$. $C_j$ could easily deduce the secret key $K_i$ in $C_i$ using the key derivation method, but not vice versa. If a user in $C_i$ wants to derive the secret key $K_j$ in $C_j$ with his $K_i$ and public parameters, the key $K_j$ could not be obtained because its security is based on the difficulties of factorization and the one-way function inverse. No user is capable of deriving the secret key of his predecessor or other unauthorized classes from the public parameters.

3. Common modulus attack:
When a plaintext $m$ is being encrypted twice using $(e_1, d_1, n)$ and $(e_2, d_2, n)$ with a common modulus $n$, $m$ can be deduced if $e_1$ and $e_2$ are relative primes. By the extension of Euclidean algorithm, $v_1$ and $v_2$ are existent such that the equation $v_1e_1 + v_2e_2 = 1$ holds. Assume that the two ciphertexts are $c_1 = m^{e_1} \mod n$ and $c_2 = m^{e_2} \mod n$, the plaintext $m$ can be obtained by the computation of $m = c_1^{v_1} \times c_2^{v_2} \mod n$. Nevertheless, anyone who obtains the public parameters $(e_i, n)$ and common modulus $m$ still could not affect the security of our scheme since $d_i$ and $K_0$ are being secretly kept by CA. Even though each security class uses the same modulus with different exponential parameters, our scheme is proven to be secured against the common modulus attack.

4. Collaborative attack:
Two or more users in the lower security classes may want to collaboratively derive the secret key of the higher class $C_j$ with mutual information. The users of lower levels can collect the public parameters and
their secret keys, but $K_j = K_0 \prod_{i=1}^{d_i} \mod m$ is only known by CA and $C_j$. Without $K_0$, it is impossible to derive the secret key for unauthorized classes in the collaborative attack.

### 3.2 Efficiency Comparisons

In this subsection, the comparisons among Akl-Taylor’s scheme, Harn-Lin’s scheme, Hwang-Yang’s scheme, and our scheme are presented because all of them are suited for a large leaf class structure. All the schemes need an online CA to keep all the public parameters $PB_i$. Table 3 shows four items: number of primes, maximum public parameter, the complexity of key generation, and key derivation.

#### Table 3: The comparisons of space and computational complexity ($z \leq y \leq n$)

<table>
<thead>
<tr>
<th>Schemes</th>
<th>Number of primes</th>
<th>Maximum public parameter</th>
<th>Key generation</th>
<th>Key derivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Akl-Taylor</td>
<td>$n$</td>
<td>$\prod_{i=1}^{n} e_i$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Harn-Lin</td>
<td>$n$</td>
<td>$\prod_{i=1}^{n} e_i$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Hwang-Yang</td>
<td>$y$</td>
<td>$\prod_{i=1}^{y} e_i$</td>
<td>$O(y)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Our</td>
<td>$z$</td>
<td>$\prod_{i=1}^{z} e_i$</td>
<td>$O(z)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

The generation of prime number will bring issues to the storage size and computing power so we will discuss it first. In the procedure of deciding prime number, let’s assume a POSET hierarchy has $n$ security classes. In Akl-Taylor’s scheme and Harn-Lin’s scheme, both of them choose a prime number for each class so the number is correlated to the total number of nodes $n$.

The number of the primes in Hwang-Yang’s scheme should be computed by the formula:

$$y = \left( \sum_{for \ all \ LG_i} g_i \right) + n_a + n_t,$$

where:

- $LG$ is the term for “leaf-group”, where the security classes are the leaf secure classes, and $LG_i$ denotes the $i$th leaf-group.
• \( g_i \) is a minimum number such that \( \binom{n_i}{k} \geq h_i \). \( \binom{n_i}{k} \), the result of mathematical combination, is a number of ways for choosing \( k \) from \( g_i \) and \( h_i \) denotes the number of classes in \( LG_i \). Here, the \( k \) is the number of primes for distinguishing leaf classes in a leaf group.

• \( n_a \) denotes the number of leaf security classes that have two or more immediate ancestors.

