A Cryptographic Key Assignment Scheme in a Hierarchy for Access Control *

Min-Shiang Hwang

Department of Information Management
Chaoyang University of Technology
168, Gifeng E. Rd., Wufeng,
Taichung County, TAIWAN 413, R.O.C.
Email: mshwang@mail.cyut.edu.tw
Fax: 886-4-3742337

September 14, 2001

*This research was partially supported by the National Science Council, Taiwan, R.O.C., under contract no.: NSC86-2621-E-324-001.
Abstract

Access control is one of mechanisms for data protection in a computer system. Many literatures based on cryptography have been proposed to solve the problem of access control in hierarchic structures. Recently, Liaw and Lei proposed an optimal heuristic algorithm for multilevel data security. But, their heuristic algorithm can only be used in a tree structure, which is a special case of a partially-ordered hierarchy. In this article, we present a modification of their algorithm that enables the algorithm to be used in a partially-ordered structure.

Keywords: Access control, Cryptography, Data security, Multilevel.

1 Introduction

In 1983, Akl and Taylor [1] proposed an elegant solution for controlling access to information among a group of users in a hierarchy. In such a hierarchy, the users and the information items they own are divided into a number of disjoint sets of security classes, \( C_1, C_2, \ldots, C_n \), and the relationships among security classes correspond to a partially-ordered set (poset, for short) hierarchy, as shown in Figure 1.

\[
\begin{align*}
C_1 & \quad C_2 \\
& \quad C_3 \\
C_4 & \quad C_5 \\
& \quad C_6
\end{align*}
\]

Figure 1: An example of a partially ordered hierarchy.
For the partially-ordered structure, let $C_i$ and $C_j$ be nodes in this structure. Furthermore, $C_i \leq C_j$ where $C_i$ is called the descendant of $C_j$ and $C_j$ is called the ancestor of $C_i$. If there is another existing node $C_k$ such that $C_i \leq C_k \leq C_j$, $C_i$ is called an immediate descendant (or children) of $C_j$, and $C_j$ is called an immediate ancestor (or father) of $C_i$. If there is no another node $C_i$ such that $C_i \leq C_j$, then $C_j$ is called a leaf security class.

In the Akl-Taylor scheme [1], each security class $C_i$ is assigned a distinct prime to be its public parameters, $PB_i$. The secret key, $K_i$, for each security class $C_i$ is calculated with the public parameters $PB_i$ by a central authority in the system. The information items owned by $C_i$ are encrypted by an available symmetric (one-key) cryptosystem with the enciphering key $K_i$. This information can only be retrieved by the security class $C_j$, where $C_i \leq C_j$. Using the public parameters, $PB_i$ and $PB_j$, and the secret key $K_j$, $C_j$ can derive $K_i$ to decipher the information items owned by $C_i$. The Akl-Taylor scheme suggests an elegant solution in a poset hierarchy for the access control problem. However, a large amount of storage to store the public parameters is required. In 1985, MacKinnon et al. [2] presented an improved algorithm for the Akl-Taylor scheme, called the canonical assignment, to reduce the value of public parameters. This scheme certainly reduces the number of distinct primes. However, a large amount of storage is still needed to store the $PB_i$.

In addition, it is difficult to find an optimal canonical algorithm, which can make the assignment optimal for an arbitrary poset hierarchy. In 1988, Sandhu [3] used one-way functions to create a cryptographic implementation of a tree hierarchy for access control. (The tree hierarchy is a special case of a poset hierarchy.) Each secret key $K_i$ for security class $C_i$ is generated with its own identity (ID) name and its immediate ancestor’s secret key through a one-way function. In the scheme, no extra public parameter is needed for the key derivation. However, there are two drawbacks: one is that computa-
tional overhead is incurred in deriving keys. The other is that the proposal is only implemented in a tree hierarchy, which is less flexible in applications. In 1990, Harn and Lin [4] proposed an approach similar to the Akl-Taylor scheme. But, instead of using a top-down design approach as in the Akl-Taylor scheme, Harn and Lin presented a bottom-up key generating scheme. In the Harn-Lin scheme, the size of storage space to store the public parameters for most security classes is smaller than that is needed for the MacKinnon et al. and Akl-Taylor schemes. However, in the Harn-Lin scheme when there are many security classes in the system, a large amount of storage space is required to store the public parameters. In 1992, Chang et al. [5] and in 1993 Liaw et al. [6] proposed other schemes. Their schemes are based on Newton’s interpolation method and a predefined one-way function. However, the computations needed for key generation and derivation in their schemes are time consuming. Furthermore, their schemes are insecure against cooperative attacks [7, 8].

