Design of a Lattice-based Access Control Scheme

Chia-Chu Chiang¹, Coskun Bayrak¹, Remzi Seker¹, Umit Topaloglu², Murat Demirer¹,³, Nasrola Samadi¹, Suleyman Tek¹, Bian Jiang¹, GuangXu Zhou¹, and Xiaoran Wang¹

¹Department of Computer Science
University of Arkansas at Little Rock
2801 South University Avenue
Little Rock, Arkansas, USA
²UAMS
³Computer Science Department
Faculty of Science and Letters
Kultur Universitesi, Istanbul, Turkey
{cxchiang|cxbayrak}@ualr.edu

Abstract—We survey the literature for access control schemes in a user hierarchy. Some schemes have already been shown to be insecure or incorrect. Many schemes assume very restrictive subordinating relationships existing in a hierarchy where users are grouped into partially ordered relationships without taking resources into consideration. We believe that a practical access control scheme should support access control in a lattice where users and resources are both together grouped into partially ordered relationships. In this paper, we present a scheme to achieve this goal. We also study existing schemes for their efficiency and performance. Based on the results of the study, we design an efficient scheme to support dynamic key management.

Keywords—Access Control, Cryptographic Keys, Cryptography, Dynamic Key Management, Hierarchical Access Control, Lattice, and Rekeying

I. INTRODUCTION

An access control scheme ensures that only legitimate users are able to access resources assigned to them. A simple access control scheme is to create secret keys for all the resources and only users who are able to access resources using legitimate keys. A major problem of this kind of scheme is that users will be overwhelmed with keys as the number of resources increases. Akl and Taylor [1] proposed a cryptographic solution to this problem for access control in a user hierarchy represented by a partially ordered set. In the hierarchy, users are grouped into classes. The partial order on the hierarchy implies that a resource accessible to a class C_i is also accessible to a class C_j if C_j ≤ C_i. The Akl and Taylor’s scheme allows a user in a higher privileged class to derive the keys of its child nodes. Nowadays, there are many proposed schemes for access control in a user hierarchy. Some schemes [2-4] are proved to be insecure or incorrect in the articles [5-7]. Others [8-11] are claimed to be insecure in the article [12].

Many schemes [13-20] for access control assume very strict subordinating relationships in a user hierarchy. Users in a class have the same access to the same set of resources. However, access control for users having different access rights to multiple resources widely exists in practice. Users and resources usually have many-to-many relationships. For example, a company might provide services for different membership groups to access different resources. Generally, in such kind of applications with this type of access control, users are grouped into a partially ordered hierarchy as well as resources. Unfortunately, it is not clear to see whether resources are merged into the user hierarchies presented in a large number of published access control schemes. In this paper, we propose a Lattice-based access control key management scheme where users and resources are merged into a partially ordered lattice. The scheme employs a Central Authority (CA) to maintain partially ordered relations of users and resources. The CA is also responsible for assigning keys to classes. The scheme employs a simple and efficient key generation and derivation algorithm. In addition, the scheme further improves the efficiency of key management with the introduction of S-nodes to the lattice.

The paper is organized as follows. A literature survey is presented in Section 2. The main purpose of this survey is to understand how existing schemes differently handle access control in a user hierarchy. Most importantly, how efficiently these schemes perform dynamic key management in terms of performance, storage, and key updates. Section 3 lists a set of design requirements of our proposed scheme. According to the design requirements, our scheme with the key generation and derivation algorithm is presented in Sections 4 and 5. Section 6 analyzes the secure tolerance of our scheme. Our scheme is demonstrated with an example in Section 7. Finally, the paper is summarized in Section 8.

II. LITERATURE SURVEY OF EXISTING ACCESS CONTROL SCHEMES

After conducting a literature survey on access control, we classified the schemes into two categories: ones with public cryptography and ones without public cryptography. Since we are interested in designing a scheme to be applied in a public key infrastructure, the schemes without public cryptography [6, 13, 30] will not be considered in this research. For the schemes with public cryptography, the schemes are evaluated in terms of the computation time of key generation and derivation and the storage space required in key generation. The computation time is measured by the average number of operations performed in key generation and derivation. The
storage space is defined as the average number of secret keys and public parameters needed to perform computations. In addition, the scheme is also evaluated in terms of rekeying complexity defined as the average number of keys required to be updated as a user joins or leaves a class.

