International Journal of Computers and Applications, Vol. 30, No. 4, 2008

A NEW CLASS OF KEY MANAGEMENT SCHEME FOR ACCESS CONTROL IN DYNAMIC HIERARCHIES

X. Zou* and L. Bai**

Abstract

Cryptographic techniques for *Hierarchical Access Control* (HAC) have recently attracted intensive research interests. A large number of key management schemes have been proposed for access control in hierarchy. Their constructions are based mainly on one-way functions and are computationally secure. Many of these schemes were however, found with different problems in terms of structure, security, efficiency and dynamics. In this paper, we propose a new key management scheme that can be used for access control in dynamic hierarchies. The new scheme is based on the secret sharing principle, without using a one-way function. Consequently, it is unconditionally secure. In addition, it has the following properties: (i) supporting full dynamics at both node and user levels; (ii) allowing any random reconfiguration of access hierarchy; (iii) utilizing the same algorithm for key computation and also descendant key derivation (regardless how far its descendant is away from the node); and (iv) eliminating performance bottlenecks at the trusted *Central Authority* (CA).

Key Words

Information security, information theory, secret sharing, hierarchical access control (HAC), symmetric polynomial, key management

1. Introduction

Access control is used to verify whether a user has rights to access certain resources and to grant or deny access to the resources requested [1]. Access control is a fundamental problem in secure computing or communication systems involved with resource sharing. Generally speaking, many systems require hierarchical access control in various levels or different users. For example, resources are assigned in different levels and users can have different privileges to access these resources. A user, accessing a resource at a higher level, is automatically granted access to his or her corresponding children's or descendants' resources. This kind of access control is referred to as hierarchical access control (HAC). An HAC structure can be represented by a *Directed Acyclic Graph* (DAG). HAC has broad applications in operating systems, databases, and networking, and is also useful in many other applications, such as classified electronic content distribution and project management. One of the most widely used HAC technologies is the *Role-based access control* model [2, 3].

More recently, another class of HAC techniques, referred to as cryptographic HAC (CHAC) solutions, has attracted attention from many researchers. CHAC provides a better method, when compared to traditional HAC techniques. It assumes that every node in an access hierarchy can be assigned with a (cryptographic) key. Any ancestor can use its key to compute its descendants' keys, but a descendant cannot use its key to determine its ancestors' keys. An access hierarchy must be flexible to add new nodes, delete existing nodes, move nodes around, merge several sets of nodes, or split nodes into multiple sets. Moreover, users/resources are allowed to be added to, deleted from, and moved around the nodes in the access hierarchy. Whenever such an update occurs, the keys of the nodes will be changed correspondingly. We can also periodically refresh keys in HAC to prevent keys from being exposed.

The first CHAC solution was proposed by Akl *et al.* [4, 5] in 1983. Since then, there have been many CHAC schemes proposed [6-16]. These schemes basically rely on a one-way function so that a node, v, can easily compute v's descendants' keys, whereas v's key is computationally difficult to compute by v's descendant nodes. Many existing schemes have some of the following problems: (1) some schemes were found with security flaws shortly after they were proposed; (2) some schemes cannot support the reconfiguration of a hierarchy; (3) some schemes require the access hierarchy to be in a certain form, so that it must be a tree or a DAG with only one root; and (4) member revocation is one of the most difficult processes in cryptographic schemes, therefore, it is important to address this problem so that the revocation process can be dealt with efficiently.

In this paper, we propose a new CHAC scheme based

* Department of Computer and Information Science, Indiana University-Purdue University Indianapolis, Indianapolis, IN 46202, USA; e-mail: xkzou@cs.iupui.edu

** Department of Electrical and Computer Engineering, Temple University, Philadelphia, PA 19122, USA; e-mail: lbai@temple.edu

commended by Dr. Hai Jin
per no. 202-2419)

on symmetric polynomials. Unlike many existing schemes, which are based on one-way functions, our scheme is based on a secret sharing method which makes the scheme unconditionally secure[1] (also see the discussion in [17, 18]). As well, CHAC requires two types of key operations: (1) key computation – a node, v, computes its own key: (2) key derivation – a node, v, computes its descendants' keys.

In most existing schemes, key derivation is different from key computation. Key derivation requires iterative computation of keys for nodes along the path from a node to its descendant, which is inefficient if the path is long. In our scheme, both operations are made by substituting (different) parameters in the same polynomial function assigned to node v. Thus, the key derivation efficiency can be improved. Our scheme also supports full dynamics at both node and user levels and permits any random access hierarchies. More importantly, removing nodes and/or users is an operation as simple as adding nodes and/or users in the hierarchy. A trusted Central Authority (CA) can assign secrets (polynomials) to corresponding nodes so that nodes can compute their keys. As well, nodes can derive their descendants' keys, without involvement of the CA, as soon as polynomial functions were distributed to them. In addition, the storage requirement and computation complexity at the CA are almost the same as that at individual nodes, thus, the CA would not produce a performance bottleneck and can deal with dynamic operations efficiently.