Recently, Liaw and Lei [9] presented an optimal heuristic algorithm for assigning cryptographic keys in a tree structure for multilevel data security. The Liaw-Lei scheme uses a top-down design approach, as does the Akl-Taylor scheme [1]. The Liaw-Lei scheme not only reduces the amount of storage required for storing public parameters, but also is simple and efficient in generating and deriving keys. However, their heuristic algorithm can only be used in a tree structure, which is a special case of a partially-ordered hierarchy. In this article, we present a modification of their algorithm that enables the algorithm to be used in a partially-ordered structure; that is, a user at a higher level can derive the keys of other users below him from his own cryptographic key.
Table 1: Algorithm for assigning a prime to each node.

<table>
<thead>
<tr>
<th>Input:</th>
<th>A hierarchy $H$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>A set of $n_{R_i}$, primes or the products of several primes, assigned to the nodes of $H$</td>
</tr>
</tbody>
</table>

/* Compute $n_{R_i}$ for each node $R_i$ starting from the leaves and going up to the root. */
1. For each non-root node $R_i$ do
2. if $R_i$ has only one child and no sibling
3. then $n_{R_i} = p_i \prod n_{R_j}$
4. else $n_{R_i} = p_e$, where $p_e$ is the smallest prime differs from all other primes previously used in $R_i$ and $lR_i$;
5. $n_{R_0} = 1$;

2 A Modified Liaw-Lei Scheme

In this section, we introduce two algorithms created by modifying the algorithm Assignment in [9]. We assume that, as in the Akl-Taylor, Hanr-Lin, and other schemes, there is a central authority (CA, for short) in the system. The main task of the CA is to generate secret keys, public identity integers, and derivation keys for all security classes. In Table 1, corresponding to phase 1 of the algorithm Assignment in [9], we assign each node $R_i$ of the hierarchy a prime or the product of several primes, $n_{R_i}$, starting from the leaves and working up to the root. In Table 2, corresponding to phase 2 of the algorithm Assignment in [9], we compute a public parameter $lR_i$ for each node $R_i$, starting from the root and working down to the leaves. For convenience, we use the following notation in our proposed algorithms:

- $\overline{R_i}$: denotes the father node of $R_i$,
- $R_i$: denotes the children nodes of $R_i$,
- $lR_i$: denotes the sibling nodes of $R_i$, and
- $\overline{lR_i}$: denotes the uncle nodes of $R_i$. That is, the relationship of $\overline{lR_i}$ to $\overline{R_i}$ is that of sibling.

We call a node crossbred if the node has two or more father nodes.

Whenever all nodes have been assigned a public parameter by the algo-
Table 2: Algorithm for computing public parameters.

<table>
<thead>
<tr>
<th>Input:</th>
<th>A hierarchy ( H ) with primes assigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>A set of ( t_{R_i} ), public parameters, assigned to each node of ( H )</td>
</tr>
</tbody>
</table>

\[
\text{/* Compute } t_{R_i} \text{ for each node } R_i \text{ starting from the root} \\
\text{and going down to leaves */} \\
1. \quad t_{R_0} = 1; \\
2. \quad \text{For each non-root node } R_i \text{ do} \\
3. \quad \text{if } R_i \text{ is non-crossbred node} \\
4. \quad \text{then } t_{R_i} = p_l t_{R_l} \prod n_{R_i}; \\
5. \quad \text{else } t_{R_i} = \text{lcm}(t_{(R_i)}) \prod n_{R_i}, \text{ and} \\
6. \quad \text{for all } R \in lR_i \text{ do } t_R = t_{RnR_i}; \\
\]

Algorithms in Table 1 and Table 2, the CA computes the secure key \( K_{R_i} \) for each node \( R_i \) as follows.

\[
K_{R_i} = K_0^{t_{R_i}} \mod m, \quad (1)
\]

where \( m \) is a multiplication of two large prime \( p \) and \( q \); \( K_0 \) is a random secret integer. \( K_0 \) and \( m \) are relatively prime. \( K_0, p, \) and \( q \) are kept secret and \( m \) is public.

Once the secret node keys \( K_{R_i} \) and the corresponding public information parameter \( t_{R_i} \) are generated, the node \( R_i \) can derive the secret key \( K_{R_j} \) for its descendant \( R_j \) by computing the following:

\[
K_{R_j} = K_{R_i}^{t_{R_j}/t_{R_i}} \mod m \quad \text{if} \quad R_j \leq R_i.\quad (2)
\]

Next, we will show that the key derivation in Equation (2) is correct as follows.

\[
K_{R_i}^{t_{R_j}/t_{R_i}} \mod m \\
= K_0^{t_{R_i} t_{R_j}/t_{R_i}} \mod m \\
= K_j.
\]
3 Security Analysis

In this section, we prove the correctness of our security enforcement scheme. We need to satisfy the following two statements:

- A user at a higher level can derive the keys of the other users below him from his own secret key, while the opposite is not allowed.
- Two or more users collaborating at a lower level of the system cannot derive a higher-level key to which they are not entitled.

