INTERNATIONAL JOURNAL OF NETWORK MANAGEMENT
Int. J. Network Mgmt 2007; 17: 437–450
Published online 11 May 2007 in Wiley InterScience
(www.interscience.wiley.com) DOI: 10.1002/nem.656

A dynamic key management solution to access hierarchy

Xukai Zou*,†,[1], Yogesh Karandikar[1] and Elisa Bertino[2]

[1]*Department of Computer Science, Indiana University-Purdue University Indianapolis, IN 46202, USA*
[2]*CERIAS and Department of Computer Science, Purdue University, West Lafayette, IN 47907, USA*

SUMMARY

Hierarchical access control (HAC) has been a fundamental problem in computer and network systems. Since Akl and Taylor proposed the first HAC scheme based on number theory in 1983, cryptographic key management techniques for HAC have appeared as a new and promising class of solutions to the HAC problem. Many cryptographic HAC schemes have been proposed in the past two decades. One common feature associated with these schemes is that they basically limited dynamic operations at the node level. In this paper, by introducing the innovative concept of 'access polynomial' and representing a key value as the sum of two polynomials in a finite field, we propose a new key management scheme for dynamic access hierarchy. The newly proposed scheme supports full dynamics at both the node level and user level in a uniform yet efficient manner. Furthermore, the new scheme allows access hierarchy to be a random structure and can be flexibly adapted to many other access models such as 'transfer down' and 'depth-limited transfer'. Copyright © 2007 John Wiley & Sons, Ltd.

1. INTRODUCTION

The term *access control* means controlling access to certain resources. It involves granting or preventing access to resources as required [1]. It is a fundamental security issue in any computer or communication system, where there are users and resources. In most systems, resources are organized in a hierarchy of classes.[1] A typical hierarchy has a partial order $\leq$, where $Class_j \leq Class_i$ means that a user at $Class_i$ can access the resources at $Class_i$ as well as those at $Class_j$. $Class_i$ is called an ancestral class of $Class_j$, whereas $Class_j$ is called a descendant class of $Class_i$. A typical hierarchy is represented by a directed acyclic graph (DAG)[2] (see Figure 1). The type of access control required for resources arranged in a hierarchy is called hierarchical access control (HAC).

Traditional access control techniques are typically classified into three categories [2,3]: (1) discretionary access control (DAC) [4,5]; (2) mandatory access control (MAC) [6,7]; and (3) role-based access control (RBAC) [8–11].

Recently, a new class of HAC techniques, called cryptography-based HAC (CHAC) solutions, has received much attention from researchers. CHAC is looked upon as an alternative to traditional non-cryptographic techniques for HAC. Cryptographic HAC techniques assume that every node in a hierarchy is assigned a (cryptographic) key and all resources at the node are encrypted by the node key. A user at a node gets the node key and the ancestors of the node can derive that key (from their own keys). However, any user cannot derive a key of a higher node in the hierarchy. Encryption makes it possible to store the data in (a large) public storage or to transmit the data over (insecure) open channels.

*Correspondence to: Xukai Zou, Department of Computer Science, Indiana University-Purdue University Indianapolis, IN 46202, USA.
†Email: xkzou@cs.iupui.edu

[1]*A class generally represents a set of resources and users who can access those resources.*
[2]*The nodes of a DAG represent classes in the hierarchy. Nodes, vertices and classes are used interchangeably in the paper.*

Received 23 October 2006
Revised 9 March 2007
Accepted 27 March 2007

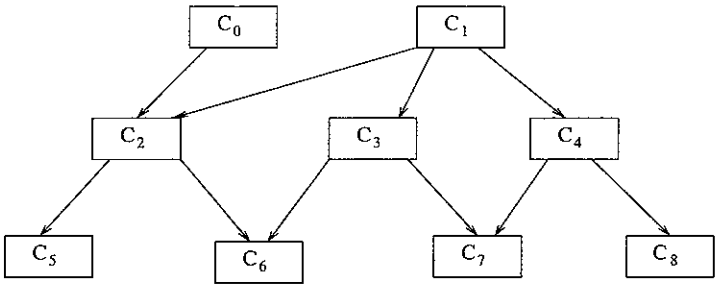

Figure 1. A typical hierarchy

Attackers outside the system are blocked by the strength of cryptographic techniques used and misbehaving users cannot access information they are not supposed to get. Moreover, cryptographic solutions are suitable for use in highly dynamic distributed environments.

It is worth noting that a node represents a class of users and resources. Node secrets need to be securely distributed to all the users of a class in an efficient and scalable way. Few existing CHAC schemes have dealt with this issue; instead they assume that there is a key distribution scheme which can distribute node secrets to its users securely [12–14]. A comprehensive survey of group key management schemes for secure group communication can be found in Zou *et al.* [15]. A unique feature of the new proposed scheme, compared with the existing CHAC schemes, is that it is able not only to be used for key derivation among ancestral nodes and descendant nodes (node level) but also can be seamlessly used in securely distributing nodes secrets to their users (user level). The dynamics at both levels can be dealt with in a uniform and efficient way.

The rest of the paper is organized as follows. In section 2, we briefly summarize existing CHAC schemes. Section 3 presents the newly proposed scheme. We analyze the security of the newly proposed scheme, followed by complexity and performance analysis in Section 4. An illustrative example and experimental results are presented in Section 5. We discuss the features of the proposed scheme in Section 6, followed by a conclusion and future work in Section 7.

2. RELATED WORK

Akl and Taylor [16] proposed the first cryptography-based solution to the problem of HAC in 1983. They assume that an access hierarchy is a partial order relation $\leq$ and represented by a directed acyclic graph (DAG) with only one root. They also assume the presence of a trusted central authority called a Group Controller (GC) (note that this is true for all schemes). The GC selects two large primes p and q and computes $M = pq$ and makes it public. The GC assigns a distinct prime p_i to every vertex v_i in the hierarchy. Now for each v_i, the GC computes $t_i = \Pi_{v_j \not\leq v_i} p_j$ and makes all t_i public. The root vertex v_0 gets $t_0 = 1$. The GC selects a random key K_0 for the root and uses K_0 to compute other vertex keys as $K_i = K_0^{t_i} \bmod M$. Each v_i gets its key K_i securely. Any vertex v_i can derive key K_j of a vertex v_j such that $v_j \leq v_i$ as $K_j = K_i^{t_j/t_i}$ but no such v_j can get K_i. McKinnon *et al.* [17] proposed an algorithm to assign primes that achieves near-optimal assignment. The CHAC scheme proposed by Akl and Taylor is limited to hierarchies with just one root (Note: the scheme can be extended to support multiple roots by introducing a virtual root to which the multiple roots are connected) and provides almost no support for accommodating changes to the hierarchy. Chick and Tavares [18] proposed an improved scheme. This scheme allows a general DAG with multiple roots and no prime is assigned to roots. The GC computes $T = \Pi_{i=1}^{n} p_i \bmod M$ and then for each vertex v_i computes $t_i = \Pi_{v_j \leq v_i} p_j \bmod M$. All t_i are made public. The GC selects a random secret K_0 and computes any key $K_j = K_0^{T/t_i} \bmod M$. Now, any v_i can derive the key of a vertex v_j such that $v_j \leq v_i$ as $K_j = K_i^{t_j/t_i}$. This scheme allows adding vertices in a hierarchy without affecting other vertex keys. The constraint for doing this is that the newly added vertex cannot be a child of any other vertices. As can be seen, this scheme is still very limited in dealing with dynamics.