• \( n_t \) denotes the number of non-leaf security classes.

Our scheme is based on the Hwang-Yang’s scheme with a large improvement. The number of the primes in our scheme is computed by the formula:

\[
z = n_a + n_t
\]

where \( n_a \) and \( n_t \) are defined as in Hwang-Yang’s scheme.

**Theorem 2.** In a \( n \) node hierarchy tree, the number of primes used in Akl-Taylor’s scheme, Harn-Lin’s scheme, Hwang-Yang’s scheme, and our proposed scheme are \( n \), \( n \), \( y \) and \( z \) respectively. The condition \( z \leq y \leq n \) should be satisfied.

**Proof.** The number of primes used in Akl-Taylor’s scheme and Harn-Lin’s scheme are the same as the total number of nodes, so both schemes need \( n \) prime numbers. From Equation (1) and Equation (2), the condition \( z \leq y \) is achieved. So every leaf class that has more than one immediate predecessor is the worst case of \( z \), and the upper limit of \( z \) is \( n \). Therefore, the condition \( z \leq y \leq n \) is satisfied. Of course, the worst case for all schemes is that they all need \( n \) prime numbers. □

Assume that there is a large hierarchy which has 500 security classes shown in Figure 2. Both Akl-Taylor’s and Harn-Lin’s schemes must have \( n = 500 \) primes, so the amount of required storage is enormous for a large hierarchy.
In Hwang-Yang’s scheme, \( LG_1 = \{C_8, C_9\} \), \( LG_2 = \{C_{10}\} \) and \( LG_3 = \{C_{11}, \cdots, C_{500}\} \) so it reduces the number of primes to \( y = (2 + 1 + 12) + 1 + 7 = 23 \) primes (with Equation (1)) by the mathematical concept of combination). In the same environment, our proposed scheme only needs \( z = 1 + 7 = 8 \) primes with the Equation (2). By comparing the required number of primes, our scheme is substantially lower than other schemes since \( 8 \leq 23 \leq 500 \).

The length of the maximum number of public parameters results in the computational load of CA. It costs \( n \) multiplications in both Akl-Taylor’s and Harn-Lin’s schemes when the maximum number of public parameters is being computed. There are \( y \) multiplications in Hwang-Yang’s scheme and \( z \) multiplications in our scheme. For example, in the case of Figure 2, there are 500 security classes in the POSET structure. We first calculated the number of primes to be used for each class by using the above schemes in Table 4. From the results, the maximum multiplications used for every scheme are 499 times in Akl-Taylor’s scheme, 500 times in Har-Lin’s scheme, 23 times in Hwang-Yang’s scheme and 8 times in our scheme. Note that this data shows only the number for the most complex computational class obviously, our scheme is the most efficient one.

The computational complexity of the key generation is usually proportional to the number of primes. The computational complexity of our scheme is the most efficient comparing with other schemes, where \( O(z) \leq O(y) \leq O(n) \) and the key derivation in all schemes is only \( O(1) \) if the computation of the hash function is disregarded. Our scheme obviously executed more efficiently than other schemes from the compared results.

