Smart Computing Review

A New Key Management Scheme for a User Hierarchy based on a Hybrid Cryptosystem

Vanga Odelu[1], Ashok Kumar Das[2] , and Adrijit Goswami[3]

[1] Rajiv Gandhi University of Knowledge Technologies / Hyderabad 500 032, India / odelu.vanga@gmail.com]

[2] Center for Security, Theory and Algorithmic Research, International Institute of Information Technology / Hyderabad 500 032, India / iitkgp.akdas@gmail.com, ashok.das@iiit.ac.in

[3] Department of Mathematics, Indian Institute of Technology / Kharagpur 721 302, India / goswami@maths.iitkgp.ernet.in]

* Corresponding Author: Vanga Odelu

Received December 21, 2012; Revised January 30, 2013; Accepted February 7, 2013; Published February 28, 2013

Abstract: In general applications', changing a secret key of a user provides maximum security for the system. Thus, a user in an organization needs to change his/her secret key from time to time to achieve maximum security. Changing the secret key of a user in an organization is more frequent than a user simply joining or leaving the organization. Dynamic properties such as changing the key of a user in a hierarchy affects the performance of an access control scheme more than other dynamic properties such as adding or deleting a security class to the hierarchy. As a result, we need to design an effective access control scheme to change keys of users in the hierarchy. However, most schemes proposed in the literature do not provide an efficient solution. In this paper, we propose a new key management mechanism for hierarchical access control that provides an effective solution. Moreover, our scheme efficiently supports other dynamic properties, such as adding or deleting a security class. Further, we proposed an efficient procedure for key generation and key derivation for access control. Through formal and informal security analysis, we prove our scheme tolerates various known attacks in access control. We show that our scheme has better performance compared to other related schemes. As a result, higher security along with efficient access control properties makes our scheme more suitable for practical applications.

Keywords: Changing secret key, hierarchical access control, digital signature, one-way hash function, security

DOI: 10.6029/smartcr.2013.01.005

Introduction

User hierarchy is a critical issue in access control policy for information protection systems. Since the computer communication system tends to be increasingly complicated, computer cryptography and information security obviously become more important than ever before. The rapid growth of computer networks and technology constructs a favorable environment that can tolerate multi-users in a hierarchy, which makes sharing resources unavoidable. Both industrial and academic fields have serious concerns about protecting information security from unauthorized users. Within this user hierarchy structure, a user is allowed to access all of the files of users at lower levels in the hierarchy. A hierarchy is a natural way of organizing users to reflect authority and responsibility. High-level users inherit all privileges of, and access to, users below them. The top level user possesses the greatest authority, and the bottom level user has the least authority.

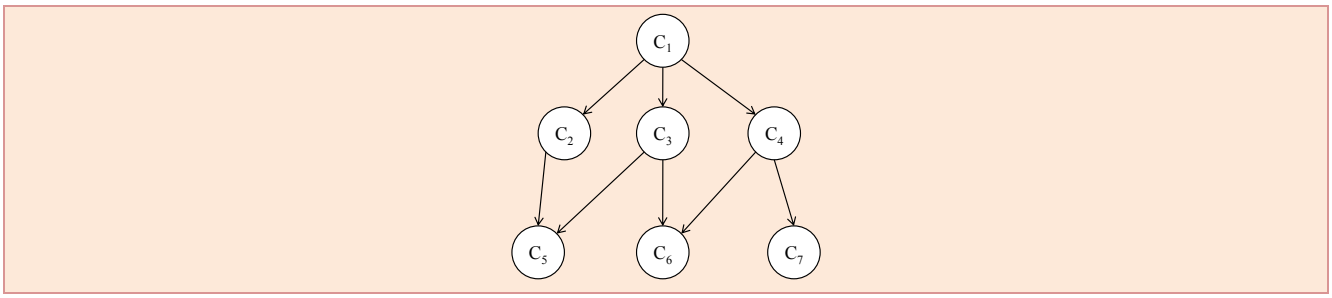

Figure 1. An example of a POSET hierarchy

Users are divided into disjoint security classes according to their privileges. A set of N disjoint security classes, say C, where $C = \{C_1, C_2, \ldots, C_N\}$ forms a partially ordered set (POSET) with respect to a binary relation "≤". $C_j \leq C_i$ means the security level of the class C_i is higher or equal to the security level of the class C_j, and we call security class C_i the predecessor of security class C_j, and security class C_j a successor of security class C_i. Let this relation be denoted by $(C_i, C_j) \in R_{i,j}$. The hierarchy is managed by a trusted central authority (CA), which can initialize all the parameters in the hierarchy according to defined user privileges and which constructs public parameters corresponding to each user in the hierarchy in order to distribute secret keys SK_i's to the authorized security class C_i and declares corresponding public information. In a hierarchy, a predecessor security class C_i can always derive its successor's secret key, SK_j, so that the information encrypted by SK_j by the successor security class C_j (lower than security class C_i) can be decrypted by security class C_i using the derived secret key, SK_j, of the class C_j. Thus, C_i is able to retrieve information of C_j. For example, consider a POSET hierarchy shown in Figure 1. In this figure, we have the following relationships: $\{C_1 \leq C_1, C_2 \leq C_1, C_3 \leq C_1, C_4 \leq C_1, C_5 \leq C_1, C_6 \leq C_1, C_7 \leq C_1\}$; $\{C_2 \leq C_2, C_5 \leq C_2\}$, $\{C_3 \leq C_3, C_5 \leq C_3, C_6 \leq C_3\}$; $\{C_4 \leq C_4, C_6 \leq C_4, C_7 \leq C_4\}$; $\{C_5 \leq C_5\}$; $\{C_6 \leq C_6\}$; and $\{C_7 \leq C_7\}$. The naive solution is that security class C_1 needs to store secret keys SK_2, SK_3, SK_4, SK_5, SK_6, SK_7 and its own secret key, SK_1. In the same way, other security classes C_2, C_3, C_4 need to store keys for their successors, too. In practical applications, each security class will have a large number of successors, and storing the corresponding keys leads to huge storage overhead and insecurity from storing the keys. This requires design of an efficient dynamic access control scheme to manage secret keys in the hierarchy.

Review of the Previous Works

Akl and Taylor [1] first proposed the concept of a cryptographic key assignment scheme in a general POSET hierarchy based on integer factorization. As pointed out by Wang and Laih [2], Akl and Taylor's scheme is inflexible because all keys are related in a generally dependent manner, and it is not easy to adjust the hierarchy. In addition, it is hard to update a key without reassigning a lot of other keys in the hierarchy. Later, Harn and Lin [3] proposed an improvement on Akl and Taylor's scheme. However, when there are many security classes in the hierarchy, Harn and Lin [3] require a large amount of storage space due to a large number of public parameters, as explained by Hwang and Yang [4]. Harn and Lin's scheme reduces the required storage space for the public parameters in the system as compared to the Akl-Taylor scheme. After that, Wang and Laih [5] showed that Hwang and Yang's scheme is vulnerable to the collusion attack, where some security classes conspire to derive the secret keys of other leaf security classes. Recently, Lo et al. [6] proposed a key management scheme for a large leaf class hierarchy based on a public key cryptosystem. However, if the hierarchy consists of a large

number of leaf security classes with more than one parent, Lo et al.'s scheme needs to generate a large number of primes. Wong et al.[21] proposed a secure group communications protocol using key graphs. Their scheme provides three rekeying strategies and the group key management service, using any of these three rekeying strategies. Their scheme is also scalable to large groups with frequently joining and leaving of members.