DOI: 10.1002/nem

The Akl–Taylor scheme discussed above is called a directly dependent scheme since the keys of descendant nodes are directly derived from those of ancestral nodes. Correspondingly, there is a class of schemes called indirectly dependent schemes. In this class of schemes, all vertex keys are generated randomly and then the GC relates ancestral keys to descendant keys by some one-way function. For example, Lin [19] proposed the first indirectly dependent CHAC scheme. This scheme assumes an identification ID_i for every vertex v_i. Each vertex selects its own key K_i and sends it to the GC via the secure channel assumed between the vertex and the GC. For all v_j such that v_j is a child of v_i, the GC computes $r_{ij} = F(K_i; ID_j) \oplus K_j$, where $F(K; ID)$ is a cryptographic one-way function. Now, any v_i can get the key of any child v_j as $K_j = F(K_i; ID_j) \oplus r_{ij}$, but no v_j can get v_i's key. Furthermore, any v_i can derive the key of any descendant v_j by following the path from v_i to v_j. This scheme allows adding or deleting nodes. Still it suffers from the key distribution problem at the user level, in particular when a user leaves from a node, as the departure user can store keys of descendant nodes and continue to use them.

Typical CHAC schemes proposed in the past two decades include, but are not limited to, those of Sandhu [20], Zhong [21], Chen *et al.* [22], Hwang *et al.* [23], Shen and Chen [24], Das *et al.* [25], Chang *et al.* [26], Harn and Lin [27], Chang *et al.* [28], and Atallah *et al.* [29]. All these schemes deal solely with the node-level dynamics. Conversely, there are many key management schemes which are good at supporting user groups and copying with user-level dynamics [30–32]. However, few of these schemes can handle both node-level and user-level dynamics efficiently.

3. A NOVEL CHAC SCHEME BASED ON ACCESS POLYNOMIALS

In this section, we will propose a novel concept/construction of an access polynomial (AP) and its utilization for a novel CHAC scheme. The scheme is applicable to a general hierarchy represented by a DAG (as in Figure 1). We do not pose any restrictions on structure of the hierarchy as some existing schemes do, making our scheme truly generic. Like other CHAC schemes, we assume the presence of a GC which generates polynomials and random keys, publicizes key materials, and maintains the system during dynamic operations. We also assume that each user in the system has a permanent personal secret ID (SID) and a corresponding secret value $H(SID)$, where $H(x)$ is a t-degree polynomial in a finite field F_q (see the discussion below).

3.1 *Principle*

The new scheme is based on polynomials over a finite field F_q [33], in particular, a novel concept/construction of an access polynomial (AP). Here q is a large prime as the system modulus and made public. All arithmetic operations will be performed with mod q, which is omitted for simplicity. The principle is as follows.

The GC selects a random t-degree polynomial $h(x)$, called a masking polynomial, and keeps it secret ($t > 0$).

Degree t determines the security of polynomials, hence the GC can select a large t to ensure security against collusion attacks. A large t will not significantly affect efficiency due to the linear complexity of our scheme in terms of t. For every vertex v_j in the hierarchy, the GC selects a unique (secret) identification $ID_j \in F_q$ and computes $h(ID_j)$ and gives $(ID_j; h(ID_j))$ to v_j securely (see Section 3.4 for the secure distribution using the same mechanism at the user level). The GC also selects a key K_j for every vertex v_j.

Now, for every vertex v_j, the GC constructs an access polynomial (AP) $A_j(x)$, using the ID of v_j and IDs of all v_j's ancestors v_i as follows:

$$A_j(x) = (x - ID_j) \times \prod_{v_j \le v_i} (x - ID_i) + 1$$

DOI: 10.1002/nem

This construction assures that $A_j(x = r) = 1$ for all $r \in \{ID_j$, IDs of v_j's ancestors$\}$ or a random value otherwise. The GC selects a random polynomial $S_j(x)$ of degree t and computes a polynomial $T_j(x) = K_j - S_j(x)$. Thus, $K_j = S_j(x) + T_j(x)$. This equation guarantees that when a user has a share of $S_j(x)$ and a share of $T_j(x)$ at the same value x, the key K_j will be recovered. Both $S_j(x)$ and $T_j(x)$ are kept secret. The GC then computes and publishes two polynomials as $P_j(x) = S_j(x) * A_j(x) + h(x)$ and $R_j(x) = T_j(x) + h(x)$.[3] This construction makes sure that only v_j and its ancestors can derive key K_j (see the discussion below). We will discuss the security of our scheme in detail in Section 4.

3.2 *Key Computation/Derivation*

A vertex v_j can compute its own key K_j by evaluating $S_j(ID_j) = P_j(ID_j) - h(ID_j)$ since $P_j(ID_j) = A_j(ID_j) * S_j(ID_j) + h(ID_j) = 1 * S_j(ID_j) + h(ID_j) = S_j(ID_j) + h(ID_j)$ and $T_j(ID_j) = R_j(ID_j) - h(ID_j)$ first and setting $K_j = S_j(ID_j) + T_j(ID_j)$.

An ancestor v_i of v_j can derive v_j 's key K_j by computing $S_j(ID_i) = P_j(ID_i) - h(ID_i)$ since $P_j(ID_i) = A_j(ID_i) * S_j(ID_i) + h(ID_i) = 1 * S_j(ID_i) + h(ID_i) = S_j(ID_i) + h(ID_i)$ and $T_j(ID_i) = R_j(ID_i) - h(ID_i)$ first and setting $K_j = S_j(ID_i) + T_j(ID_i)$.

If a user with ID_r who is not in v_j or any of its ancestral nodes attempts to get key K_j, the user will end up with $S_j(ID_r) * A_j(ID_r) + T_j(ID_r)$, which is a random value and is useless. As a result, correct hierarchical access control is enforced. Next, we will discuss how the above AP scheme can support dynamic operations.

3.3 *Node/Vertex-Level Dynamics*

There are two levels of dynamics: node level as well as user or resource level. Node-level dynamics include adding, deleting, moving, merging, and splitting nodes and adding and deleting edges. User-level dynamics include a user's joining a node, leaving a node, and moving from one node to another. Correspondingly, the resource dynamics includes adding a resource to a node, removing a resource from a node, and moving a resource from one node to another. The resource dynamics can be processed in the same way as the user dynamics; thus we ignore it for simplicity. Hence, we describe the node-level dynamics in this subsection and then the user-level dynamics in the next subsection.