The rest of the paper is organized as follows. We briefly discuss the related research works in section two. Section three describes our proposed scheme. Specifically, we discuss its principle in dynamic operations, and analyze its security and efficiency. We discuss its efficiency issue and suggest a possible extension of symmetric polynomials to general (asymmetric) polynomials in section four. Section five is given for the conclusion and future work.

2. Related Work

In this section, we summarize and classify typical CHAC schemes.

It is generally assumed, in CHAC schemes, that there is a central authority (CA) which manages the hierarchy and generates and distributes keys. The scheme assumes there is only one root class, c_0, which is the ancestor of all other n classes c_i, where $i = 1, 2, \ldots, n$. A CA selects two large prime numbers, p and q, and sets $M = pq$. In addition, the CA chooses $n+1$ more prime numbers, p_i, which are assigned to classes c_i for $i = 0, 1, 2, \ldots, n$. The CA assigns the root class c_0, with integer $t_0 = 1$, and selects a large random number, k_0, which is assigned to c_0 as a master key. Then the CA calculates the following numbers for classes c_i, where $i = 1, 2, \ldots, n$:

$$t_i = \prod_{c_i \not\leq c_i} p_j$$

$$k_i = k_0^{t_i} \bmod M$$

where t_i $(i = 0, 1, 2, \ldots, n)$ and M are made as public information and k_i are keys distributed to c_i $(i = 0, 1, 2, \ldots, n)$. The scheme ensures that a class, c_j, can compute another class c_i's key, if $c_i \leq c_j$, by calculating

$$k_i = k_j^{t_i/t_j} \bmod M$$

However, class c_i cannot compute class c_j's key, k_j, because the RSA-like one-way function used here prevents k_j from being calculated. In addition, the function also prevents any nodes from computing the keys of their ancestors by collusion. Consequently, the HAC policy is enforced.

The above scheme, which was proposed by Akl and Taylor [4], is the first CHAC solution for enforcing access control in an HAC. Unfortunately, one limitation was its complex operation for dynamic reconfiguration when nodes are added or removed from the hierarchy. When node membership changes, keys have to be updated accordingly to prevent problems of backward and forward secrecy. Concepts of backward secrecy and forward secrecy are introduced in secure group communication systems [18]: (1) backward secrecy means that a newly joined user cannot obtain the previous keys to decrypt earlier encrypted messages. For example, a newly joined user becomes a parent node in which he or she should not be able to derive his or her children's previous keys to decrypt any previous encrypted messages that his or her children were granted access rights; (2) forward secrecy means that a revoked or disassociated member should not be able to obtain his or her children's keys in the new hierarchy. It is important in CHAC policy to prevent this member from continually deciphering future group communication after he or she has left the group.

Another drawback is that Akl and Taylor's scheme was not directly applicable to the hierarchy with multiple roots. One way to mitigate this problem is to introduce a new fake root without any user associated and to link this root to all original roots. Unfortunately, in this system dynamic operations still cannot be solved. To address both multiple root and dynamics problems, several approaches [5, 7] were proposed by reassigning p_i differently.

Following Akl and Taylor's seminal work, researchers proposed many other CHAC schemes [12, 13, 15, 16, 18, 19]. Most of them are constructed similarly to Akl and Taylor's scheme, by using one-way functions. Some one-way function based schemes have been found insecure. For example, two such schemes, proposed in [20], have the same collusion problem, as pointed out by Yi in [21]. Besides one-way function based CHAC schemes, there are several interesting CHAC schemes [14, 22] constructed by using polynomials and interpolations. Unfortunately, these two schemes were demonstrated to be vulnerable for possible exploitations, initially by Hsu and Wu [23]. Consider the case that two nodes, u and v, have a common child

[1] A cryptosystem is said to be computationally secure if the best algorithm for breaking it requires at least N operations, where N is some specified, very large number. On the other hand, a cryptosystem is said to be unconditionally secure if it cannot be broken by attackers, even if the attackers collude and have infinite computational resources. Unconditional security implies that when an opponent does not know the secret key, k, then k appears to him as a random element in the key space. That is, the opponent has no way of distinguishing the key, k, from any other element in the key space.

node, a, in the access hierarchy. Node v has another node, b. Although node b is not a child node of u, node u can indirectly derive node b's key by traversing through nodes a and v. This exploitation was addressed and eliminated, by Hsu and Wu [23], by introducing and using a one-way function. Interestingly, the modified scheme does not prevent other exploitations from being discovered. Wang and Laih [24] found an exploitation method and then proposed an improved scheme so that it becomes more secure and resilient against possibilities of being compromised. Nonetheless, another improved CHAC scheme [8] was also proposed recently to address similar exploitation problems.