Many cryptographic solutions have been proposed in the literature to solve the hierarchical access control problem efficiently. Sandhu [7] proposed a scheme for a tree hierarchy based on one-way hash function. However, it is not applicable to a general hierarchy. Yang and Li [8] proposed an access control scheme for a general hierarchy based on a one-way hash function. Chung et al. [9] proposed an efficient solution for a key management scheme. However, Das et al. [10] pointed out a security flaw in Chung et al.'s scheme, where an attacker who is not a user in any security class in the user hierarchy can derive the secret key of a security class by using the exterior root finding attack. They also proposed an improvement on Chung et al.'s scheme to remedy this serious security flaw. In recent years, several key management schemes have been proposed for a general hierarchy [11] [12] [13] [14] [15] [16]. However, these schemes do not provide an efficient solution to the dynamic key change property in a hierarchy.

Based on the literature survey, we list the following parameters to evaluate the effectiveness of cryptographic key assignment schemes in a POSET hierarchy:

- Effort needed to change a secret key of a security class in the hierarchy.
- Effort needed to update the hierarchy when a modification occurs.
- Size of secret keys and the storage space required for public parameters.
- Effort needed to compute key generation and key derivation.
- Guarantee of computational security.

Our Contributions

In this paper, we propose a new hybrid key management mechanism for a hierarchical access control scheme, which is based on RSA and symmetric key cryptosystem and provides an efficient solution to the dynamic key change property. Moreover, our scheme efficiently supports other dynamic properties such as adding and deleting a security class.

- Our scheme provides authentication and confidentiality for the secret key of a security class.
- Our scheme provides an effective and efficient solution to the problem of changing a secret key without affecting the public data of other security classes.
- The time complexity of both the key generation and the key derivation is O (1).
- Since changing a secret key provides maximum security for the user in a hierarchy, our scheme is more suitable for practical applications.
- Through formal and informal security analysis, we show that our scheme is secure against possible known attacks.
- Our scheme has better performance as compared to other related existing schemes.

The remainder of this paper is organized as follows. In Section 2, we propose our access control scheme. Section 3 describes the solution to the dynamic access control problems in the hierarchy. In Section 4, we perform security analysis of our proposed scheme. In Section 5, we prove the correctness of the key derivation phase and thoroughly analyze storage overhead and computational overhead. We compare the computational cost and performance analysis of our scheme with the closely related existing schemes in Section 6. Finally, we conclude the paper in Section 7.

Our Proposed Access Control Scheme

We assume a set of N security classes, say $C = \{C_1, C_2, \ldots, C_N\}$ which forms a POSET hierarchy with the partially ordered relation " $\leq$ ". The central authority first sets up its own public and private parameters. The CA constructs a hierarchy of the POSET $(C, \leq)$ and it assigns secret key SK_i to each security class C_i in the POSET hierarchy. In the key derivation phase, we explain the process of deriving a secret key of a successor security class in a hierarchy. We use the secure one-way hash function H(.) (for example SHA-1 [18]) and the symmetric key encryption/decryption E(.)/D(.) (for example AES-128 [17]). Note that the AES-128 128-bit cipher-text uses a 128-bit key. SHA-1 produces a 160-bit hash value or message digest.

■ Set-up phase

In this phase the central authority performs the following steps.

Step 1. The CA selects two large secret distinct primes, p and q ($p \neq q$). The primes p and q could be chosen by using either the randomized Miller-Rabin primality test algorithm [19] or the deterministic AKS primality test algorithm [20].

Step 2. The CA then computes the public parameter $m = pq$ as the product of two generated primes p and q, and the number $\varphi(m) = (p-1)(q-1)$ respectively, where $\varphi(.)$ is the Euler's totient or phi function [19]. Note that $\varphi(m) = |\{a : 0 < a < m, \gcd(a, m) = 1\}|$ is the number of positive integers which are less than m and relatively co-prime to m.

Step 3. The CA then chooses a public number e_{CA} randomly in the range $1 < e_{CA} < m\text{-}1$with $\gcd(e_{CA}, \varphi(m)) = 1$, and computes the secret number d_{CA} such that $e_{CA}d_{CA} \equiv 1 \bmod \varphi(m)$; that is, $d_{CA} = e_{CA}^{-1} \bmod(\varphi(m))$.

Step 4. The CA finally removes the primes p and q, keeps secret the pair $(d_{CA}, \varphi(m))$ at the CA, and publishes the pair (e_{CA}, m) on the authenticated public domain.

■ Hierarchy construction phase

In this phase, the CA constructs a user hierarchy according to the user privileges in the set of security classes C such that $(C, \leq)$ forms a POSET. The following are steps needed to complete this phase successfully:

Step 1. For each security class C_i, the CA chooses a secret number s_i of 128 bits.

Step 2. The CA computes sub-secret key K_i as $K_i = H(\varphi(m) \| d_{CA} \| e_{CA} \| ID_i)$ for each security class C_i in the hierarchy, where ID_i is the unique identity of the security class C_i assigned by the CA.

Step 3. For each security class C_i, the CA computes the public parameter $M_{i,j}$ as $M_{i,j} = E_{K_i}(s_j)$ for all C_j, where $(C_i, C_j) \in R_{i,j}$.

Step 4. The CA then sends the sub-secret key K_i to each class C_i via a secure channel.

Step 5. Finally, the CA publishes all public parameters $M_{i,j}$ on the authenticated public domain. The CA then removes the sub-secret keys K_i and the secret numbers s_i for security reasons.

Example 1. Consider the hierarchy shown in Figure 1. According to the user privileges, the CA constructs the cryptographic relationships among all security classes in the hierarchy. First, the CA chooses secret numbers s_1, s_2, s_3, s_4, s_5, s_6, and s_7 for the security classes C_1, C_2, C_3, C_4, C_5, C_6, and C_7 respectively. After that, the CA computes the sub-secret key K_i and public parameters $M_{i,j}$ for each security class C_i, where $(C_i, C_j) \in R_{i,j}$ as follows:

C_1: CA computes sub-secret key $K_1 = H(\varphi(m) \| d_{CA} \| e_{CA} \| ID_i)$ and then public parameters $M_{1,1} = E_{K_1}(s_1)$, $M_{1,2} = E_{K_1}(s_2)$, $M_{1,3} = E_{K_1}(s_3)$, $M_{1,4} = E_{K_1}(s_4)$, $M_{1,5} = E_{K_1}(s_5)$, $M_{1,6} = E_{K_1}(s_6)$, $M_{1,7} = E_{K_1}(s_7)$.

C_2: CA computes sub-secret key $K_2 = H(\varphi(m) \| d_{CA} \| e_{CA} \| ID_2)$ and then public parameters $M_{2,2} = E_{K_2}(s_2)$, $M_{2,5} = E_{K_2}(s_5)$.