- *Adding a node as a leaf:* The GC assigns the new leaf node a unique ID_{new}, $h(ID_{new})$ and key K_{new}. The GC constructs $A_{new}(x)$ including ID_{new} and ancestral IDs of the new node, computes $P_{new}(x)$ and $R_{new}(x)$, and makes them public. No other nodes are affected in any way.
- *Adding an intermediate node:* The only change from the above case is that all descendant nodes of the new node are affected. Each descendant v_j needs a new $A'_j(x)$ and hence new $P'_j(x)$ and $R'_j(x)$. Whether a new key K'_j is needed depends on application requirements. If the added node is allowed to access the previous key (and thus the data) of its descendants, the old key K_j does not need to be changed. Otherwise, a new key K'_j is needed. Since $P'_j(x)$ and $R'_j(x)$ are public, changing and sending them to node v_j (or multicasting them to the users of v_j) is simple, without any security complication.
- *Deleting a leaf node:* The GC can delete the node by simply discarding all its public and private parameters. (The discarded IDs and $h(ID)$s should be recorded somehow to prevent them from being reused until the system polynomial is refreshed.)
- *Deleting an intermediate node:* The GC discards all its public and private parameters. No ancestor of the deleted node is affected in any way. For every descendant v_j of the deleted node, the GC changes key K_j and polynomials $A_j(x)$, $P_j(x)$, and $R_j(x)$. For the descendants of the deleted node, a policy is used to either place them as the descendants of the parent (if any) of the deleted node or as the descendants of some other nodes. We ignore the policy issue since it is irrelevant to our discussion here.

[3] *The access polynomial $A_j(x)$ is just contained in $P_j(x)$ but not $R_j(x)$.*

DOI: 10.1002/nem

- *Adding an edge:* Suppose an edge from v_i to v_j is added, the GC recomputes $A'_j(x)$ with inclusion of IDs of v_i and v_i's ancestors who were not the ancestors of v_j, selects a new key K'_j, recomputes and publicizes $P'_j(x)$ and $R'_j(x)$. At the same time, the GC performs this for every descendant of v_j which was previously not a descendant of v_i.
- *Deleting an edge:* If an edge from v_i to v_j is deleted, the GC recomputes $A'_j(x)$ with exclusion of IDs of v_i and v_i's ancestors who will not be the ancestors of v_j, selects a new key K'_j (if needed), recomputes and publicizes $P'_j(x)$ and $R'_j(x)$. At the same time, the GC performs this for every descendant of v_j which will not be a descendant of v_i from now on.
- *Moving a node:* This is equivalent to a combination of deleting and adding a node.
- *Merging n nodes:* This is equivalent to deleting $n - 1$ nodes and changing key materials of one node.
- *Splitting a node:* This is equivalent to adding a few new nodes and changing key materials of one node.

3.4 User-Level Dynamics

The user-level dynamics are important as members can join/leave a vertex at any point in time. The join operation is generally easy and the GC simply gives the vertex *ID* and $h(ID)$ to the joining user. The user can then compute its node key and derive its descendants' keys as well. However, when a user leaves a vertex, all descendant node keys need to be changed; otherwise the leaving user may store those keys and continue to use them.

Let us consider a user's departure at a vertex v_i. The GC changes key K_i to K'_i, ID_i to ID'_i, and $h(ID_i)$ to $h(ID'_i)$. The GC recomputes $P'_i(x)$ and $R'_i(x)$ according to the new K'_i. Thus the ancestors of v_i are not affected in any way. They can get v_i's key using (new) public polynomials $P'_i(x)$ and $R'_i(x)$ just as they did before. For every descendant v_j of v_i, key K_j needs to be changed to K'_j. The GC recomputes $A'_j(x)$ using the new ID'_i and computes and makes public the new $P'_j(x)$ and $R'_j(x)$.

It should be pointed out that once a node's *ID* is changed, the new *ID* (and its corresponding $h(ID)$) needs to be distributed to all its users securely. One elegant feature of our AP mechanism is that the scheme itself can be used to distribute *ID* and $h(ID)$ efficiently. As mentioned previously, every user in the system will be assigned a permanent personal secret identification *SID* and a corresponding $H(SID)$ (note: $H(x)$ is a masking polynomial at user level and multiple $H(x)$s can be selected/used, one for each node). Then the GC constructs an AP $A(x)$ which contains *SID*s of all the users belonging to the node and computes and publicizes two pairs of polynomials:

$$\{P_1(x) = A(x) * S_1(x) + H(x), R_1(x) = T_1(x) + H(x)\} \text{ and}$$
$$\{P_2(x) = A(x) * S_2(x) + H(x), R_2(x) = T_2(x) + H(x)\}$$

where $ID = S_1(x) + T_1(x)$ and $h(ID) = S_2(x) + T_2(x)$. Thus, the users of the node can obtain *ID* and $h(ID)$ securely.

4. SECURITY AND PERFORMANCE ANALYSIS

In this section, we will discuss the security and performance of our scheme. As we will see from the following discussion, because $S_j(x)$ is randomly selected and is changing every time, $T_j(x)$, $P_j(x)$, and $R_j(x)$ are constantly changing as well. Moreover, all public $P_j(x)$s are independent. Thus an attacker or a malicious user cannot figure out secret information from public polynomials, regardless of the number of public polynomials the attacker collects. In addition, there is a strict value binding requirement among all these polynomials: $P_j(x)$, $S_j(x)$, $A_j(x)$, $h(x)$, $R_j(x)$, and $T_j(x)$. That is, only when each of them is evaluated at the same x value are their combinations meaningful. As a result, multiple malicious users cannot successfully collude since their known values are with different x and cannot be combined together. In summary, the security of the scheme is guaranteed. As for the complexity of the scheme, it mainly comes from computing polynomials $P_j(x)$ and $R_j(x)$. $P_j(x)$ has degree $m + t$ and $R_j(x)$ has degree t, so the complexity is of the order of $m + t$. This linear complexity makes the scheme efficient and allows a larger t to be used. We will discuss them in detail next.

DOI: 10.1002/nem

4.1 Security Analysis

The attacks to the new scheme can come from either an individual or collusion of multiple users. Let us consider the single-user attack first.