The one-way function based CHAC schemes are typically classified in two categories [18]: (i) directly dependent key schemes and (ii) indirectly dependent key schemes. Category (i) schemes [5, 7, 13, 18, 25] generate classes' keys from some preselected parameters. Any class key can be directly derived from its parent's key by using a one-way function (and some public information), but a class cannot derive its parent's key from the same one-way function. As can be seen, Akl-Taylor's scheme belongs to this category. Category (ii) schemes [11, 12, 15, 16, 19] generate some public known parameters from preselected classes' keys by using a one-way function. Any parent node can use these public, known parameters with its own key to derive its descendants' keys, but a descendant cannot compute any key of its ancestors. Lin's scheme [11] is an early example of this category. Recently, Atallah *et al.* [19] proposed a CHAC scheme that is very efficient. The scheme elegantly uses the one-way function twice, resulting in localization, an important property by which only values of local nodes/edges are affected when nodes/edges are added or deleted.

3. A Symmetric Polynomial based HAC Scheme

In this section, we propose a new CHAC scheme based on symmetric polynomial functions. The underlying principle of the new scheme is that of secret sharing. As a result, unlike the existing CHAC schemes, which are computationally secure, the new proposed scheme is unconditionally secure.

3.1 Principle

Assume a set of n classes, $\{C_1, C_2, \ldots, C_n\}$, has an ancestral classes set, as $\{S_1, S_2, \ldots, S_n\}$, where $m_i = |S_i|$ is the number of ancestral classes of a class, C_i, for $i = 1, 2, \ldots, n$ and $0 \le m_i \le n-1$. We need to choose another number, m, such that $m \ge \max\{m_1, m_2, \ldots, m_n\} + 1$. m is the number of variables/parameters in a polynomial function used to construct our CHAC scheme.

In the example shown in Fig. 1 we have nine classes, $C_1, C_2, \ldots, C_9\}$, and their ancestral sets, $\{S_1 = \{\emptyset\}, S_2 = \{\emptyset\}, S_3 = \{C_1, C_2\}, S_4 = \{C_2\}, S_5 = \{C_2\}, S_6 = \{C_1, C_2, C_3\}, S_7 = \{C_1, C_2, C_3, C_4\}, S_8 = \{C_2, C_4, C_5\}, S_9 = \{C_2, C_5\}\}$. As a result, we need to choose $m \ge \max\{m_1, m_2, \ldots, m_9\} + 1 = 5$. Suppose we let $m = 7$, which will allow us to expand the hierarchy without changing m value, if we want to add more new nodes or new links. A larger m value is better, because we need not change the polynomial function when more nodes are added until the m value becomes less than the maximum number in the ancestral classes number set.

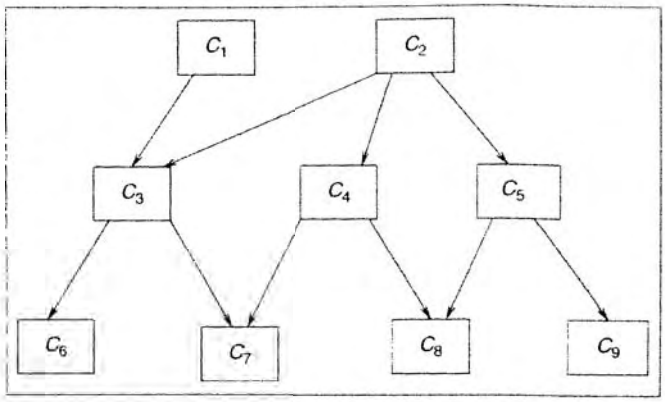

Figure 1. A typical hierarchy.

A CA selects a large positive integer, p, as the system modulus (p need not be a prime)[2] and a threshold number, t, so that fewer than $t+1$ users cannot collaborate to disclose their ancestors' keys. Then, the CA can randomly generate a symmetric polynomial[3] in m variables with coefficients from $\mathbb{Z}_p$ in which the degree of any variables is at most t as

$$P(x_1, x_2, \ldots, x_m) = \sum_{i_1=0}^{t} \sum_{i_2=0}^{t} \cdots \sum_{i_m=0}^{t} a_{i_1, i_2, \ldots, i_m} x_1^{i_1} x_2^{i_2} \cdots x_m^{i_m} \pmod{p}$$

where $a_{i_1, i_2, \ldots, i_m}$ are $\binom{m+t}{m}$ coefficients randomly generated by the CA. The polynomial function $P(x_1, x_2, \ldots, x_m)$ is kept as a secret by the CA. Every class in the hierarchy has a polynomial function, which is derived from $P(x_1, x_2, \ldots, x_m)$, and the polynomial function is transmitted[4] to each class securely by the CA.