Akl and Taylor [1] proved the following two conditions:

\[ t_{R_a} \mid t_{R_b} \text{ if and only if } R_a \leq R_b; \quad (3) \]

\[ \gcd_{R_b \geq R_a} t_{R_a} \not\mid t_{R_b}. \quad (4) \]

To ensure the security of our scheme, we also insert a limit,

\[ \frac{\min_{R_b \geq R_i} t_{R_i}}{\gcd_{R_j \geq R_a} t_{R_j}} = 1, \]

as in [9], to prevent collaborated attacks. We prefer to reference [9] for details.

Thus Equation (4) is rewritten as follows:

\[ \begin{align*}
\text{if } & \frac{\min_{R_b \geq R_i} t_{R_i}}{\gcd_{R_j \geq R_a} t_{R_j}} \neq 1 & \text{then collaborating users have no privileges} \\
\text{else } & \gcd_{R_b \geq R_a} t_{R_b} \not\mid t_{R_a}. \quad (5)
\end{align*} \]

In the following we verify the security of our scheme.

Lemma 3.1 The proposed scheme satisfies Equation (3).

Proof. We divide the proof into the following two cases.

Case 1: if \( R_a \leq R_b \) then \( t_{R_a} \mid t_{R_b} \).

For convenience, we rewrite step 4 and step 5 in Table 2 as follows:

\[ \begin{align*}
\left\{ \begin{array}{l}
\quad t_{R_i} = t_{R_i}^{\prod n_{IR_i}p_1}, \\
\quad t_{R_i} = \text{lcm}(t_{R_i})^{\prod n_{IR_i}p_1}.
\end{array} \right. \quad (6)
\]
From the above equations, we know that

\[ t(R_a) \models t(R_b). \] (7)

Thus, this case satisfies Equation (3).

**Case 2:** if \( R_a \not\leq R_b \) then \( t(R_b) \not\models t(R_a) \).

We divide the proof into the following three subcases.

**Case 2.1:** neither \( R_a \) or \( R_b \) is a crossbred node.

In this subcase, the proof is the same as Lemma 1 in [9].

**Case 2.2:** \( R_a \) is a crossbred node.

We divide the proof into the following three subcases.

**Case 2.2.1:** \( R_b < R_a \)

Since \( t(R_b) \) has the factor of \( t(R_a)p_1 \), by transitivity, \( t(R_b) \) has the factor of \( t(R_a)p_1^i \), where \( i \geq 1 \). Thus, \( t(R_b) \not\models t(R_a) \).

**Case 2.2.2:** \( R_b \leq \overline{tR_a} \), that is, \( R_b \) is a uncle node of \( R_a \) or descendant node of \( \overline{tR_a} \).

From step 6 in Table 2, we know that \( t(R_b) \) has the factor of \( n_{R_a} \), but \( t(R_a) \) does not contain \( n_{R_a} \). Thus \( t(R_b) \not\models t(R_a) \).

**Case 2.2.3:** \( R_b > \overline{tR_a} \).

Let the common ancestor of \( R_a \) and \( R_b \) be \( \overline{R_{ab}} \). Assume that \( \overline{R_a} \), the ancestor of \( R_a \), and \( \overline{R_b} \), the ancestor of \( R_b \), are children nodes of \( \overline{R_{ab}} \). From Equation (6), we can easily see that both \( t(R_a) \) and \( t(R_b) \) contain the product \( t(R_{ab})p_1 \). Thus, after division by \( t(R_a) \) and \( t(R_b) \), \( t(R_a) \) does not contain \( n_{R_a} \) but \( t(R_b) \) does. Thus, \( t(R_b) \not\models t(R_a) \) and this subcase satisfies Equation (3).

**Case 2.3:** \( R_b \) is a crossbred node.

We divide the proof into the following three subcases.

**Case 2.3.1:** \( R_a > R_b \)

In this subcase, the proof is the same as in case 2.2.1.

**Case 2.3.2:** \( R_a \leq \overline{tR_b} \).
From Equation (6), we know that
\[ t_{R_a} | t_{R_b} \quad (8) \]
\[ t_{R_b} \not| t_{R_a} \quad (9) \]
The above equations implying \( t_{R_a} \not| t_{R_b} \). From Equation (6), we know that \( t_{R_b} \) contains \( n_{R_b} \), but \( R_a \) does not. Thus, \( t_{R_b} \not| t_{R_a} \) and this subcase satisfies Equation (3).

**Case 2.3.3:** \( R_a > tR_b \).

In this subcase, the proof is similar to that for Case 2.2.3. **Q.E.D**

**Lemma 3.2** The modified scheme satisfies Equation (5).

**Proof.** The proof is similar to that of Lemma 2 in [9]. **Q.E.D**

From Lemma 3.1 and 3.2, the following theorem is derived directly.

**Theorem 3.1** The modified scheme satisfies Equation (3) and (5).

## 4 Conclusions

We have proposed an access control scheme for a partially-ordered hierarchy by modifying Liaw-Lei’s scheme, which can only be used in a tree structure. We have also shown that the modified scheme prevents cooperative attacks.

## References