C_3: CA computes sub-secret key $K_3 = H(\varphi(m) \| d_{CA} \| e_{CA} \| ID_3)$ and then public parameters $M_{3,3} = E_{K_3}(s_3)$, $M_{3,5} = E_{K_3}(s_5)$, $M_{3,6} = E_{K_3}(s_6)$.

C_4: CA computes sub-secret key $K_4 = H(\varphi(m) \| d_{CA} \| e_{CA} \| ID_4)$ and then public parameters $M_{4,4} = E_{K_4}(s_4)$, $M_{4,6} = E_{K_4}(s_6)$, $M_{4,7} = E_{K_4}(s_7)$.

C_5: CA computes sub-secret key $K_5 = H(\varphi(m) \| d_{CA} \| e_{CA} \| ID_5)$ and then public parameters $M_{5,5} = E_{K_5}(s_5)$.

C_6: CA computes sub-secret key $K_6 = H(\varphi(m) \| d_{CA} \| e_{CA} \| ID_6)$ and then public parameters $M_{6,6} = E_{K_6}(s_6)$.

C_7: CA computes sub-secret key $K_7 = H(\varphi(m) \| d_{CA} \| e_{CA} \| ID_7)$ and then public parameters $M_{7,7} = E_{K_7}(s_7)$.

■ Key generation phase

In order to complete this phase the CA performs the following steps:

Step 1. The CA chooses a random secret key SK_i for each security class C_i and computes the signature $Sign_i$ of the secret key SK_i as $Sign_i = H(e_{CA} \| m \| SK_i \| ID_i)$.

Step 2. For each security class C_i, the CA computes the public parameter PK_i as $PK_i = [(SK_i \oplus H(Sign_i \| s_i \| ID_i)) \| (Sign_i)]^{d_{CA} \bmod \varphi(m)} \pmod m$ for sharing the secret key with the authorized security classes.

Step 3. Finally, the CA removes the secret keys SK_i for security reasons. The CA then publishes the public parameters PK_i on the authenticated public domain.

Example 2. Reconsider the hierarchy shown in Figure 1. The CA chooses secret keys SK_1, SK_2, SK_3, SK_4, SK_5, SK_6 and SK_7, and then computes the corresponding signatures $Sign_1$, $Sign_2$, $Sign_3$, $Sign_4$, $Sign_5$, $Sign_6$ and $Sign_7$ of the security classes C_1, C_2, C_3, C_4, C_5, C_6 and C_7, respectively, as $Sign_1 = H(e_{CA} \| m \| SK_1 \| ID_1)$, $Sign_2 = H(e_{CA} \| m \| SK_2 \| ID_2)$, $Sign_3 = H(e_{CA} \| m \| SK_3 \| ID_3)$, $Sign_4 = H(e_{CA} \| m \| SK_4 \| ID_4)$, $Sign_5 = H(e_{CA} \| m \| SK_5 \| ID_5)$, $Sign_6 = H(e_{CA} \| m \| SK_6 \| ID_6)$

and $Sign_7 = H(e_{CA} \| m \| SK_7 \| ID_7)$. The CA computes the public parameters $PK_1 = [(SK_1 \oplus H(Sign_1 \| s_1 \| ID_1)) \| (Sign_1)]^{d_{CA} \bmod \varphi(m)} \pmod m$, $PK_2 = [(SK_2 \oplus H(Sign_2 \| s_2 \| ID_2)) \| (Sign_2)]^{d_{CA} \bmod \varphi(m)} \pmod m$, $PK_3 = [(SK_3 \oplus H(Sign_3 \| s_3 \| ID_3)) \| (Sign_3)]^{d_{CA} \bmod \varphi(m)} \pmod m$, $PK_4 = [(SK_4 \oplus H(Sign_4 \| s_4 \| ID_4)) \| (Sign_4)]^{d_{CA} \bmod \varphi(m)} \pmod m$, $PK_5 = [(SK_5 \oplus H(Sign_5 \| s_5 \| ID_5)) \| (Sign_5)]^{d_{CA} \bmod \varphi(m)} \pmod m$, $PK_6 = [(SK_6 \oplus H(Sign_6 \| s_6 \| ID_6)) \| (Sign_6)]^{d_{CA} \bmod \varphi(m)} \pmod m$, and $PK_7 = [(SK_7 \oplus H(Sign_7 \| s_7 \| ID_7)) \| (Sign_7)]^{d_{CA} \bmod \varphi(m)} \pmod m$ for the security classes C_1, C_2, C_3, C_4, C_5, C_6, and C_7, respectively.

■ Key derivation phase

If the security class C_i wants to derive the secret key SK_j of its successor security class C_j, where $(C_i, C_j) \in R_{i,j}$, C_i needs to execute the following steps:

Step1. The security class C_i derives the secret number s_j using its own sub-secret key K_i as $s_j' = D_{K_i}(M_{i,j})$ and then derives $[(SK_j \oplus H(Sign_j \| s_j \| ID_j)) \| (Sign_j)]$ as $[(SK_j \oplus H(Sign_j \| s_j \| ID_j)) \| (Sign_j)] = PK_j^{e_{CA}} \bmod m$.

Step2. The security class C_i then computes the hash value $H(Sign_j \| s_j' \| ID_j)$ using derived secret number s_j' in Step1 and retrieves $Sign_j$. After that, C_i computes the secret key SK_j of the security class C_j as $SK_j' = (SK_j \oplus H(Sign_j \| s_j \| ID_j)) \oplus H(Sign_j \| s_j' \| ID_j)$. The CA further computes $H(e_{CA} \| m \| SK_j' \| ID_j)$ using the public e_{CA}, m and ID_j, and the derived secret key SK_j'. The CA then verifies whether the condition $Sign_j = H(e_{CA} \| m \| SK_j' \| ID_j)$ holds. If it holds, C_i confirms that the derived secret key SK_j is the legitimate secret key of C_j.

Solution to Dynamic Access Control Problem

In this section, we show that our scheme can efficiently support dynamic key management problems, including inserting a new security class into the hierarchy, deleting an existing security class, and changing the secret key of a security class.

■ Inserting a new security class into the hierarchy

Let a new security class C_x with the relationships $C_j \le C_x \le C_i$ be added into an existing user hierarchy. The CA executes the following steps in order to manage the access priority of C_x in the hierarchy:

Step 1. For the security class C_x, the CA chooses a random secret number s_x of 128 bits.

Step 2. The CA computes the sub-secret key K_x as $K_x = H(\varphi(m) \| d_{CA} \| e_{CA} \| ID_x)$ for the security class C_x.

Step 3. The CA computes public parameters $M_{x,y}$ for the security class C_x as $M_{x,y} = E_{K_x}(s_y)$ for all C_y, where $(C_x, C_y) \in R_{x,y}$ and public parameters $M_{p,x}$ for each predecessor class C_p of C_x as $M_{p,x} = E_{K_p}(s_x)$, where $(C_p, C_x) \in R_{p,x}$.

Step 4. The CA then sends the sub-secret key K_x for security class C_x via secure channel.

Step 5. The CA chooses a random secret key SK_x for security class C_x and computes the signature $Sign_x$ of secret key SK_x as $Sign_x = H(e_{CA} \| m \| SK_x \| ID_x)$. The CA then computes public parameter PK_x as $PK_x = [SK_x \oplus H(Sign_x \| s_x \| ID_x) \| Sign_x]^{d_{CA} \bmod \varphi(m)} \pmod m$.