To derive proper keys in the hierarchy, the CA generates some publicly known numbers: (i) n random numbers, s_i, associated with C_i for $i = 1, 2, \ldots, n$, and (ii) $(m-1)$ additional random numbers, s'_j, for $j = 1, 2, \ldots, m-1$ (note: s_i and s'_j belong to $\mathbb{Z}_p$). For each class, C_i, with an ancestor set, $S_i = \{C_{i_1}, C_{i_2}, \ldots, C_{i_{m_i}}\}$, where i_j is an ordinal number that $1 \le i_j \ne i \le n$, class C_i is given a polynomial function $g_i(x_{m_i+2}, x_{m_i+3}, \ldots, x_m)$ which can be derived from $P(x_1, x_2, \ldots, x_m)$ by the CA as

$$g_i(x_{m_i+2}, x_{m_i+3}, \ldots, x_m) = P(s_i, s_{i_1}, s_{i_2}, \ldots, s_{i_{m_i}}, x_{m_i+2}, x_{m_i+3}, \ldots, x_m)$$

[2] This system is unlike public key cryptosystems which require numbers of at least 1024 bits. Instead, it is equavalent to the secret cryptosystems in terms of security strength. Thus, a p of 128 or more bits is large enough.

[3] P is called a symmetric polynomial if $a_{i_1, \ldots, i_m} = a_{\pi(i_1), \ldots, \pi(i_m)}$ for all permutations π of $\{1, \ldots, m\}$.

[4] Transmission of a polynomial means to transmit its coefficients and, thus, the communication cost is of $O(mlog(p))$ bits.

and class C_i's key, k_i, becomes

$$k_i = g_i(s'_1, s'_2, \ldots, s'_{m-m_i-1}) = P(s_i, s_{i_1}, s_{i_2}, \ldots, s_{i_{m_i}}, s'_1, s'_2, \ldots, s'_{m-m_i-1})$$

Class C_i can also use the same polynomial function, $g_i(x_{m_i+2}, x_{m_i+3}, \ldots, x_m)$, to compute its descendants' keys. Supposing C_j is one of the descendants for class C_i, we denote a new ancestral set as $S_{j\backslash i}$ which is defined as

$$S_{j\backslash i} \triangleq S_j \backslash (S_i \cup \{C_i\}) = \{C_{(j\backslash i)_1}, C_{(j\backslash i)_2}, \ldots, C_{(j\backslash i)_{r_j}}\}$$

where $r_j = |S_{j\backslash i}|$ and $(j\backslash i)_l$ is an ordinal number that $1 \le (j\backslash i)_l \le n$ for $l = 1, 2, \ldots, r_j$. Precisely speaking, set $S_{j\backslash i}$ is a collection of ancestors for class C_j, but excluding C_i and those classes which are ancestors of both classes C_i and C_j. As a result, key k_j can be calculated by class C_i as

$$\begin{aligned} k_j &= g_i(s_j, s_{(j\backslash i)_1}, s_{(j\backslash i)_2}, \ldots, s_{(j\backslash i)_{r_j}}, s'_1, s'_2, \ldots, s'_{m-m_i-2-r_j}) \\ &= P(s_i, s_j, s_{i_1}, s_{i_2}, \ldots, s_{i_{m_i}}, s_{(j\backslash i)_1}, s_{(j\backslash i)_2}, \ldots, s_{(j\backslash i)_{r_j}}, s'_1, s'_2, \ldots, s'_{m-m_i-2-r_j}) \end{aligned}$$

As s_i and s'_j are publicly known, class C_i can compute its key and its descendants' keys, but not its ancestors' keys, by using the polynomial function $g_i(x_{m_i+2}, x_{m_i+3}, \ldots x_m)$ assigned to class C_i.

Let us use the same example shown in Fig. 1, where $m = 7$. The CA randomly generates a polynomial function, $P(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$, with seven parameters. The CA can then compute nine polynomial functions for classes C_i from the symmetric polynomial function $P(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$. Once polynomial functions are obtained, they are transmitted securely to every class C_i respectively. For instance, class C_3 will have a polynomial function $g_3(x_4, x_5, x_6, x_7)$ as

$$g_3(x_4, x_5, x_6, x_7) = P(s_3, s_1, s_2, x_4, x_5, x_6, x_7)$$

because $S_3 = \{C_1, C_2\}$ and their associated numbers are s_1 and s_2. As well, another form of the polynomial function is

$$g_3(x_4, x_5, x_6, x_7) = P(s_1, s_2, s_3, x_4, x_5, x_6, x_7)$$

because the symmetric polynomial function gives the same result no matter what order of s_i is used. Consequently, key k_3 is computed as

$$k_3 = g_3(s'_1, s'_2, s'_3, s'_4) = P(s_1, s_2, s_3, s'_1, s'_2, s'_3, s'_4)$$

We can compute another polynomial function for class C_7 as

$$g_7(x_6, x_7) = P(s_1, s_2, s_3, s_4, s_7, x_6, x_7)$$

because $S_7 = \{C_1, C_2, C_3, C_4\}$ and their associated numbers are s_1, s_2, s_3 and s_4. Key k_7 is computed as