One of the key efficiency measures for Lattice-based access control schemes is the complexity of operations in key generation and derivation algorithms. Many schemes [12, 20, 22-23] using simple exclusive-or operations and hash functions in their key generation and derivation algorithms give reasonable performance $O(n)$ where $n$ is the number of nodes in the hierarchy. Several schemes [1, 8, 10, 13, 24-27] are less efficient due to the use of modular exponentiation, discrete logarithms, polynomial interpolations, and additional usage of encryption/decryption.

Schemes giving reasonable performance in key generation and derivation may require a large amount of storage space of public parameters and secret keys for computation. Such kind of schemes includes Akl and Taylor, Chen et. al., Chen and Huang, Chick and Tavares, MacKinnon et. al., and Zhong [1, 13, 20, 28-29].

Rekeying as a user joining or leaving a class determines a scheme’s efficiency in dynamic key management. Very few schemes [12, 15] have rekeying handled locally where rekeying does not propagate to successors and predecessors of the affected class. Majorities of schemes [20, 30] require rekeying of all the successors of the affected class. Few schemes [1, 28] even require all the classes in the system to be rekeyed.

III. DESIGN REQUIREMENTS OF LATTICE BASED SCHEME

In general, a designer faces a variety of tradeoffs between algorithm complexity, efficiency, security, and space requirements. The following is a set of design requirements we consider,

- The scheme must support access control in a lattice represented by partially ordered relationships of users as well as resources.
- The scheme must be secure. Users in a higher-level class should be allowed to derive the keys of subordinating classes but not the other way around.
- The scheme must be efficient. The minimum requirements of our scheme are that all the keys should not be derived as a user joins or leaves a class. The computation of key generation and derivation should not be in an exponential time.
- The scheme should not require a large amount of storage space for storing parameters and secrets.
- The scheme must ensure to combat the ex-member problem. As a user joins or leaves a class, the keys of the impacted classes must be updated.

To meet the above requirements, we first propose a lattice to form partially ordered relationships of users and resources. The lattice will support many-to-many relationships between users and resources. A sample implementation of the lattice can be found in [31]. Our scheme uses a hash function and the exclusive-or operation to implement the key generation and derivation algorithm. In [32], Sklavos and Koufopavlou had the experiments on the performance of hash functions. The result of their work will help select a hash function with the better performance. The rekeying of all the successors of the affected class is enforced in the scheme to combat the ex-member problem. We also introduce a new concept of S-class nodes in the lattice to improve the performance of the key management protocol. The additions of S-nodes to the lattice also reduce the total amount of public parameters required for the key computation.

IV. KEY GENERATION AND DERIVATION

Suppose $C_i$ is the immediate successor of $C_j$ where $C_j \leq C_i$ in a lattice. CA randomly selects two large positive integers as a secret key $K_j$ and a public parameter $L_j$ for the class $C_j$. CA also selects two large positive integers as $K_i$ and $L_i$ for the class $C_i$. A public parameter $\alpha_{i,j} = h(K_i \oplus L_j) \oplus K_j$ is computed and assigned to the edge of $C_i$ and $C_j$.

To derive the key of $C_j$, CA uses the public parameter $L_j$ of $C_j$ and computes $h(K_i \oplus L_j) \oplus \alpha_{i,j}$ for $K_j$.

V. DYNAMIC KEY MANAGEMENT PROTOCOL

A. Adding a Class

Assume a class $C_k$ is to be added to the lattice between $C_i$ and $C_j$ where $C_j \leq C_i$. CA randomly selects two large positive integers as $K_k$ and $L_k$ for $C_k$. For any class $C_i$ with $C_k \leq C_i$, CA computes $\alpha_{i,k} = h(K_i \oplus L_k) \oplus K_k$. For any class $C_j$ with $C_j \leq C_k$, CA computes $\alpha_{k,j} = h(K_k \oplus L_j) \oplus K_j$.
B. Deleting a Class

Assume a class $C_k$ is to be removed from the lattice. CA removes $C_k$ with $K_k$ and $L_k$. For any class $C_i$ with $C_k \leq C_i$, CA removes $\alpha_{i,k}$ and creates $\alpha_{i,j}$. For any class $C_j$ with $C_j \leq C_k$, CA removes $\alpha_{k,j}$.