The key is split as $K_j = S_j(x) + T_j(x)$. To get the key K_j, any user needs a share of $S_j(x)$ and $T_j(x)$ at the same value of x, say $x = ID_r$. The way that $S_j(x)$ and $T_j(x)$ are publicized via $P_j(x)$ containing $A_j(x)$ guarantees that only a valid user, i.e. the one belonging to vertex v_j or its ancestors, can get a share of them. Even if an invalid user succeeds in guessing a valid user's ID as say ID_{guess}, still the invalid user cannot get the share of $S_j(x)$ (or $T_j(x)$) at $x = ID_{guess}$ since it is publicized via polynomial $P_j(x) = A_j(x) * S_j(x) + h(x)$ (or $R_j(x) = T_j(x) + h(x)$). The invalid user can only evaluate $P_j(x = ID_{guess}) = A_j(ID_{guess}) * S_j(ID_{guess}) + h(ID_{guess})$ $= S_j(ID_{guess}) + h(ID_{guess})$ (or $R_j(x = ID_{guess}) = T_j(ID_{guess}) + h(ID_{guess})$). Since the invalid user does not have $h(ID_{guess})$ and cannot compute $h(ID_{guess})$, the invalid user cannot get the share of $S_j(x)$ and $T_j(x)$ at the same value and hence key K_j. This security against guessing attacks is achieved by clever construction of the access polynomial and use of the masking polynomial.

Another single-user attack can occur when an attacker or a malicious user tries to compute a vertex v_i's ID_i from multiple publicized polynomials in which ID_i is contained. There are three cases to consider. First, even though ID_i is contained in $A_i(x)$ and the format of $A_i(x)$ is known, $A_i(x)$ is totally hidden within $P_i(x)$ by two secret polynomials $S_i(x)$ and $h(x)$. Thus, any user other than those in v_i cannot figure out $A_i(x)$ from $P_i(x)$ and hence ID_i. Secondly, when K_i changes (to K_i'), $P_i(x)$ will be changed (to $P_i'(x)$). Even though $A_i(x)$ is contained in both $P_i(x)$ and $P_i'(x)$, they are independent due to the randomness of $S_i(x)$ and $S_i'(x)$ and independence between $S_i(x)$ and $S_i'(x)$. Therefore, any user other than those in v_i cannot figure out ID_i from multiple $P_i(x)$s. Finally, even though ID_i is contained at the same time in $P_i(x)$ and multiple $P_j(x)$s where v_j is a descendant of v_i, again, due to the independence among $P_i(x)$ and $P_j(x)$s, ID_i still cannot be figured out. In summary, due to the independence among the publicized polynomials, the attack of getting ID_i from them is void. Furthermore, even if ID_i were figured out, the attacker would still need to figure out $h(ID_i)$.

Let us consider collusion attacks now. The typical collusion is that two (or more) vertices try to attack their common ancestor. Assume two users a and b from two vertices with $(ID_a, h(ID_a))$ and $(ID_b, h(ID_b))$ collude to attack their common ancestor v_j for getting key K_j or ID_j. Since $P_j(x)$ contains neither ID_a nor ID_b, their collusion for getting K_j or ID_j from $P_j(x)$ is no stronger than any of them individually. Secondly, let us consider that they are trying to get ID_j from $P_a(x)$ and $P_b(x)$ in which ID_j is contained. The values both users know are 10 elements: ID_a, ID_b, $P_a(ID_a)$, $P_b(ID_a)$, $P_a(ID_b)$, $P_b(ID_b)$, $S_a(ID_a)$, $S_b(ID_b)$, $h(ID_a)$, and $h(ID_b)$. Since $P_a(x) = S_a(x) * A_a(x) + h(x)$, which requires that x be substituted with the same value in all four polynomials $P_a(x)$, $S_a(x)$, $A_a(x)$, and $h(x)$, it is impossible for the two users to mix these 10 known values to form informative equations. For example, the two users cannot form an equation $P_a(ID_b) = S_a(ID_a) * A_a(x)$ $+ h(ID_b)$ from $P_a(x)$. Thus, they cannot get information about $A_a(x)$ and hence ID_j. As a result, this security against collusion aiming at obtaining K_j or ID_j is achieved by binding the same x between $S_j(x)$ and $T_j(x)$ and among $P_j(x)$, $S_j(x)$, $A_j(x)$ and $h(x)$.

The degree of a polynomial decides its collusion resistance. $h(x)$ is of degree t and so are $S_j(x)$ and $T_j(x)$. Thus, the proposed scheme can resist the collusion of up to t users. Since $S_j(x)$ and $T_j(x)$ are always changing, it is difficult and also useless for malicious users to collude to recover $S_j(x)$ or $T_j(x)$. Thus the collusion attack will aim at $h(x)$. Since every user has an ID and $h(ID)$, more than t users can figure out the entire $h(x)$ using polynomial interpolation. In addition, it is possible for fewer than $t + 1$ users, even one user, to figure out $h(x)$. As stated before, when a user departs from a vertex v_i, the vertex's ID_i and $h(ID_i)$ need to be changed. If there are more than $t - 1$ times of user leaves from a vertex v_i, a user who stays always at vertex v_i will obtain at least t new pairs of $(ID_i, h(ID_i))$ and, plus its original $(ID_i, h(ID_i))$, is able to interpolate $h(x)$. To solve this problem, we can do as follows: (1) selecting larger t and (2) refreshing $h(x)$ once the number of ID changes reaches $t - 1$. Thanks to the efficiency of our scheme, which allows a larger t to be selected, and thanks to the uniform feature of our scheme, which allows the refreshed ID and $h(ID)$ to be securely and efficiently distributed to users using the scheme itself, the problem can be solved without much difficulty. In addition, it is a good practice to refresh $h(x)$ periodically. How often

DOI: 10.1002/nem

to refresh $h(x)$ is an important issue which is related to security level, performance, and application domains. This is worthy of further study in the future.

4.2 *Performance Analysis*

We will consider the efficiency of our scheme in terms of storage, computation, and communication complexities. Let us assume that there are n vertices in the hierarchy and the maximum number of ancestors any vertex can have is m.

A vertex v_j needs to store its secret parameters, viz. $\{ID_j, h(ID_j), K_j\}$. This means that the storage requirement for any vertex can be considered as $O(1)$, i.e., constant and independent from n (and m too) (in fact, $O(\log q)$ since all these values are in F_q). Note that a vertex does not need to store information about the hierarchy of its descendants. In some other schemes, when a vertex derives its descendant's key, the derivation follows a chain from the vertex to its child, grandchild, . . . , and finally, the descendant. So a vertex in these schemes needs to remember/store exact information about its descendents and their hierarchical relation. However, in our scheme, the vertex only needs to substitute its ID in the polynomial directly and the descendant's key is revealed. It may be needed to store the indices of its descendants to indicate which vertices are its descendants but these indices are relatively small numbers. The GC needs to store the secret t-degree masking polynomial $h(x)$ (plus user-level masking polynomial $H(x)$) and all n access polynomials $A_j(x)$, each of maximum degree m. To store the polynomials, it is only required to store the coefficients. So the storage requirement for the GC is $O(n * m + t)$. Considering the role of the GC, this is not a great amount of storage.