$$k_7 = g_7(s'_1, s'_2) = P(s_1, s_2, s_3, s_4, s_7, s'_1, s'_2)$$

Clearly, class C_7 cannot determine its parent C_3's key, but class C_3 can compute its descendant C_7's key. As we know that

$$S_{7\backslash 3} = \{C_4\}$$

because classes C_1 and C_2 are common ancestors needed to be excluded from set $S_{7\backslash 3}$. Class C_3 can compute key k_7 as

$$\begin{aligned} k_7 = g_3(s_7, s_4, s'_1, s'_2) &= P(s_1, s_2, s_3, s_4, s_7, s'_1, s'_2) \\ &= g_7(s'_1, s'_2) \end{aligned}$$

We can see that class C_i performs key computation and key derivation in the exact same way. No matter where C_i's descendant is, the key derivation process is computed by using the same polynomial function, g_i, with different s_i and s'_i values substituted. In contrast, most existing schemes require iterative computation of keys along the path from a node to its descendant for key derivation. This can be a potential efficiency advantange for our new scheme.

Our proposed scheme supports flexible dynamic operations, such as adding, deleting, moving, merging, and splitting classes as well as adding and deleting links, at both class and user/resource levels. Moreover, a user/resource can be also added to a class, be removed from a class, and be moved from one class to another. We describe them in the following subsections.

3.2 Class Level Dynamics

Our scheme is flexible for dynamic class operations. Specifically,

- Adding a class – When a new class, C_r, is added, we need to verify whether the m value satisfies the new node constraint. (1) If $m < \max\{m_1, m_2, \ldots, m_n, m_r\} + 1$, a new m value will be generated so that $m \ge \max\{m_1, m_2, \ldots, m_n, m_r\} + 1$. As well, the CA will regenerate a new polynomial function, $P(x_1, x_2, \ldots, x_m)$, accordingly. In addition, all polynomial functions of classes are recomputed and retransmitted securely. (2) If $m \ge \max\{m_1, m_2, \ldots, m_n, m_r\} + 1$, the CA selects a random number, s_r, for the new class, C_r, so that a new polynomial function, g_r, can be computed and transmitted to class C_r securely. However, if class C_r is added as a parent class of any existing classes, we need to modify the keys of C_r's descendant classes to prevent class C_r from obtaining old keys of its descendant. (Recall that it is an issue for backward secrecy.) Therefore, the CA uses S_r to compute and transmit new polynomial functions of all C_r's descendant classes securely.
  In this process, we can see that we do not need to change the keys of any C_r's ancestral classes, nor to change the keys of any classes who are not related to class C_r. It

implies that our scheme can become efficient in some class adding operations.

- Deleting a class – When a class, C_r, is removed from the hierarchy, we need to determine whether the class C_r is a leaf node or a parent node. Here, a leaf node is defined as a node without any descendant. (1) if class C_r is a leaf node the CA can simply discard the public parameter, s_r, without changing any other keys (2) if class C_r is a parent node once class C_r is deleted from the hierarchy, we cannot allow it to compute keys of C_r's descendant classes using polynomial function g_r. We need to prevent class C_r from accessing its descendants' resources (recall it is an issue for forward secrecy). The solution is that the CA recomputes new polynomial functions of C_r's older descendants according to the newer hierarchy without substituting s_r.
  Again, our scheme only affects the keys of the classes that need to be changed, which implies that our scheme is efficient in some class deleting operations.
- Moving a class – A class, C_r, can be moved from one node to another node in the hierarchy. There are four cases: (1) when moving from a leaf node to another leaf node, the CA simply recomputes the polynomial function, g_r, according the new hierarchy and securely transmits g_r to C_r; (2) when moving from a leaf node to a parent node, the CA recomputes the polynomial functions of class C_r and C_r's new descendant classes according to the new hierarchy and the CA securely transmits polynomial functions to the affected classes; (3) when moving from a parent node to a leaf node, the CA recomputes polynomial functions of previous descendant classes of C_r and class C_r according to the new hierarchy and then securely transmits these polynomial functions to the affected classes; (4) when moving from a parent node to a parent node, the CA recomputes the polynomial functions of previous and present descendant classes of C_r and class C_r according to the new hierarchy and then securely transmits these polynomial functions to the affected classes.
- Merging classes – Two or more classes can merge and become one class, C_r. The CA needs to find previous and present descendant classes of the merging classes. The CA randomly chooses a new number, s_r, and then generates polynomial functions for all corresponding classes.
- Splitting a class – A class, C_r, splits into two classes C_{r_1} and C_{r_2}. Depending on whether C_r is a parent node or leaf node, the CA has to determine what previous and present descendant classes are associated with these classes (C_r, C_{r_1} and C_{r_2}). The CA then selects two new numbers, s_{j_1} and s_{j_2}, and generates polynomial functions for these affected classes.
- Adding a link – If two classes, C_r and C_k, are linked, we establish a new direct parent-child relationship between two classes. Assume class C_r is the parent of class C_k. There are two different cases. (1) Class C_r was an ancestor of class C_k, through other classes. The CA does not need to perform any actions. (2) Class C_r is the only parent for class C_k in the new hierarchy. The CA selects a new number, s_k, and generates new polynomial functions for class C_k and its descendants classes. The CA securely transmits new polynomial functions to these affected classes.
  It is also worth while to note that adding a link can cause the number of ancestors of a node to exceed the current m. Should this happens, a new m needs to be determined and a new polynomial $P(x_1, x_2, \ldots, x_m)$, to be generated and class polynomial functions to be computed and distributed.
- Deleting a link – If two linked classes, C_r and C_k, are disconnected (or the link is deleted), we destroy a direct parent-child relationship between two classes. Say class C_r will not be the parent of class C_k in the new hierarchy. Again, there are two different cases. (1) Class C_r is still an ancestor of class C_k, through other classes in the new hierarchy. The CA does not need to perform any actions. (2) Class C_r is not an ancestor for class C_k in the new hierarchy. The CA selects a new number s_k and generates new polynomial functions for class C_k and its descendants classes. The CA securely transmits new polynomial functions to these affected classes.