C. Adding an S-node

Suppose a lattice has $C_x$, $C_i$, and $C_j$ where $C_j \leq C_i \leq C_x$. A new class $C_k$ is to be added to the lattice with $C_j \leq C_k$. CA first removes $\alpha_{i,j}$ and then groups $C_i$ and $C_k$ into an S-node class, $S_m$. CA then randomly selects two large positive integers as $K_m$ and $L_m$ for $S_m$. CA computes $\alpha_{m,j}$ between $S_m$ and $C_j$. If there is another new class $C_z$ to be added to the lattice with $C_j \leq C_z$, CA just simply adds $C_z$ to $S_m$.

D. Deleting an S-node

Deleting a class from an S-node is just simply removing the class from the S-node. If the resulting S-node only has one class left, our scheme will still keep the class in the S-node because there is a high probability that a new class might join the S-node later.

VI. SECURITY ANALYSIS

To make sure a user in a lower privileged class not to derive the keys of its predecessors, let us look at the key derivation algorithm. Assume two classes $C_i$ and $C_j$ in a lattice with $C_j \leq C_i$. To derive the key of $C_i$, users in $C_j$ require to derive $K_i$ from $h(K_i \oplus L_j) \oplus \alpha_{i,j}$. Although $L_j$ and $\alpha_{i,j}$ are public parameters, the users in $C_j$ still have no way to generate $K_i$ since this would require inversion of the $h$ function.

In addition, users of a class in an S-node have no way to derive the keys of the other sibling classes because each class has its own private key. The key of an S-node is only used for the key derivations of its successors of the S-node. Thus, the users in the S-node can access their common resources in the lower privileged classes.

VII. AN EXAMPLE

Consider the access control table in Table I and the corresponding lattice in Figure 1. The user access control table is a conceptual model that specifies the rights that each user accesses for each file. Each row in this table indicates a list of files that can be accessed by this particular user. Each column indicates a list of users who have the rights to access this particular file.

The user access control table is implemented into a lattice hierarchy shown in Figure 1. Each node in this lattice specifies the user with the corresponding file that this user has the rights to access. The lattice in Figure 1 also specifies the precedence of a user to the subordinates. For example, USER 2 in $C_2$ has the rights to access $F_1$ in $C_4$ and USER 4 has the rights to access $F_1$, $F_2$, and $F_3$ in $C_2$, $C_3$, and $C_4$. The CA will create $K_i$, $L_i$, $\alpha_{i,j}$, and $S_i$ of all the nodes based on the neighborhood relationship among the nodes in this lattice. We also assume all the files need to be encrypted before saving on a computer and decrypted for uses by the users.

Next, we add $\{\text{USER 5}, \{F_4\}\}$ and $\{\text{USER 6}, \{F_1, F_4\}\}$ to the access table as shown in Table II where new users USER 5 and USER 6 and new files $F_4$ are joined for the access control.

A lattice corresponding to Table II is depicted in Figure 2 where $C_2$ and $C_5$ are grouped into an S-node, $S_7$. An edge between $S_7$ and $C_4$ is created with the public parameter, $\alpha_{7,4}$ indicating that User 2 and User 6 are both sharing a common file $F_1$ at the same precedence level in the lattice. CA improves the performance in fetching $F_1$ by following the common link $\alpha_{7,4}$ for User 2 and User 6. Since USER 2 is using the common
link to C₄, the link between C₂ and C₄ is no longer needed for access control. Although USER 6 in C₅ is in S₇ with USER 2 in C₂ and USER 6 is not sharing F₄ with USER 2, therefore CA needs to create a link between C₅ and C₆. The link allows CA to access F₄ for USER 4 and USER 6 but not USER 2.

The corresponding lattice of table IV is depicted below. Now, since USER 7 is sharing F₁ with USER 2 and USER 6, there is no action to be taken in creating its own link to C₄. Instead, CA just joins C₈ to S₇.

In this scenario, C₈ just joins S₇ with its own key K₈ and public parameter L₈. Table V lists Kᵢ, Lᵢ, αᵢⱼ, and Sᵢ of all the nodes in the lattice depicted in Figure 3.

At this moment, assume USER 4 in C₁ is going to access F₁ and F₄. CA is required to do the following steps to derive K₄ and K₅ for the encryption/decryption of F₁ and F₄.

1. Compute K₅ = h(K₁ ⊕ L₅) ⊕ α₁₅;
2. Compute K₄ = h(K₇ ⊕ L₄) ⊕ α₇₄; and
3. CA sends K₄ and K₅ to the users in C₁ to encrypt/decrypt F₁ and F₄.