Step 6. Finally, the CA removes secret key SK_x and secret number s_x for security reasons. The CA then publishes the public parameters PK_x, $M_{x,y}$ and $M_{p,x}$ on the authenticated public domain.

Example 3. Let a new security class C_8 be inserted into the user hierarchy given in Figure 1 such that $C_5 \le C_8 \le C_3$. The resulting hierarchy is shown in Figure 2. The CA first chooses the secret number s_8 and computes the public parameters for the predecessor security classes C_3 and C_1 of security class C_8 as $M_{3,8} = E_{K_3}(s_8)$ and $M_{1,8} = E_{K_1}(s_8)$, respectively, and the public parameter for security class C_8 as $M_{8,5} = E_{K_8}(s_5)$ where $(C_8, C_5) \in R_{5,8}$. The CA chooses secret key SK_8 of security class C_8 and computes signature $Sign_8$ for secret key SK_8 as $Sign_8 = H(e_{CA} \| m \| SK_8 \| ID_8)$ and then computes the public parameter PK_8 as $PK_8 = [(SK_8 \oplus H(Sign_8 \| s_8 \| ID_8)) \| (Sign_8)]^{d_{CA} \bmod \varphi(m)} \pmod m$. Finally, the CA removes secret number s_8 and secret key SK_8, and publishes public information $M_{3,8}$, $M_{1,8}$, $M_{8,5}$ and PK_8 on the authenticated public domain.

■ Deleting an existing security class from the hierarchy

Assume that an existing security class C_y must be deleted from the user hierarchy such that the relationship $C_j \le C_y \le C_i$ breaks up. The CA updates the access link of the security class C_y to terminate the access privileges of C_y over its successor. The CA performs the following steps in order to maintain the forward security of the successor classes:

Step 1. The CA removes all the parameters corresponding to the security class C_y.
Step 2. The CA renews secret number s_c and secret key SK_c for each successor security class C_c ($\neq C_y$) of the deleted security class C_y.
Step 3. The CA recomputes public parameters $M_{p,c}$ to all the predecessors of each successor security class C_c ($\neq C_y$) of deleted security class C_y. The CA then updates public parameter PK_c of each successor security class C_c of C_y with renewed secret number s_c and secret key SK_c.
Step 4. Finally, the CA publishes the public parameter on the authenticated public domain, and deletes all secret numbers s_c and secret keys SK_c for security reasons.

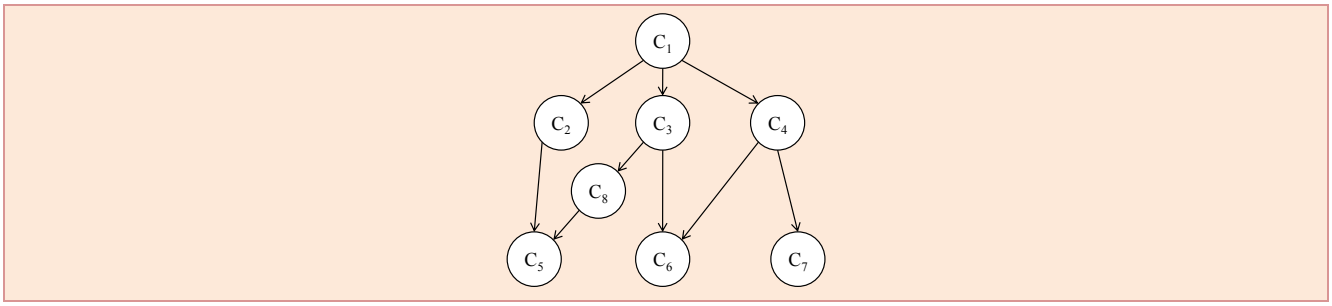

Figure 2. Hierarchy in Figure 1 after adding a new security class C_8

Example 4. Assume that existing security class C_3 is deleted from the hierarchy given in Figure 2; the resulting hierarchy is shown in Figure 3. The CA updates the hierarchy as follows: the CA reselects secret numbers s_8, s_6 and s_5, the secret keys SK_8, SK_6 and SK_5 of the successor security classes C_8, C_6 and C_5 of deleted security class C_3 as s_8', s_6' and s_5' for secret keys SK_8', SK_6' and SK_5', respectively. The CA then recomputes the public parameters as $M_{1,8} = E_{K_1}(s_8')$, $M_{1,6} = E_{K_1}(s_6')$, $M_{1,5} = E_{K_1}(s_5')$, and $M_{2,5} = E_{K_2}(s_5')$ with the newly generated secret numbers s_8', s_6' and s_5', respectively. Further, the CA computes the signatures on renewed secret keys SK_8', SK_6' and SK_5' as $Sign_8' = H(e_{CA} \| m \| SK_8' \| ID_8)$, $Sign_6' = H(e_{CA} \| m \| SK_6' \| ID_6)$ and $Sign_5' = H(e_{CA} \| m \| SK_5' \| ID_5)$, respectively, and then computes the public parameters PK_8, PK_6 and PK_5 as $PK_8 = [SK_8' \oplus H(Sign_8' \| s_8' \| ID_8) \| Sign_8']^{d_{CA} \bmod \varphi(m)} \pmod m$, $PK_6 = [SK_6' \oplus H(Sign_6' \| s_6' \| ID_8) \| Sign_6']^{d_{CA} \bmod \varphi(m)} \pmod m$, and $PK_5 = [SK_5' \oplus H(Sign_5' \| s_5' \| ID_8) \| Sign_5']^{d_{CA} \bmod \varphi(m)} \pmod m$ for security classes C_8, C_6 and C_5, respectively. The CA declares public parameters $M_{1,8}$, $M_{1,6}$, $M_{1,5}$, $M_{2,5}$, PK_8, PK_6 and PK_5 on the authenticated public domain and removes the secret numbers s_8', s_6', s_5' and secret keys SK_8', SK_6', SK_5' for security reasons.

■ Changing secret key of a security class in the hierarchy

Let security class C_i change its secret key, SK_i, for security reasons. The CA needs to update the corresponding public parameter PK_i. The CA generates a random secret key, say SK_i', and then computes its signature $Sign_i'$ as $Sign_i' = H(e_{CA} \| m \| SK_i' \| ID_i)$ and then computes public parameter PK_i' as $PK_i' = [(SK_i' \oplus H(Sign_i' \| s_i \| ID_i)) \| (Sign_i')]^{d_{CA} \bmod \varphi(m)} \pmod m$. Note that except for public parameter PK_i, all public data remains is unchanged in our scheme.