3.3 User Level Dynamics

In a CHAC scheme every class represents certain access privileges. A group of users in a class can share a key if they belong to the same class. For example, all users in class C_j can compute the keys of class C_j and its descendant classes. Dynamic user operations deal with how a user can join in a class or leave a class and possible displacement from one class to a different class. They all require the class key to be changed after any user operation is completed so that the issues of backward secrecy and forward secrecy can be addressed.

Specifically, our scheme can revoke a user from a class C_j. It is as quick and efficient as to join a user in the class C_j. Both operations require that the CA randomly select a new public parameter, s_j, for C_j and recompute a new polynomial function, g_j, by using the new s_j. As the polynomial function g_j is newly produced, other polynomial functions and keys are also recomputed for the descendant classes of C_j. This will guarantee both backward secrecy and forward secrecy. The efficiency can be improved if backward secrecy or forward secrecy is not required.

Another common user operation is to allow a user to move from one class, C_j, to another class, C_k. Here, the CA will randomly choose two new public parameters, s_j and s_k, for C_j and C_k so that new polynomial functions and keys are recomputed and transmitted to C_j, C_k, and their descendants respectively. Thus, both backward secrecy and forward secrecy are guaranteed.

3.4 Key Refresh

The new scheme is easy to refresh keys so that the keys can be better protected. To accomplish a key refresh operation the CA can either change s_j, for a particular class C_j if it asks for key refreshing, or change s_j for all classes in the hierarchy.

3.5 Security and Efficiency Analysis

Evidently, the construction of our scheme is dependent on polynomial function $P(x_1, x_2, \ldots, x_m)$, which is a function of m and t (but independent from n, the total number of nodes in the access hierarchy). As a result, these values have direct impacts on our scheme's security and efficiency.

- As our proposed scheme is developed by using the same principle as many other secret sharing schemes [26–28], the security of secret sharing schemes is dependent on t. Blundo *et al.* [26] proved that these schemes are unconditionally secure to t users' collusion attacks. In other words, when t or fewer users, no matter which classes they come from, collaborate, they cannot reveal any bit information of polynomial function $P(x_1, x_2, \ldots, x_m)$ or their ancestral classes' keys. However, more than t users can not only reconstruct their parents' keys, but also completely determine polynomial function $P(x_1, x_2, \ldots, x_m)$ so that all keys can be exposed. In summary, the scheme is unconditionally secure and is capable of defending against up to t users' collusion attacks.
- Efficiency is measured in two ways: (i) storage space for polynomial functions and (ii) computation complexity for determining keys from a polynomial function. Specifically, the CA needs to store $\binom{m+t}{m}$ coefficients[5] because the CA generates a polynomial function, $P(x_1, x_2, \ldots, x_m)$, with $\binom{m+t}{m}$ coefficients. The CA can also use $P(x_1, x_2, \ldots, x_m)$ to generate other polynomial functions and keys of classes so that computation complexity is also dependent on the polynomial function $P(x_1, x_2, \ldots, x_m)$. As the power of polynomial $P(x_1, x_2, \ldots, x_m)$ is t, the computation complexity for the CA to compute each polynomial g_i is $O((t+1)^m)$. For individual classes C_i, they also require storage space for their assigned polynomial function g_i. The storage space of class C_i is $\binom{m+t-1}{m-1}$ because the number of coefficients of g_i is $\binom{m+t-1}{m-1}$. Similarly, the computation complexity for C_i to compute k_i or derive their descendant keys is $O(t^{m-1})$.