Next, let us remove USER 2 from the user access table. It means that F₂ will no longer become inaccessible to USER 4 in C₁. In this scenario, CA just removes C₂ from S₇ without impacting the other nodes in the lattice. The resulting lattice is depicted in Figure 4.

At this moment, assume USER 4 in C₁ is going to access F₁ and F₄. CA is required to do the following steps to derive K₄ and K₅ for the encryption/decryption of F₁ and F₄.

1. Compute K₅ = h(K₁ ⊕ L₅) ⊕ α₁₅;
2. Compute K₄ = h(K₇ ⊕ L₄) ⊕ α₇₄; and
3. CA sends K₄ and K₅ to the users in C₁ to encrypt/decrypt F₁ and F₄.

Next, let us remove USER 2 from the user access table. It means that F₂ will no longer become inaccessible to USER 4 in C₁. In this scenario, CA just removes C₂ from S₇ without impacting the other nodes in the lattice. The resulting lattice is depicted in Figure 4.

Table III creates Kᵢ, Lᵢ, αᵢⱼ, and Sᵢ of all the nodes in the lattice as shown in Figure 2. The link with α₂₄ is removed from the table.

Table IV. A USER ACCESS CONTROL TABLE WITH ADDITIONS OF ({USER 7}, {F₁, F₅}) and ({USER 4}, {F₅})

<table>
<thead>
<tr>
<th>USER</th>
<th>F₁</th>
<th>F₂</th>
<th>F₃</th>
<th>F₄</th>
<th>F₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER 1</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>USER 2</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER 4</td>
<td>√</td>
<td></td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>USER 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER 6</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
</tbody>
</table>

The corresponding lattice of table IV is depicted below. Now, since USER 7 is sharing F₁ with USER 2 and USER 6, there is no action to be taken in creating its own link to C₄. Instead, CA just joins C₈ to S₇.

In this scenario, C₈ just joins S₇ with its own key K₈ and public parameter L₈. Table V lists Kᵢ, Lᵢ, αᵢⱼ, and Sᵢ of all the nodes in the lattice depicted in Figure 3.

Table V. COMPUTATIONS OF PARAMETERS AND KEYS IN FIGURE 3

<table>
<thead>
<tr>
<th>USER</th>
<th>L₁</th>
<th>α₁₂</th>
<th>K₁</th>
<th>S₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>C₁</td>
<td>L₁</td>
<td>α₁₂ = h(K₁ ⊕ L₁ ⊕ K₂ ⊕ L₃ ⊕ K₃)</td>
<td>K₁</td>
<td>–</td>
</tr>
<tr>
<td>C₂</td>
<td>L₂</td>
<td>α₂₃ = h(K₂ ⊕ L₃ ⊕ K₃)</td>
<td>K₂</td>
<td>S₇</td>
</tr>
<tr>
<td>C₃</td>
<td>L₃</td>
<td>α₃₅ = h(K₃ ⊕ L₅ ⊕ K₅)</td>
<td>K₅</td>
<td>S₇</td>
</tr>
<tr>
<td>C₄</td>
<td>L₄</td>
<td>α₄₇ = h(K₄ ⊕ L₇ ⊕ K₇)</td>
<td>K₇</td>
<td>–</td>
</tr>
<tr>
<td>C₅</td>
<td>L₅</td>
<td>–</td>
<td>K₅</td>
<td>S₇</td>
</tr>
<tr>
<td>S₇</td>
<td>L₇</td>
<td>α₇₄ = h(K₄ ⊕ L₄ ⊕ K₄)</td>
<td>K₇</td>
<td>–</td>
</tr>
</tbody>
</table>

Next, let us add ({USER 7}, {F₁, F₅}) and ({USER 4}, {F₅}) to the access control table as shown in Table IV. A new row and a new column are added to the control access table to specify the access rights of USER 7 on F₁ and F₅.

Table IV. A USER ACCESS CONTROL TABLE WITH ADDITIONS OF ({USER 7}, {F₁, F₅}) and ({USER 4}, {F₅})

<table>
<thead>
<tr>
<th>USER</th>
<th>F₁</th>
<th>F₂</th>
<th>F₃</th>
<th>F₄</th>
<th>F₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>USER 2</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>USER 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER 4</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>USER 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER 6</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
</tbody>
</table>
Table VI lists $K_i$, $L_i$, $\alpha_{i,j}$, and $S_i$ of all the nodes in the lattice shown in Figure 4. The shaded row in Table VI indicates $C_2$ is no longer considered by CA for access control.