As far as computation complexity is concerned, one advantage of the proposed scheme is that both the node key computation and the descendant key derivation are directly evaluated in the same way. The evaluation for K_j consists of computing $P_j(ID)$ and $R_j(ID)$, two subtractions (subtracting $h(ID)$), and one addition $(S_j(ID) + T_j(ID))$. $P_j(x)$ has degree $t + m$ and $R_j(x)$ has degree t. So the key computation/derivation complexity is $O(t + m)$, which is also independent from n. It is linear as well, which makes our scheme very efficient. In addition, the key computation/derivation is so efficient that we can have t be very large, which improves the security of the new scheme. Thus in our scheme security and efficiency do not conflict with each other as they do in most existing schemes.

Regarding communication cost, whenever a vertex v_j's key K_j is updated (due to the dynamics), $P_j(x)$ and $R_j(x)$ need to be publicized (or broadcast), which again is $O(t + m)$ and independent from n.

The complexities of the new scheme are summarized in Table 1. As a result, the complexities of our scheme are completely independent from n and thus the new scheme is very efficient and scalable.

5. AN ILLUSTRATIVE EXAMPLE AND EXPERIMENT RESULTS

We implemented the above proposed scheme and give one example generated by the program in this section. We evaluated the running times and the results for different scenarios are also presented here.

Suppose the access hierarchy is as Figure 2 and let $t = 10$ and $q = 251$. We will consider both node-level operations and user-level operations. Suppose the masking polynomial is

$$h(x) = 96x^{10} + 67x^9 + 191x^8 + 164x^7 + 41x^6 + 229x^5 + 234x^4 + 150x^3 + 205x^2 + 54x + 61$$

Storage	Computation	Communication
$O(1)$ at users $O(n * m + t)$ at GC	$O(t + m)$ for key derivation $O(t + m)$ for computing $P(x)$ and $R(x)$	$O(t + m)$

Table 1. Complexities of the newly proposed scheme

DOI: 10.1002/nem

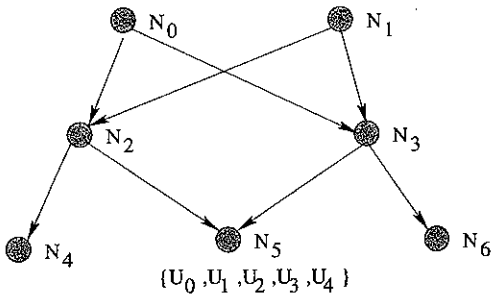

Figure 2. An example

and the secret node IDs and their corresponding $h(ID)$ for nodes $N_0, N_1, \ldots, N_6$ are

$(ID_0 = 85, h(ID_0) = 136)$, $(ID_1 = 119, h(ID_1) = 122)$, $(ID_2 = 145, h(ID_2) = 177)$, $(ID_3 = 30, h(ID_3) = 57)$, $(ID_4 = 186, h(ID_4) = 3)$, $(ID_5 = 160; h(ID_5) = 18)$, $(ID_6 = 225, h(ID_6) = 88)$.

Let us first consider node-level key distribution and key derivation using node N_5 as an example. Supposing the GC randomly selects a node key for N_5 as $K_5 = 213$, it will distribute the key as follows:

- compute N_5's access polynomial as

$$A_5(x) = (x - h(ID_0))(x - h(ID_1))(x - h(ID_2))(x - (ID_3))(x - h(ID_5)) + 1$$
$$= x^5 + 214x^4 + 114x^3 + 141x^2 + 161x + 220;$$

- select a random polynomial such as

$$S_5(x) = 204x^{10} + 24x^9 + 167x^8 + 35x^7 + 110x^6 + 233x^5 + 131x^4 + 114x^3 + 38x^2 + 211x + 249$$

and compute

$$T_5(x) = 47x^{10} + 227x^9 + 84x^8 + 216x^7 + 141x^6 + 18x^5 + 120x^4 + 137x^3 + 213x^2 + 40x + 215;$$

- compute $P_5(x)$ and $R_5(x)$ respectively and publicize them:

$$P_5(x) = 204x^{15} + 6x^{14} + 196x^{13} + 5x^{12} + 116x^{11} + 1x^{10} + 55x^9 + 87x^8 + 156x^7 +$$
$$160x^6 + 88x^5 + 88x^4 + 129x^3 + 86x^2 + 219x + 123; \text{ and}$$
$$R_5(x) = 143x^{10} + 43x^9 + 24x^8 + 129x^7 + 182x^6 + 1x^{10} + 247x^5 + 103x^4 + 36x^3 + 167x^2 + 94x + 25.$$

As a result, nodes N_0, N_1, N_2, N_3, N_5 can compute key K_5 from $P_5(x)$ and $R_5(x)$. For example, N_2 computes the key as

$$K_5 = (P_5(145) - h(145)) + (R_5(145) - h(145)) = (12 - 117) + (53 - 177) = -165 - 124 = -289 = 213\%251$$

and N_5 also computes the key as

$$K_5 = (P_5(160) - h(160)) + (R_5(160) - h(160)) = (136 - 18) + (113 - 18) = 118 + 95 = 213 = 213\%251$$

However, other nodes which are not the ancestors of N_5 cannot compute the key. For example, N_4 obtains $(P_5(186) - h(186)) + (R_5(186) - h(186)) = (211 - 3) + (87 - 3) = 208 + 84 = 292 = 41\%251$, which is not the key.

Next, let us consider how the same access polynomial mechanism can be used to distribute the node ID and $h(ID)$ to node users at the user level. Consider N_5 and assume that it involves five users U_0, U_1, U_2, U_3, U_4. Here assume that we have a different masking polynomial $H(x)$ for N_5 as:

$$H(x) = 189x^{10} + 4x^9 + 244x^8 + 34x^7 + 163x^6 + 73x^5 + 83x^3 + 84x^3 + 125x^2 + 237x + 70$$

DOI: 10.1002/nem

Moreover, assume that the users are assigned their $(SID, H(SID))$ as follows:

$$(SID_0 = 5, H(SID_0) = 221), (SID_1 = 54, H(SID_1) = 23), (SID_2 = 189, H(SID_2) = 8),$$
$$(SID_3 = 74, H(SID_3) = 199), (SID_4 = 140, H(SID_4) = 123)$$

From these SIDs, the access polynomial at the user level can be formed as

$$A(x) = \Pi_{i\in\{0,4\}}(x - H(SIDi)) + 1 = x^5 + 40x^4 + 55x^3 + 23x^2 + 72x + 210$$

In order to distribute $ID_5 = 160$ to these users securely, the GC constructs and publicizes polynomials $P(x)$ and $R(x)$ as

$$P(x) = 130x^{15} + 114x^{14} + 88x^{13} + 94x^{12} + 158x^{11} + 105x^{10} + 173x^9 + 140x^8 + 244x^7 + 58x^6 + 217x^5 + 87x^4 + 25x^3 + 140x^2 + 233x + 187$$

and

$$R(x) = 59x^{10} + 70x^9 + 148x^8 + 128x^7 + 80x^6 + 222x^5 + 202x^4 + 24x^3 + 165x^2 + 234x + 190$$

where

$$S(x) = 130x^{10} + 185x^9 + 96x^8 + 157x^7 + 83x^6 + 102x^5 + 132x^4 + 60x^3 + 211x^2 + 3x + 40$$

and

$$T(x) = 121x^{10} + 66x^9 + 155x^8 + 94x^7 + 168x^6 + 149x^5 + 119x^4 + 191x^3 + 40x^2 + 248x + 120$$