In summary, we can observe three key points relating to the efficiency of the new proposed scheme. They are: (1) the CA and any class/user have almost the same storage requirement and computation complexity, which implies that the CA may not become a performance bottleneck; (2) key computation and key derivation are the same operation, which implies that a class can derive or compute keys in the same efficiency no matter where the class is in the hierarchy; and (3) the storage and time complexities are mainly dependent on m, which implies that our scheme can support many classes as long as m is small. Conceptually, the hierarchy is efficient if it is "fat" (the hierarchy is spanned in the horizontal direction).

4. Asymmetric Polynomials for Improving Efficiency

The symmetric polynomial based scheme has the advantage that s_i and s'_j values can be substituted into the polynomial in any order. For example, $P(s_1, s_3, s'_1) = P(s_3, s'_1, s_1)$. It is beneficial for a class to compute its descendants' keys without worrying about how to substitute parameters in an order according to where its descendants are resided in the hierarchy. A problem associated with the scheme is high computation complexity introduced by symmetric polynomial functions. We showed that our scheme has a high computation complexity, in the order of $O(t^{m-1})$, and a large storage requirement, used to store $\binom{m+t-1}{m-1}$ coefficients. If resources are constrained or expensive, our scheme works efficiently for a hierarchy with "fatter" or "shorter" structure (m is small). To apply our scheme in a "thin" or "taller" HAC structure, where m could be large, it would be difficult and inefficient to use symmetric polynomial functions. Instead, the CA can use an asymmetric polynomial function to generate keys for classes in the hierarchy. For example, the asymmetric polynomial may be selected as:

$$P(x_1, x_2, \ldots, x_m) = \sum_{i_1+i_2+\cdots+i_m=t} a_{i_1 i_2 \cdots i_m} x_1^{i_1} x_2^{i_2} \cdots x_m^{i_m}$$

If so, the keys and polynomial functions have to be computed with s_i and s'_j being substituted in a certain order according to the hierarchy. Similarly, a class C_i and its parent class C_j have to compute k_i by substituting s_i and s'_i into their corresponding polynomial functions following the same order. The order can be determined in the same order as the topology-sort. When s_i and s'_i are substituted in polynomial functions, they are both used in an ascending order. How to design such asymmetric polynomials, so that both efficiency and security can be maintained is an interesting and challenging task to pursue.

5. Conclusion

In the paper, we proposed a new CHAC scheme for dynamic access hierarchies based on the secret sharing principle. The new scheme exhibits the possibility of new class CHAC schemes, different from the traditional CHAC class which utilizes a one-way function. The new scheme is unconditionally secure (up to t user collusion) and elegant for its mechanism of computing keys, deriving keys, and enforcing correct access control. We also discussed its possible extension as a more computationally efficient algorithm by using a random and non-symmetric polynomial function rather than a strictly "symmetric polynomial" function. We will continue this challenging work. To further explore the potential applications of the newly proposed scheme, we plan to implement a software suite of CHAC protocols incorporating various CHAC schemes with this newly proposed scheme, so that we can compare their performance in real environments. Due to the space limitation, we did not discuss in detail its practical performance and omitted its performace comparison with other schemes. We will address all these issues in future work.

Acknowledgement

This work was partially supported by the U.S. NSF grant CCR-0311577.

[5] The CA also needs to store the hierarchy which is $O(n)$.

References

[1] A. Silberschatz, P.B. Galvin, & G. Gagne, *Operating system concepts*, Sixth Edition (Hoboken, NJ: John Wiley & Sons, Inc., 2001).

[2] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, & C.E. Youman, Role-based access control models, *IEEE Computer*, *29*(2), 1996, 38–47.

[3] H. Wang, J. Cao, & Y. Zhang, A flexible payment scheme and its role-based access control, *IEEE Transactions on Knowledge and Data Engineering*, *17*(3), 2005, 425–436.

[4] S.G. Akl & P.D. Taylor, A cryptographic solution to the problem of access control in a hierarchy, *ACM Transactions on Computer Systems (TOCS)*, *1*(3), 1983, 239–248.

[5] 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*, *34*(9), 1985, 797–802.

[6] T.-S. Chen, Y.-F. Chung, & C.-S. Tian, A novel key management scheme for dynamic access control in a user hierarchy, in *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC)*, Hong Kong, China, 2004, 396–401.

[7] G.C. Chick & S.E. Tavares, Flexible access control with master keys, *Proceedings on Advances in Cryptology: CRYPTO '89, LNCS*. Santa Barbara, CA, USA, 1989, 316–322.

[8] M.L. Das, A. Saxena, V.P. Gulati, & D.B. Phatak, Hierarchical key management scheme using polynomial interpolation, *SIGOPS Operating Systems Review*, *39*(1), 2005, 40–47.