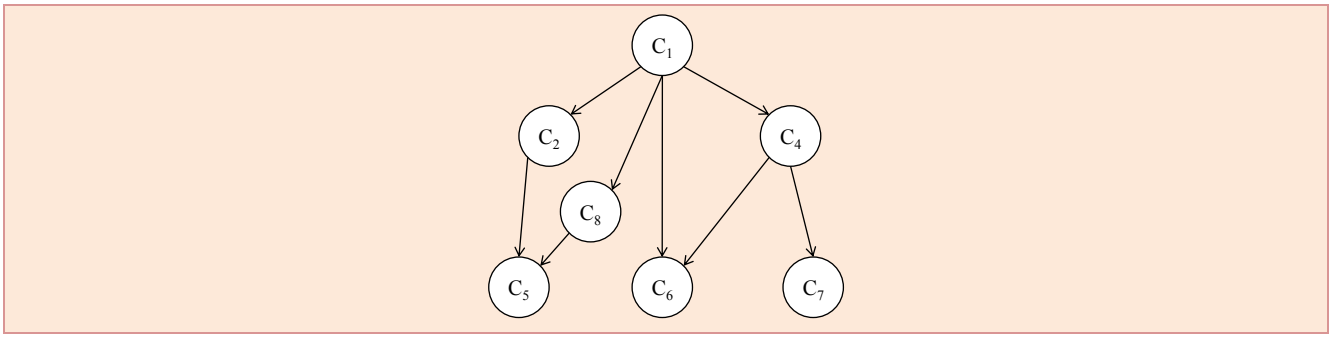

Figure 3. Hierarchy after deleting an existing security class C_3 in Figure 2

Example 5. Consider the hierarchy given in Figure 2. Suppose the security class C_3 wants to change its secret key SK_3. The CA generates a new secret key, SK_3', and computes its signature $Sign_3'$ as $Sign_3' = H(e_{CA} \| m \| SK_3' \| ID_3)$ and then computes the corresponding public parameter PK_3' as $PK_3' = [SK_3' \oplus H(Sign_3' \| s_3 \| ID_3) \| Sign_3']^{d_{CA} \bmod \varphi(m)} \pmod m$. The other public parameters remain unchanged.

Security Analysis of the Proposed Scheme

In this section we show that our scheme has the ability to tolerate the following known attacks required to design an idle access control scheme.

■ Contrary attack

In the contrary attack, a successor security class C_j, being an insider attacker, tries to derive the secret key SK_i of its predecessor security class C_i using the available public parameters and sub-secret key K_j. Note that in our scheme, C_j first needs to decrypt the public parameter $M_{i,i}$, where $(C_i, C_i) \in R_{i,j}$ in order to retrieve secret number s_i of security class C_i. After that, C_j computes secret key SK_i of predecessor security class C_i, but the task of decrypting $M_{i,j}$ without knowing the sub-secret key K_i of security class C_i is a computationally infeasible problem. Further, solving PK_i without the secret number s_i is again a computationally infeasible problem. Therefore, our scheme is secure against this attack.

■ Interior collecting attack

In this attack, a successor subordinated to *n* parents, say C_i, C_{i+1}, …, C_{i+n-1} tries to derive the secret key by collecting available public parameters. Since both the symmetric-key cryptosystem (AES-128) and hash function (SHA1) behave like random oracles, the task of deriving the secret number and then the secret key becomes a computationally infeasible problem. Hence, our scheme is secure against this attack.

■ Exterior collecting attack

In this attack, an external attacker tries to derive/compute the secret key from a lower level security class using the available public parameters in the authenticated public domain. However, due to the resistant property of the symmetric-key cryptosystem and hash function H(.), it is a computationally infeasible problem for the external attacker to retrieve the secret number and then the secret key. Therefore, our scheme resists the exterior collecting attack.

■ Collaborative attack

Here, several successor security classes, C_j, launch an attack in order to derive the secret key of its predecessor security class C_i. However, the security classes C_j require symmetric-key decryption to retrieve the secret number of C_i and then secret key SK_i. However, these are again computationally infeasible problems due to one-way and collision-resistant properties of the hash function H(.). As a result, our scheme is secure against this attack.

■ Forward security of successors while changing $C_j < C_x < C_i$ to $C_j < C_i$

Assume that the relationship $C_j < C_x < C_i$ is modified to another relationship $C_j < C_i$ after removing security class C_x from the hierarchy. In our scheme, the CA not only deletes the security class C_x, but it also updates the accessibility link relationship between classes C_i and C_j, removing all public parameters associated with C_x. Further, the CA renews secret number s_j and secret key SK_j and also updates the corresponding public parameters. Thus, the authority of security class C_x over C_j completely terminates, so that C_x will not have any further ability to retrieve the updated secret key SK_j of security class C_j. Therefore, the forward security of C_j is retained, and as a result, our scheme has the ability to preserve the forward security property.

■ Man in the middle attack

In a man in the middle attack, a malicious user can masquerade as the CA and publish some planned public parameters $M_{i,j}$ or PK_i on the authenticated public domain. Let a security class C_i use these public parameters to compute the secret key SK_j of its successor security class C_j. From our key derivation phase described in Section 2.4, note that security class C_i further computes the signature on the computed secret key SK_j and then verifies whether the computed signature matches the corresponding original signature $Sign_j$. If there is a mismatch, it suggests some attacker has changed the public data in the public domain. Usually, the public domain is considered write-protected. However, if an attacker has even the ability to update the public data in the public domain, the attacker will not be successful in changing the public data. Furthermore, the insider cannot change public data PK_i correctly without having private key d_{CA} of the CA, which is kept secret by the central authority. This is because our scheme provides the digital signature and the authentication mechanisms on secret keys in order to avoid such an attack. Hence, our scheme is also secure against the masquerade attack.

■ Formal security analysis of the proposed scheme

The following two formal definitions of indistinguishability of encryption–chosen plaintext attack (IND–CPA) and hash function, defined for the purpose of providing security for our new scheme in the following theorem.

Definition 1: IND–CPA Wu and Chen [16] describe IND and CPA as follows: Let *SE/ME* be the single/multiple eavesdropper and $O_{n_1}, O_{n_2}, \ldots, O_{n_N}$ be N different encryption oracles associated with secret keys $n_1, n_2, \ldots, n_N$. We define the advantage functions of *SE* and *ME* to be: $Adv_{\Omega,SE}^{IND-CPA}(l)$ $Adv_{\Omega,SE}^{IND-CPA}(l)$ = 2Pr [SE $\leftarrow O_{n_1}$; (m_0, $m_1 \leftarrow_R$ SE); $\theta \leftarrow_R$ {0, 1}; $\gamma \leftarrow_R O_{n_1}(m_\theta)$: SE ($\gamma$) = θ]-1, and $Adv_{\Omega,ME}^{IND-CPA}(l)$ = 2Pr [ME $\leftarrow O_{n_1}, O_{n_2}, \ldots, O_{n_N}$; ($m_0$, $m_1 \leftarrow_R$ ME); $\theta \leftarrow_R$ {0, 1}; $\gamma_1 \leftarrow_R O_{n_1}(m_\theta)$, $\gamma_2 \leftarrow_R O_{n_1}(m_\theta)$, $\gamma_2 \leftarrow_R O_{n_2}(m_\theta)$, …, $\gamma_N \leftarrow_R O_{n_N}(m_\theta)$: ME($\gamma_1, \gamma_2, \ldots, \gamma_N$) = θ]-1, respectively, where $\theta \leftarrow_R$ {0, 1} represents that the θ is a bit chosen randomly. Then we say the encryption scheme is IND–CPA secure in the single (multiple) eavesdropper setting if $Adv_{\Omega,SE}^{IND-CPA}(l)$ (and respectively, $Adv_{\Omega,ME}^{IND-CPA}(l)$) are negligible (in security parameter l) for any probabilistic, polynomial time (PPT) adversary *SE* (and respectively, *ME*).