The storage space of secret key depends on the access method. Both Akl-Taylor’s and Harn-Lin’s schemes assign the public parameters directly, so that each class records its own key and it’s successors’. On the contrary, both Hwang-Yang’s and our schemes utilize the key derivation method to obtain a
Table 4: Public parameters for each security class in different schemes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Security classes</th>
<th>Public parameters (PBi)</th>
<th>Number of primes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Akl-Taylar’s</td>
<td>$C_1$</td>
<td>$e_1$</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$C_2$</td>
<td>$e_1e_3e_6e_7e_{10} \cdots e_{500}$</td>
<td>495</td>
</tr>
<tr>
<td></td>
<td>$C_3$</td>
<td>$e_2e_4e_5e_{10}$</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td></td>
<td>$C_8$</td>
<td>$e_1e_2e_3e_4e_5e_6e_7e_9 \cdots e_{500}$</td>
<td>499</td>
</tr>
<tr>
<td></td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td></td>
<td>$C_{500}$</td>
<td>$e_1 \cdots e_{499}$</td>
<td>499</td>
</tr>
<tr>
<td>Har-Lin’s</td>
<td>$C_1$</td>
<td>$e_1 \cdots e_{500}$</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>$C_2$</td>
<td>$e_2e_4e_5e_8e_{10}$</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>$C_3$</td>
<td>$e_3e_6e_7e_{10} \cdots e_{500}$</td>
<td>494</td>
</tr>
<tr>
<td></td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td></td>
<td>$C_8$</td>
<td>$e_8$</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td></td>
<td>$C_{500}$</td>
<td>$e_{500}$</td>
<td>1</td>
</tr>
<tr>
<td>Hwang-Yang’s</td>
<td>$C_1$</td>
<td>$e_1 \cdots e_{23}$</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>$C_2$</td>
<td>$e_2e_4e_5e_8e_9$</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>$C_3$</td>
<td>$e_3e_6e_7e_{10}e_{12} \cdots e_{23}$</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td></td>
<td>$C_8$</td>
<td>$e_8$</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td></td>
<td>$C_{500}$</td>
<td>$e_{20}e_{21}e_{22}e_{23}$</td>
<td>4</td>
</tr>
<tr>
<td>Our</td>
<td>$C_1$</td>
<td>$e_1 \cdots e_8$</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>$C_2$</td>
<td>$e_2e_4e_5e_8$</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>$C_3$</td>
<td>$e_3e_6e_7e_8$</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td></td>
<td>$C_8$</td>
<td>$K_8 \oplus H(K_4, C_8, C_4)$</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td></td>
<td>$C_{500}$</td>
<td>$K_{500} \oplus H(K_7, C_{500}, C_7)$</td>
<td>0</td>
</tr>
</tbody>
</table>
successor’s key so that each class only records its secret key. For example, in the case shown in Figure 2, both Akl-Taylor’s and Harn-Lin’s schemes need storage space for 1991 keys where class \( C_1 \) records 500 keys, \( C_2 \) records 6 keys, \( C_3 \) records 494 keys, \( C_4 \) records 3 keys, \( C_5 \) and \( C_6 \) records 2 keys, \( C_7 \) records 491 keys, and all leaf classes record their own keys only. Both Hwang-Yang’s and our schemes need space for 500 keys only. Let us assume each public parameter is 1024 bits, so both Akl-Taylor’s and Harn-Lin’s schemes need a size of 1991 \( \times \) 1024 bits for secret key storage. Both Hwang-Yang’s and our schemes only need a size of 500 \( \times \) 1024 bits for key storage.

4 Conclusion

We have proposed a new key assignment to solve the problems of access control for a large POSET hierarchy. With substantially fewer numbers of primes and noticeable storage saving, the proposed scheme offers a worthy improvement in efficiency. Additionally, our scheme can also ensure that the security of the predecessor could not be revealed by any unauthorized successors from the violation of access policy. Moreover, our proposed scheme efficiently offers dynamic key management when a class is added or removed while satisfying the requirements listed in Section 1.2.

In an organization, the number of leaf classes is significantly larger than the number of non-leaf classes. The alternation in leaf classes happens more frequently than in non-leaf classes. Our proposed scheme offers a perfect reduction on the number of primes, which results in a notable efficiency improvement. When a class is added or removed, recomputing for all the parameters and keys of the organization became unnecessary. Our proposed scheme has the lowest amount of key regeneration when the leaf class is added or removed. Compared with other schemes, our proposed scheme has the highest efficiency and still maintains the essential and sufficient security in a large POSET hi-
erarchy.

5 Acknowledgement

This work was supported in part by the Taiwan Information Security Center (TWISC) and the National Science Council under the grants NSC95-2218-E-001-001, and NSC95-2218-E-011-015.

References


[14] Xuan Hung Le and Sungyoung Lee and Young-Koo Lee and Heejo Lee and Murad Khalid and Ravi Sankar, “Activity-oriented access control