[9] L. Harn & H.Y. Lin, A cryptographic key generation scheme for multilevel data security, *Computers and Security*, *9*(6), 1990, 539–546.

[10] M.-S. Hwang, C.-H. Liu, & J.-W. Lo, An efficient key assignment for access control in large partially ordered hierarchy, *Journal of Systems and Software*, *67*(2), 2004, 99–107.

[11] C.H. Lin, Dynamic key management scheme for access control in a hierarchy, *Computer Communications*, *20*(15), 1997, 1381–1385.

[12] I. Ray, I. Ray, & N. Narasimhamurthi, A cryptographic solution to implement access control in a hierarchy and more, *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, Monterey, CA, USA, 2002, 65–73.

[13] R.S. Sandhu, Cryptographic implementation of a tree hierarchy for access control, *Information Processing Letter*, *27*(2), 1998, 95–98.

[14] V.R.L. Shen & T.-S. Chen, A novel key management scheme based on discrete logarithms and polynomial interpolations, *Computers and Security*, *21*(2), 2002, 164–171.

[15] S. Zhong, A practical key management scheme for access control in a user hierarchy, *Computers and Security*, *21*(8), 2002, 750–759.

[16] X. Zou, B. Ramamurthy, & S. Magliveras, Chinese remainder theorem based hierarchical access control for secure group communications, *Lecture Notes in Computer Science (LNCS)*, 2229, 2001, 381–385.

[17] D.R. Stinson, *Cryptography: Theory and practice*, Third Edition (Boca Raton, USA: CRC Press, 2005).

[18] X. Zou, B. Ramamurthy, & S.S. Magliveras, editors, *Secure group communications over data networks* (New York, NY, USA: Springer, 2004).

[19] M.J. Atallah, K.B. Frikken, & M. Blanton, Dynamic and efficient key management for access hierarchies, *ACM CCS'05*, Alexandria, VA, USA, 2005, 190–202.

[20] H. Chien, An efficient time-bound hierarchical key assignment scheme, *IEEE Transactions on Knowledge and Data Engineering*, *16*(10), 2004, 1301–1304.

[21] X. Yi, Security of chien's efficient time-bound hierarchical key assignment scheme, *IEEE Transactions on Knowledge and Data Engineering*, *17*(9), 2005, 1298–1299.

[22] T. Wu & C. Chang, Cryptographic key assignment scheme for hierarchical access control, *International Journal of Computer Systems Science and Engineering*, *16*(1), 2001, 25–28.

[23] C.L. Hsu & Z.S. Wu, Cryptanalyses and improvements of two cryptographic key assignment schemes for dynamic access control in a user hierarchy, *Computers and Security*, *22*(5), 2003, 453–456.

[24] S.Y. Wang & C.S. Laih, Cryptanalyses of two key assignment schemes based on polynomial interpolations, *Computers and Security*, *24*, 2005, 134–138.

[25] N.Y. Lee & T. Hwang, Comments on dynamic key management schemes for access control in a hierarchy, *Computer Communication*, *22*(1), 1999, 87–89.

[26] C. Blundo, L.A.F. Mattos, & D.R. Stinson, Generalied beimelchor scheme for broadcast encryption and interactive key distribution, *Theoretical Computer Science*, *200*, 1998, 313–334.

[27] A.D. Santis, A.L. Ferrara, & B. Masucci, Unconditionally secure key assignment schemes, *Discrete Applied Mathematics*, *154*(2), 2006, 234–252.

[28] A. Shamir, How to share a secret, *Communications of the ACM*, *22*(1), 1979, 612–613.

Biographies

Xukai Zou is an Assistant Professor in the Computer Science Department, Indiana University-Purdue University, Indianapolis. He received his B.S. degree (1983) from Zhengzhou University, China; his M.S. degree (1986) from Huazhong University of Science and Technology, China; and Ph.D. degree (2000) from University of Nebraska-Lincoln, USA. His current research focusses on communication networks, applied cryptography and network security. Dr. Zou has published more than 30 articles in security-related topics and, as the leading author, the books *Secure Group Communication over Data Networks* (Springer, 2004) and *Trust and Security in Collaborative Computing* (World Scientific, 2007). He is a recipient of the U.S. National Science Foundation Cyber Trust Award and has served as a Program Chair, a Member of Technical Program Committees, an Editorial-Board Member, and a Reviewer for a number of international journals and conferences.

Li Bai is an Assistant Professor in the Electrical and Computer Engineering Department at Temple University. He received his B.S. (1996) degree in electrical engineering at Temple University, M.S. (1998) and Ph.D. (2001) degrees in electrical and computer engineering from Drexel University. His research interest includes reliability, dependable secure computing, and wireless sensor network. Dr. Bai has published more than 20 articles in these research areas. He has served as a Program Chair, a Member of Technical program Committees, and a Reviewer for a number of international journals and conferences. Dr. Bai also served as the Chair of IEEE Philadelphia section in 2007.