Definition 2: Hash function There exists a secure one-way hash function H : X = $\{0,1\}^* \rightarrow Y^* = Z_p^*$ which satisfies the following requirements [19]:

- Given y ∈ Y, it is hard to find x ∈ X such that H(x) = y.
- Given x ∈ X, it is hard to find x′ ∈ X (x′ ≠ x) such that H(x′) = H(x).
- It is hard to find x, x′ ∈ X (x′ ≠ x) such that H(x′) = H(x).

Theorem 1: Assume that the used symmetric-key cryptosystem scheme *Ω* is IND–CPA secure. Then under the assumption that the hash function H(.) closely behaves like a random oracle, the proposed key management scheme is secure against an adversary deriving secret keys of any security class.

Proof: We follow the formal security proof of our scheme similar to that presented by Das et al. [10]. One can also follow the proof given in [22]. A hierarchical access control scheme for a POSET hierarchy is said to be secure if it is computationally infeasible for any attacker to derive the secret key of any class. For example, we choose an arbitrary security class $C_j \in C$ whose secret key is K_j, where C = {$C_1, C_2, \ldots, C_N$}. In our proof, we construct an adversary *A* such that it can derive the secret key K_j of C_j.

We now define the following two random oracles for the adversary *A*:

- **Reveal1:** This unconditionally outputs the input plaintext message M using the symmetric cryptosystem *Ω* using the corresponding public parameters and given ciphertext message $E_k(M)$, without knowing the symmetric key *k*.
- **Reveal2:** This random oracle unconditionally outputs the input string *x* from the corresponding hash value *y = H(x)*.

Adversary *A* needs to run the following two experimental algorithms $Exp1_{HACP,A}^{IND-CPA}$ and $Exp2_{HACP,A}^{HASH}$ for our proposed hierarchical access control scheme, say HACP provided in Algorithm 1 and Algorithm 2, respectively.

Algorithm 1 $Exp1_{HACP,A}^{IND-CPA}$

1. Read public parameter $M_{i,j} = E_{K_i}(s_j)$ from the public domain.
2. Call *Reveal1* oracle on input $M_{i,j}$. Let $s_j' \leftarrow Reveal1(M_{i,j})$.
3. Read the public parameter PK_j from the public domain. Using the public key e_{CA} of the CA, compute $PK_j^{e_{CA}}$ (mod m) = ($SK_j \oplus H(Sign_j \| s_j \| ID_j)) \| Sign_j$.
4. Using the derived secret number s_j' and the signature $Sign_j$, compute $H(Sign_j \| s_j' \| ID_j)$ and the secret key SK_j'= ($SK_j \oplus H(Sign_j \| s_j \| ID_j)) \oplus H(Sign_j \| s_j' \| ID_j)$.

5. Compute the hash value $H(e_{CA} \| m \| SK_j' \| ID_j)$ and compare it with the derived signature $Sign_j$. If it matches, return 1(true); else return 0 (false).

We define $Succ_{HACP,A}^{IND-CPA} = Pr[Exp1_{HACP,A}^{IND-CPA} = 1] - 1$. Then the advantage function for $Exp1_{HACP,A}^{IND-CPA}$ is defined by $Adv_{HACP,A}^{IND-CPA}(t_1, q_{R_1}) = \max_A Succ_{HACP,A}^{IND-CPA}$, where the maximum is taken over all A with execution time t1 and the number of queries q_{R_1} to the Reveal1 oracle. We say that our scheme is secure against deriving the secret key Kj of any security class Cj if $Adv_{HACP,A}^{IND-CPA}(t_1, q_{R_1}) \leq \varepsilon$, for any sufficiently small $\varepsilon > 0$.

Algorithm 2 $Exp2_{HACP,A}^{HASH}$

1. Read the public parameter PK_j from the public domain. Using public key e_{CA} of the CA, compute $PK_j^{e_{CA}} \pmod m = (SK_j \oplus H(Sign_j \| s_j \| ID_j)) \| Sign_j$.
2. Call *Reveal2* oracle on input $Sign_j$. Let $SK_j' \leftarrow Reveal2(Sign_j)$.
3. Call *Reveal2* oracle on input $H(Sign_j \| s_j \| ID_j)$. Let $Sign_j' \leftarrow Reveal2(H(Sign_j \| s_j \| ID_j))$.
4. Check the condition $Sign_j' = Sign_j$. If it holds, return 1(true); else return 0 (false).

Similar to $Succ_{HACP,A}^{IND-CPA}$, we also define $Succ_{HACP,A}^{HASH} = Pr[Exp2_{HACP,A}^{HASH} = 1] - 1$. Then the advantage function for $Exp2_{HACP,A}^{HASH}$ becomes $Adv_{HACP,A}^{HASH}(t_2, q_{R_2}) = \max_A Succ_{HACP,A}^{HASH}$, where the maximum is taken over all A with execution time t_2 and the number of queries q_{R_2} to the *Reveal2* oracle. Our scheme is said to be secure against deriving the secret key SK_j of any security class C_j, if $Adv_{HACP,A}^{HASH}(t_2, q_{R_2}) \leq \varepsilon$, for any sufficiently small $\varepsilon > 0$.

Consider the above experiment, $Exp1_{HACP,A}^{IND-CPA}$ in Algorithm 1. If the attacker can decrypt $M_{i,j}$ successfully without knowing the key, then he/she can derive the secret key SK_j of C_j ($C_i > C_j$). However, it is a computationally infeasible problem since Ω is IND–CPA secure given Definition 1. Since $Adv_{\Omega,SE}^{IND-CPA}(l)$ (and respectively, $Adv_{\Omega,ME}^{IND-CPA}(l)$) is negligible (in the security parameter l) for any probabilistic, polynomial time (PPT) adversary SE (and respectively, ME), so $Adv_{HACP,A}^{IND-CPA}(t_1, q_{R_1})$ is also negligible as it is dependent on the former.

We now consider another experiment, $Exp2_{HACP,A}^{HASH}$ given in Algorithm 2, conducted by an adversary. If the adversary has the ability to solve the hash function problem provided in Definition 2, then the adversary can directly derive secret key SK_j of security class C_j ($C_i > C_j$). Since the problem is deriving the input string from the given hash value, it is computationally infeasible to derive the secret key SK_j as the hash function H(.) closely behaves like a random oracle. Thus, $Adv_{HACP,A}^{HASH}(t_2, q_{R_2})$ is negligible. As a result, no adversary is able to derive the secret key of any security class.

Performance Analysis of Our Scheme

In this section, we analyze the correctness of our key derivation phase, storage space and computational cost.

■ Correctness of key derivation phase

Theorem 2 provides the correctness proof of our key derivation phase.

Theorem 2. Assume that there are two security classes C_i and C_j with the relationship $(C_i, C_j) \in R_{i,j}$ in a POSET hierarchy. The security class C_i can then derive secret key SK_j of C_j using its own sub-secret key K_i and the available public parameters in the authenticated public domain.