Thus, these users can obtain ID_5. For example, U_2 computes

$$ID_5 = (P_5(189) - H(189)) + (R_5(189) - H(189)) = (115 - 8) + (61 - 8) = 107 + 53 = 160 = 160\%251$$

Similarly, to distribute $h(ID_5) = 18$, the GC computes and publicizes polynomials $P(x)$ and $R(x)$ as

$$P(x) = 122x^{15} + 114x^{14} + 19x^{13} + 192x^{12} + 232x^{11} + 76x^{10} + 114x^9 + 41x^8 + 218x^7 + 174x^6 + 46x^5 + 55x^4 + 202x^3 + 196x^2 + 2x + 29$$

and

$$R(x) = 67x^{10} + 1x^9 + 27x^8 + 198x^7 + 103x^6 + 46x^5 + 18x^4 + 107x^3 + 205x^2 + 3x + 87$$

where

$$S(x) = 122x^{10} + 3x^9 + 217x^8 + 87x^7 + 60x^6 + 27x^5 + 65x^4 + 228x^3 + 171x^2 + 234x + 1$$

and

$$T(x) = 129x^{10} + 248x^9 + 34x^8 + 164x^7 + 191x^6 + 224x^5 + 186x^4 + 23x^3 + 80x^2 + 17x + 17$$

In the same way, these users can obtain $h(ID_5)$. For example, U_2 computes

$$h(ID_5) = (P_5(189) - H(189)) + (R_5(189) - H(189)) = (84 - 8) + (201 - 8) = 76 + 193 = 269 = 18\%251$$

If any other user wants to obtain ID_5 and $h(ID_5)$ using the same method, they will get useless values.

It is worth mentioning that $q = 251$ and $t = 10$ were selected as such just for illustrative purposes. In real applications they should be much larger. For example, in order to prevent brute-force search attack on ID, $h(ID)$, q should be selected at least with a bit length of 128. Moreover, from Figures 3 and 4, t can be selected as 256 without affecting efficiency.

We implemented the proposed scheme in JAVA and did extensive experiments. P was selected to be 128 bits to guarantee sufficient security. The number of users and the degree of polynomial $h(x)$ are

DOI: 10.1002/nem

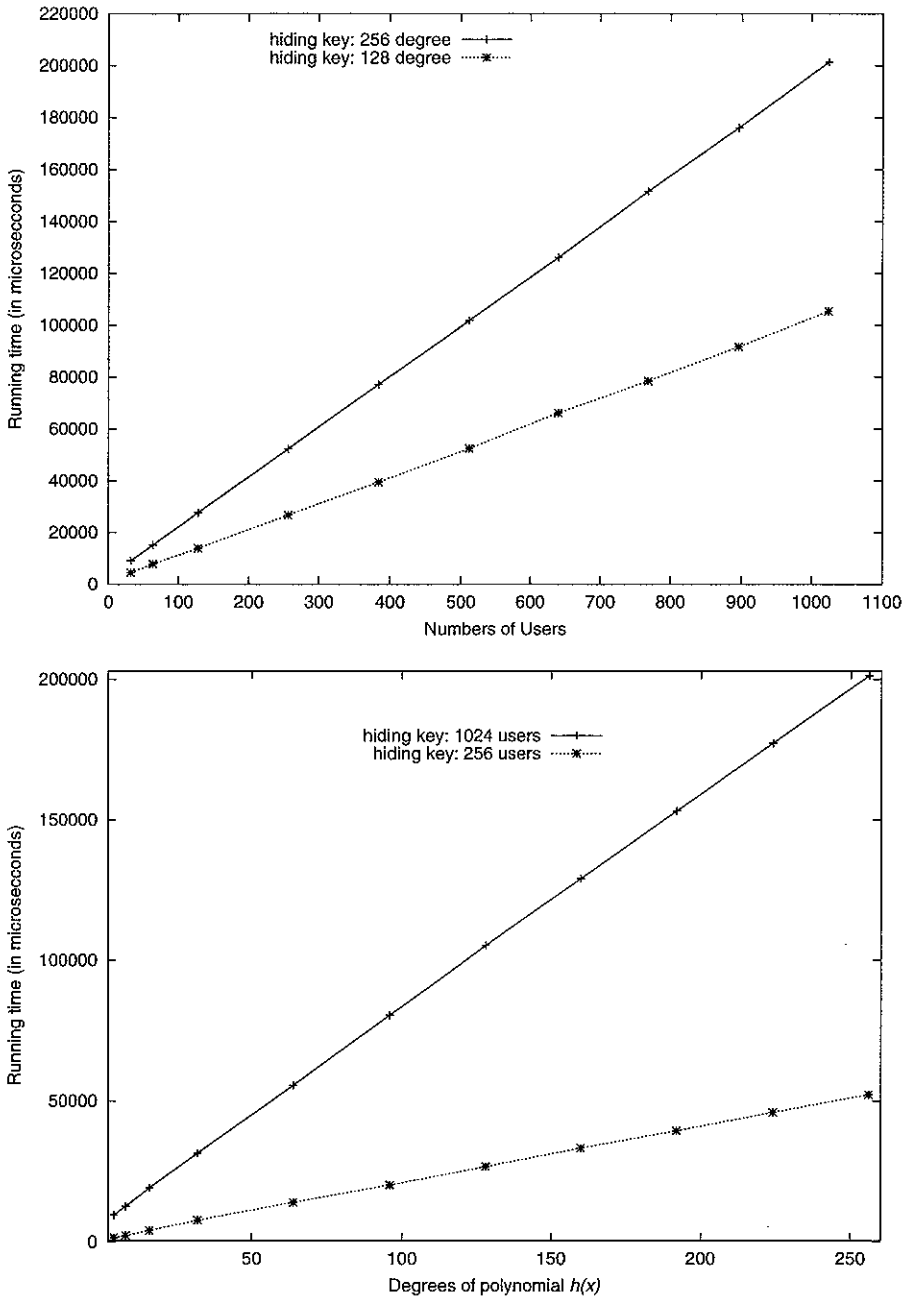

Figure 3. Running times for hiding key, based on number of users (top) and on polynomial degree (bottom)

important factors affecting the performance of the scheme. Two main operations we considered are: (1) hiding the key via generating $P(x)$ and $R(x)$ (which, of course, requires generating $S(x)$ and $T(x)$); (2) extracting the key. The program was run on a Dell laptop computer with a 1.8 GHz CPU and 1.0 Gbytes of RAM. To obtain stable running times, we ran each test case 100 times and averaged the running times. The results are shown in Figures 3 and 4.