<table>
<thead>
<tr>
<th>Node</th>
<th>$L_i$</th>
<th>$\alpha_{i,j}$</th>
<th>$K_i$</th>
<th>$S_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1$</td>
<td>$L_1$</td>
<td>$\alpha_{1,3} = h(K_1 \oplus L_3)$</td>
<td>$K_1$</td>
<td>$-$</td>
</tr>
<tr>
<td>$C_2$</td>
<td>$L_2$</td>
<td>$-$</td>
<td>$K_2$</td>
<td>$-$</td>
</tr>
<tr>
<td>$C_3$</td>
<td>$L_3$</td>
<td>$\alpha_{3,5} = h(K_3 \oplus L_5)$</td>
<td>$K_3$</td>
<td>$S_7$</td>
</tr>
<tr>
<td>$C_4$</td>
<td>$L_4$</td>
<td>$-$</td>
<td>$K_4$</td>
<td>$-$</td>
</tr>
<tr>
<td>$C_5$</td>
<td>$L_5$</td>
<td>$\alpha_{5,6} = h(K_5 \oplus L_6)$</td>
<td>$K_5$</td>
<td>$-$</td>
</tr>
<tr>
<td>$S_7$</td>
<td>$L_7$</td>
<td>$\alpha_{7,4} = h(K_7 \oplus L_4)$</td>
<td>$K_7$</td>
<td>$-$</td>
</tr>
<tr>
<td>$C_6$</td>
<td>$L_6$</td>
<td>$-$</td>
<td>$K_6$</td>
<td>$S_7$</td>
</tr>
</tbody>
</table>

Next, let us assume USER 6 in $C_5$ is removed from the system. Since USER 6 in $C_5$ no longer needs to access $F_4$, the link between $C_3$ and $C_6$ should be removed from the lattice, thus CA removes $C_3$ and the link $C_3$ to $C_6$ with $\alpha_{3,6}$. After the removal of $C_5$ from $S_7$, $C_6$ becomes a successor of the predecessor of $C_5$ which is $C_1$. Thus, CA creates a new link from $C_1$ to $C_6$ with $\alpha_{1,6}$. In addition, USER 4 in $C_1$ no longer has the precedence over USER 6 in $C_5$, CA removes the link between $C_1$ and $C_5$ with $\alpha_{1,5}$. The resulting lattice is depicted in Figure 5.

With the all scenarios demonstrated above, the final lattice is shown as follows.

Table VI. Computations of Parameters and Keys in Figure 4

You might notice that $S_7$ only contains a node $C_8$ that should be removed from the lattice. The motivation of having an $S$-node in a lattice is that the nodes in the $S$-node are all sharing the successors. Therefore, it might be a great possibility that $F_1$ in $C_4$ would be shared again by new users together with USER 7 later. That is the reason we decided to keep an $S$-node with only one node left in a lattice.

VIII. SUMMARY

We presented an efficient access control scheme in this paper with the following features,

- The scheme must support access control in a lattice represented by partially ordered relationships of users as well as resources.
- The key generation and derivation algorithm is simple and efficient.
- The scheme introduces a new concept of $S$-nodes to the lattice that further improves the performance of key generation and derivation and reduces the storage requirements.
- The scheme is resistant to collusion in that users in a class cannot derive the keys of its predecessors.
- The scheme does not require the entire system to be rekeyed as a user joins or leaves a class. Neither the scheme propagates the rekeying to predecessors or successors.

With the above features, our scheme shows more efficiency and less storage requirements than other schemes. Our scheme uses a hash function and the exclusive-or operation in the key generation and derivation algorithm. Compared to the schemes [1, 8, 10, 13, 24-27] using modular exponentiation, discrete logarithms, polynomial interpolations, and additional usage of encryption/decryption, our scheme is more efficient than these schemes. Comparable to the schemes with reasonable performance [13, 20, 28-29], our scheme uses less amount of storage space for storing public parameters and secret keys. Rekeying is definitely not needed for the entire system. Instead, rekeying only occurs locally. Definitely our schemes perform
more efficiently in rekeying than majorities of schemes [1, 12, 15, 20, 30].

ACKNOWLEDGMENT

The work was supported in part by the Department of Defense (DOD) under Award No. H98230-07-C-0403.

REFERENCES