Proof: The security class C_i derives secret key SK_j of security class C_j using the sub-secret key K_i as follows:

$$s_j' = D_{K_i}(M_{i,j})$$

$$(SK_j \oplus H(Sign_j \| s_j \| ID_j)) \| (Sign_j) = PK_j^{e_{CA}} \pmod m$$

Note that

$$PK_j^{e_{CA}} \pmod m = \{[SK_j \oplus H(Sign_j \| s_j \| ID_j) \| Sign_j]^{d_{CA} \bmod \varphi(m)} \pmod m\}^{e_{CA}} \pmod m$$

$$= [SK_j \oplus H(Sign_j \| s_j \| ID_j) \| Sign_j]^{d_{CA} e_{CA} \bmod \varphi(m)} \pmod m$$

$$= SK_j \oplus H(Sign_j \| s_j \| ID_j) \| Sign_j, \text{ since } d_{CA} = e_{CA}^{-1} \bmod (\varphi(m)).$$

The bit string $(SK_j \oplus H(Sign_j \| s_j \| ID_j)) \| (Sign_j)$ is assumed to be 320 bits, since the hash output of the secure hash function (for example SHA-1) is a 160-bit string and the string $(SK_i \oplus H(Sign_j \| s_j \| ID_j)) \| (Sign_j)$ is a concatenation of

two strings ($SK_j \oplus H(Sign_j \| s_j \| ID_j)$) and ($Sign_j$), each 160-bit strings. Therefore, the first (left-most) 160 bits of $PK_j^{e_{CA}} \pmod m$ represents the string ($SK_j \oplus H(Sign_j \| s_j \| ID_j)$) and the second (right-most) 160 bits of $PK_j^{e_{CA}} \pmod m$ represents the string ($Sign_j$).

Now the secret key SK_j' is computed using these two strings and the derived secret number s_j' of security class C_j as $SK_j' = (SK_j \oplus H(Sign_j \| s_j \| ID_j)) \oplus H(Sign_j \| s_j' \| ID_j)$. The security class C_i then verifies the validity of derived secret key SK_j' as follows. If the condition $Sign_j = H(e_{CA} \| m \| SK_j' \| ID_j)$ holds, the security class C_i confirms that the derived secret key SK_j' is legitimate and hence $SK_j' = SK_j$.

■ Storage overhead

In our scheme, the CA needs to store the private key pair $(d_{CA}, \varphi(m))$ and the public key pair (e_{CA}, m). We consider the hierarchy with N security classes, say $C_1, C_2, \ldots, C_N$, and each security class C_i has v_i successors. The private storage for each security class is 128 bits to store its own sub-secret key K_i. The storage space required for the hierarchy in our scheme is $128 \sum_{i=1}^{N} v_i$, and the storage space for public parameter PK_i requires $1024N$ bits. Therefore, the total public storage space required in our scheme is $(128 \sum_{i=1}^{N} v_i + 1024N + 2048)$ bits.

■ Computational overhead

Let T_{exp} and T_{inv} denote the time complexity of executing the modular exponent, modular inverse in the finite field $GF(2^{163})$. T_{SHA1} denotes the time complexity of hashing a 512-bit l-message block using hash function SHA-1 [18], T_{AES} denotes the time complexity of encrypting or decrypting a 128-bit message block using AES [17] with a 128-bit key, and T_{XOR} denotes the time complexity for executing a bitwise exclusive OR operation.

Our scheme requires one T_{inv} for computing private key d_{CA} of the CA and $N.T_{SHA1}$ for computing sub-secret keys of N security classes in the hierarchy, hierarchy construction requires $\sum_{i=1}^{N} v_i . T_{AES}$, and the key generation phase requires $N.(T_{SHA1} + T_e + T_{XOR})$ operations, whereas the key derivation phase requires $\sum_{i=1}^{N} v_i (T_e + T_{AES} + 2T_{SHA1} + T_{XOR})$ operations. The total computational complexity require for our scheme is $2 \sum_{i=1}^{N} v_i . T_{AES} + \sum_{i=1}^{N} v_i (T_e + 2T_{SHA1} + T_{XOR}) + N.(T_{SHA1} + T_e + T_{XOR})$.

Performance Comparison with Related Schemes

In this section, we compare the performance and security of our scheme against closely related existing schemes: the Akl-Taylor scheme [1], the Harn-Lin scheme [3] and the Hwang-Yang scheme [4] because they are based on the remote secure access public key cryptosystem. In this paper, we omit other schemes for comparison purpose, since they are not based on RSA cryptosystem.

Table 1. The computational complexity of different phases in the related scheme

Scheme	Adding security class	Deleting security class	Changing secret key	Key generation	Key derivation
Akl-Taylor	O(n)	O(n)	Not supported	O(n)	O(1)
Harn-Lin	O(l)	O(1)	Not supported	O(n)	O(1)
Hwang-Yang	O(1)	O(1)	Not supported	O(y)	O(1)
Ours	O(l)	O(l)	O(1)	O(1)	O(1)

Table 1 shows five aspects of the computational complexity, which are for adding a security class, removing a security class, changing secret keys, key generation, and key derivation. The computational cost of adding a security class is required to analyze the number of security classes needed to update secret keys when a new security class is added into POSET hierarchy. The computational cost of deleting a security class is required to analyze the number of security classes needed to update secret keys when removing an existing security class from a POSET hierarchy. Changing secret keys is the most important property of a dynamic hierarchy because it affects the security of the system. From Table 1, the Akl-Taylor scheme, Harn-Lin scheme and Hwang-Yang scheme do not support the dynamic property of changing the secret key of a security class in a hierarchy. However, our proposed scheme provides an efficient solution to the key-change property. The computational complexity for both adding a security class and deleting a security class is O(1) only in Hwang-Yang's scheme. However, Wang-Laih [5] showed that Hwang-Yang's scheme is vulnerable to the collusion attack.

In Table 2, we show the functionality of different related schemes and our scheme. From this table, it is clear that our scheme efficiently solves the dynamic access problem compared to other schemes.
From Table 3, we see that our scheme provides a digital signature and also prevents active attacks such as the "man-in-the-middle" attack. Further, our scheme is secure against all possible known attacks.

Table 2. The Functionalities of different phases in the related schemes

Scheme	Adding security class	Deleting security class	Changing secret key
Akl-Taylor	Not supported	Not supported	Not supported
Harn-Lin	Update partial hierarchy	Update partial hierarchy	Update partial hierarchy
Hwang-Yang	Update partial hierarchy	Update partial hierarchy	Update partial hierarchy
Ours	Update partial hierarchy	Update partial hierarchy	Do not update the hierarchy

where n represents number of security class in the hierarchy, l represents levels of the hierarchy and $y = \sum_{for\ all\ LG_i} g_i + n_a + n_t$, where g_i is the minimum number such that $\binom{g_i}{k} \geq h_i$, k is the number of primes for distinguishing leaf classes in a leaf group and $\binom{g_i}{k}$ means the number of ways for choosing k items from g_i and h_i denotes the number of leaf classes in LG_i, n_a denotes the number of leaf security classes that has more than one parents and n_t denotes the number of non-leaf security classes.