DOI: 10.1002/nem

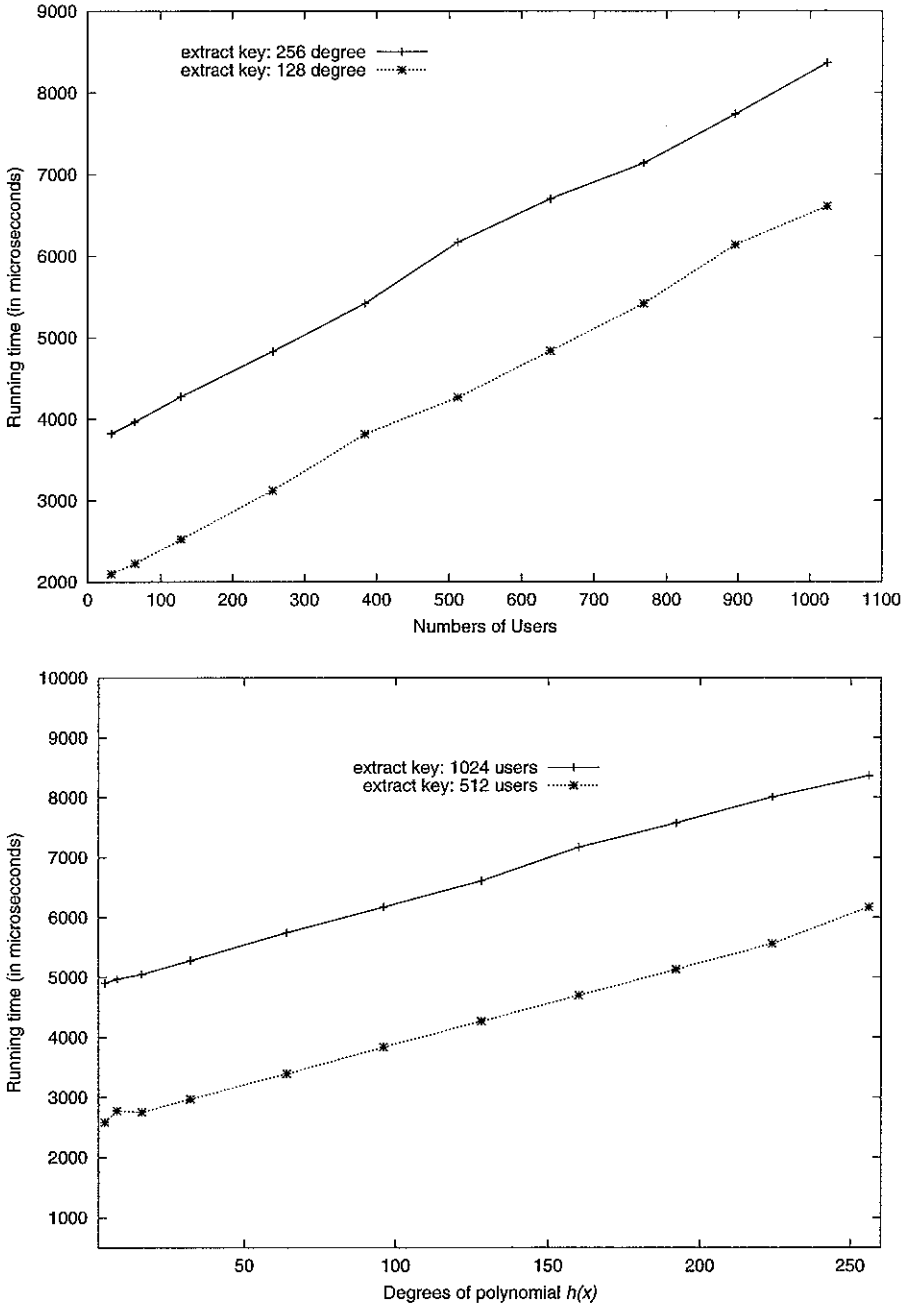

Figure 4. Running times for extracting key, based on number of users (top) and on polynomial degree (bottom)

From the figures, it can be seen that the running times for computing $P(x)$ and $R(x)$ to hide the secret key and for deriving the key from $P(x)$ and $R(x)$ are linear to both m and t. This validates the theoretical analysis in the previous section and proves that the scheme is computationally efficient. Large m indicates that the proposed scheme scales well to admit large group sizes or large number of ancestors, and large t indicates that the scheme is able to defend collusion of large populations of malicious users.

DOI: 10.1002/nem

6. DISCUSSION

In this section we will summarize the major properties of the proposed scheme.

- A good solution must be generic in the sense that it should have no restrictions on the hierarchy. The proposed scheme can be easily applied to any DAG—even those that have multiple roots.
- The best feature of the proposed scheme is the way in which it deals with user- and node-level dynamics in a uniform manner. All the cryptography-based HAC schemes, including the newest one [29] so far, lack this capability.
- The proposed scheme is scalable and does not limit the number of vertices or levels in the hierarchy. It can work well with a large number of vertices without loss of efficiency.
- Unlike many other schemes which derive keys via an iterative chain, the proposed scheme derives all keys directly.
- In most other HAC schemes, including the most recent one [29], every vertex must know both its descendants and the exact hierarchical relation among these descendants, because these schemes derive their descendants' keys via an iterative derivation chain. However, the proposed scheme does not need to know the hierarchical relation of a node's descendants.
- The proposed scheme is very efficient in terms of time and space complexity. Every node has just three secrets: its ID, $h(ID)$ and the key. All the other information is public. The computational and communication complexity is $O(t + m)$, which is far better than most of the existing schemes.
- The proposed scheme does not have a problem from revoked members. Our scheme is truly dynamic and efficient.
- The proposed scheme is very secure. Although most of the key derivation parameters are public, no one can derive keys unless they are allowed to. The degree of polynomials is assumed to be t, which can be very large. Even with a large t, efficiency is better than any other t-secure scheme. How large should t be? In general, t is application dependent. In addition, collusion itself is not an easy task and the collusion of more than two users may be easily disclosed by the colluding users themselves. From the experiment, when t is 256, the times for hiding key and extracting key for a size of 1024 users are just 0.2 s and 0.008 s.

6.1 *Enforcement of Other Access Models*

The hierarchical access model discussed so far is called 'transfer up' [29]; that is, the access permission of a vertex is transferred up to its ancestors. Some other models have been proposed [29,34,35], two of which are: (1) 'transfer down', in which the access permission of a vertex is transferred down to its descendants; and (2) 'depth-limited transfer', in which the access permission of a vertex is transferred either up or down but only to a certain depth.

Both of the new models can be implemented as easily as the regular model. For the 'transfer down' model, the access polynomial $A(x)$ of a vertex is constructed to include the IDs of its descendants. For the 'depth-limited transfer' model, the access polynomial $A(x)$ of a vertex is constructed to include the IDs of both its ancestors and descendants which are within the limited depth from the vertex.

In fact, it can be seen that the proposed scheme can be easily adapted to a 'random' access model in the following sense: if a vertex v_i access permission is wanted to be transferred to a random other vertex v_j, regardless of the relation between the two vertices, simply include v_j's ID_j in the construction of $A_i(x)$.

In summary, the proposed scheme is highly secure, flexible, efficient, scalable, generic and practical. It conforms to the specifications of an ideal cryptography-based scheme for hierarchical access control.

7. CONCLUSIONS

In this paper we proposed a novel cryptographic scheme for hierarchical access control. The proposed scheme is based on an access polynomial, a masking polynomial, and representation of a key as the sum

DOI: 10.1002/nem

of two polynomials to share keys. The proposed scheme is highly secure, efficient, scalable, generic, and practical, and provides a uniform and comprehensive solution to the dynamic hierarchical access control problem. Integrating the new scheme in real medical information system at a Veterans' Affairs medical center is one of the applications we are currently pursuing since our scheme can allow doctors, physicians, nurses, surgeons, professors, researchers, health insurance personnel, etc. to share patients' information in a hierarchical, dynamic and finely controlled manner.

ACKNOWLEDGEMENT

This work was partially supported by US NSF grant CCR-0311577.

REFERENCES

1. Silberschatz A, Galvin PB, Gagne G. *Operating System Concepts* (6th edn). Wiley: New York (2001).
2. Sandhu RS, Samarati P. Access control: principles and practice. *IEEE Communications Magazine* 1994; **32**(9): 40–48.
3. Tripunitara MV, Li N. Access control: comparing the expressive power of access control models. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, 2004; 62–71.
4. Harrison M, Ruzzoand W, Ullman J. Protection in operating systems. *Communications of the ACM* 1976; **19**(8): 461–471.
5. Lampson B. Protection (computer system access). *Operating Systems Review* 1974; **8**(1): 18–24.
6. Denning DE. A lattice model of secure information flow. *Communications of the ACM* 1976; **19**(5): 236–243.
7. Sandhu RS. Lattice-based access control model. *IEEE Computer* 1993; **26**(11): 9–19.
8. Al-Kahtani MA, Sandhu R. Rule-based RBAC with negative authorization. In *20th Annual Computer Security Applications Conference (ACSAC '04)*, 2004; 405–415.
9. Moyer MJ, Ahamad M. Generalized role-based access control. In *IEEE 21st International Conference on Distributed Computing Systems*, 2001; 391–396.
10. Park JS, Hwang J. RBAC for collaborative environments: role-based access control for collaborative enterprise in peer-to-peer computing environments. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies*, 2003; 93–99.
11. Sandhu RS, Coyne EJ, Feinstein HL, Youman CE. Role-based access control models. *IEEE Computer* 1996; **29**(2): 38–47.
12. Tseng YM. A scalable key-management scheme with minimizing key storage for secure group communications. *International Journal of Network Management* 2003; **13**(6): 419–425.
13. Wong CK, Gouda M, Lam SS. Secure group communications using key graphs. *IEEE/ACM Transactions on Networks* 2000; **8**(1): 16–30.
14. Yang WH, Shieh SP. Secure key agreement for group communications. *International Journal of Network Management* 2001; **11**(6): 365–374.
15. Zou X. Ramamurthy B, Magliveras SS (eds). *Secure Group Communications over Data Networks*. Springer: New York, 2004.
16. Akl SG, Taylor PD. A cryptographic solution to the problem of access control in a hierarchy. *ACM Transactions on Computer Systems* 1983; **1**(3): 239–248.
17. MacKinnon SJ, Taylor PD, Meijer H, Akl SG. An optimal algorithm for assigning cryptographic keys to control access in a hierarchy. *IEEE Transactions on Computers* 1985; **34**(9): 797–802.
18. Chick GC, Tavares SE. Flexible access control with master keys. In *Proceedings on Advances in Cryptology (CRYPTO '89)*, LNCS 435, 1989; 316–322.
19. Lin CH. Dynamic key management scheme for access control in a hierarchy. *Computer Communications* 1997; **20**(15): 1381–1385.
20. Sandhu RS. Cryptographic implementation of a tree hierarchy for access control. *Information Processing Letters* 1988; **27**(2): 95–98.
21. Zhong S. A practical key management scheme for access control in a user hierarchy. *Computers and Security* 2002; **21**(8): 750–759.
22. Chen TS, Chung YF, Tian CS. A novel key management scheme for dynamic access control in a user hierarchy. In *International Computer Software and Applications Conference (COMPSAC)*, 2004; 396–397.

DOI: 10.1002/nem

23. Hwang MS, Liu CH, Lo JW. An efficient key assignment for access control in large partially ordered hierarchy. *Journal of Systems and Software* 2004; **73**(3): 507–514.
24. Shen VRL, Chen TS. A novel key management scheme based on discrete logarithms and polynomial interpolations. *Computers and Security* 2002; **21**(2): 164–171.
25. Das ML, Saxena A, Gulati VP, Phatak DB. Hierarchical key management scheme using polynomial interpolation. *SIGOPS Operating Systems Review* 2005; **39**(1): 40–47.
26. Chang CC, Lin IC, Tsai HM, Wang HH. A key assignment scheme for controlling access in partially ordered user hierarchies. In *International Conference on Advanced Information Networking and Applications (AINA)*, 2004; 376–379.
27. Harn L, Lin HY. A cryptographic key generation scheme for multilevel data security. *Computers and Security* 1990; **9**(6): 539–546.
28. Chang CC, Hwang RJ, Wu TC. Cryptographic key assignment scheme for access control in a hierarchy. *Information Systems* 1992; **17**(3): 243–247.
29. Atallah MJ, Frikken KB, Blanton M. Dynamic and efficient key management for access hierarchies. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS '05)*, 2005; 190–202.
30. Abdel-Hafez A, Miri A, Orozco-Barbosa L. Scalable and fault-tolerant key agreement protocol for dynamic groups. *International Journal of Network Management* 2006; **16**(3): 185–202.
31. Lee FY, Shieh SP. Scalable and lightweight key distribution for secure group communications. *International Journal of Network Management* 2004; **14**(3): 167–176.
32. Tseng YM. Efficient authenticated key agreement protocols resistant to a denial-of-service attack. *International Journal of Network Management* 2005; **15**(3): 193–202.
33. Lidl R, Niederreiter H. *Introduction to Finite Fields and their Applications*. Cambridge University Press: New York (1986).
34. Bell D, LaPadula L. Secure computer systems: mathematical foundations. *Technical Report MTR2547*, MITRE Corporation: Bedford, MA, 1973.
35. Crampton J. On permissions, inheritance and role hierarchies. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS '03)*, 2003; 85–92.