Table 3. The security comparison of related schemes

Scheme	Whether provides digital signature to the secret key	Whether prevents "man-in-the-middle" attack
Akl-Taylor	No	No
Harn-Lin	No	No
Hwang-Yang	No	No
Ours	Yes	Yes

Conclusions

In this paper, we proposed a new key management scheme in a user hierarchy. Our scheme provides an efficient solution to the problem of changing secret keys of a security class in the hierarchy. In addition, our scheme has the ability to efficiently support other dynamic properties, such as adding a new security class to the hierarchy and deleting an existing security class from the hierarchy. Through formal security analysis as well as informal security analysis, we have shown that our scheme has the ability to tolerate known attacks. Moreover, our scheme is efficient compared with the other approaches. Considering the higher security and efficiency in solving dynamic access control problems, our scheme is more suitable for practical applications. As the applications of the proposed scheme are concerned, the proposed scheme can be also used for ad-doc communications or infrastructure communications.

References

[1] S. G. Akl and P. D. Taylor, "Cryptographic solution to a problem of access control in a hierarchy," *ACM Transactions on Computer Systems* (*TOCS*), vol. 1, no. 3, pp. 239–248, Aug. 1983. Article (CrossRef Link)
[2] S. Wang and C. Laih, "Merging: An Efficient Solution for a Time-Bound Hierarchical Key Assignment Scheme," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 1, pp. 91–100, Jan.-Mar. 2006. Article (CrossRef Link)
[3] L. Harn and H. Y. Lin, "A cryptographic key generation scheme for multilevel data security," *Computers & Security*, vol. 9, no. 6, pp. 539–546, 1990. Article (CrossRef Link)
[4] M.-S. Hwang and W.-P. Yang, "Controlling access in large partially ordered hierarchies using cryptographic keys," *Journal of Systems and Software*, vol. 67, no. 2, pp. 99–107, 2003. Article (CrossRef Link)
[5] S.-Y. Wang and C.-S. Laih, "Cryptanalysis of Hwang-Yang scheme for controlling access in large partially ordered hierarchies," *Journal of Systems and Software*, vol. 75, pp. 189–192, 2005. Article (CrossRef Link)
[6] J.-W. Lo, M.-S. Hwang and C.-H. Liu, "An efficient key assignment scheme for access control in a large leaf class hierarchy," *Information Sciences*, vol. 181, pp. 917–925, 2011. Article (CrossRef Link)

[7] R. S. Sandhu, "Cryptographic implementation of a tree hierarchy for access control," *Information Processing Letters*, vol. 27, no. 2, pp. 95–98, Feb. 1988. Article (CrossRef Link)
[8] C. Yang and C. Li, "Access control in a hierarchy using one-way hash functions," Computers & Security, vol. 23, no. 8, pp. 659–664, 2004. Article (CrossRef Link)
[9] Y.F. Chung, H.H. Lee, F. Lai, and T. S. Chen, "Access control in user hierarchy based on elliptic curve cryptosystem," *Information Sciences*, vol. 178, no. 1, pp. 230–243, 2008. Article (CrossRef Link)
[10] A. K. Das, N. R. Paul, and L. Tripathy, "Cryptanalysis and improvement of an access control in user hierarchy based on elliptic curve cryptosystem," *Information Sciences*, vol. 209, pp. 80–92, 2012. Article (CrossRef Link)
[11] F.G. Jeng and C.M. Wang, "An efficient key-management scheme for hierarchical access control based on elliptic curve cryptosystem," *Journal of Systems and Software*, vol. 79, no. 8, pp. 1161–1167, 2006. Article (CrossRef Link)
[12] Y.-L. Lin and C.-L. Hsu, "Secure key management scheme for dynamic hierarchical access control based on ECC," *Journal of Systems and Software*, vol. 84, no. 4, pp. 679–685, 2011. Article (CrossRef Link)
[13] M. Nikooghadam and A. Zakerolhosseini, "Secure Communication of Medical Information using Mobile Agents," *Journal of Medical Systems*, vol. 36, no. 6, pp. 3839-3850, 2012. Article (CrossRef Link)
[14] M. Nikooghadam, A. Zakerolhosseini, and M. E. Moghaddam, "Efficient utilization of elliptic curve cryptosystem for hierarchical access control," *Journal of Systems and Software*, vol. 83, no. 10, pp. 1917–1929, 2010. Article (CrossRef Link)
[15] S.-F. Tzeng, C.-C. Lee and T.-C. Lin, "A novel key management scheme for dynamic access control in a hierarchy," *International Journal of Network Security*, vol. 12, no. 3, pp. 178–180, 2011.
[16] S. Wu, and K. Chen, "An efficient key-management scheme for hierarchical access control in E-Medicine system," *J. Med. Syst.*, vol. 36, no. 4, pp. 2325-2337, 2012. Article (CrossRef Link)
[17] Advanced Encryption Standard, FIPS PUB 197, National Institute of Standards and Technology (NIST), U.S. Department of Commerce, November 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.
[18] Secure Hash Standard. FIPS PUB 180-1, National Institute of Standards and Technology (NIST), U.S. Department of Commerce, 1995.
[19] W. Stallings, "Cryptography and network security: principles and practices," 3rd edition, Englewood Cliffs, NJ: Prentice Hall, 2003.
[20] M. Agrawal, N. Kayal and N. Saxena, "PRIMES IS IN P," Annals of Mathematics, vol. 160, no. 2, pp. 781–793, 2004. Article (CrossRef Link)
[21] C. K. Wong, M. G. Gouda, and S. S. Lam, "Secure Group Communications Using Key Graphs," *IEEE/ACM Transactions on Networking*, vol. 8, no. 1, pp. 16-30, 2000. Article (CrossRef Link)
[22] J. H. Lee and J. M. Bonnin, "HOTA: Handover optimized ticket-based authentication in network-based mobility management," *Information Sciences*, vol. 230, pp. 64-77, 2013. Article (CrossRef Link)

Vanga Odelu received his M.Tech. degree in Computer Science and Data Processing from the Indian Institute of Technology, Kharagpur, India, on July 2011. His research interests include cryptography, network security and hierarchical access control. He has published 6 papers in international journals and conferences in the area of hierarchical access control.

Ashok Kumar Das is currently working as an Assistant Professor in the Center for Security, Theory and Algorithmic Research of the International Institute of Information Technology (IIIT), Hyderabad 500 032, India. He received his Ph.D. degree in Computer Science and Engineering, M.Tech. degree in Computer Science and Data Processing, and M.Sc. degree in Mathematics, all from the Indian Institute of Technology, Kharagpur, India, on April 2009, January 2000 and July 1998, respectively. His current research interests include cryptography, wireless sensor network security, proxy signature, hierarchical access control and remote user authentication. He has published 40 papers in international journals and conferences in these areas.

Adrijit Goswami received his MSc and PhD degree from Jadavpur University, India, in 1985 and 1992 respectively. In 1992, he joined the Indian Institute of Technology, Kharagpur, India, where at present he is Professor in the Mathematics Department. He has published articles in JORS, EJOR, International Journal of Production Economics, Computers and Mathematics with Applications, Productional Planning and Control, Opsearch, International Journal of Systems Science, The Journal of Fuzzy Mathematics, Journal of Information & Knowledge Management, Expert systems, International Journal of Uncertainty, Fuzziness and Knowledge-based Systems, Journal of Applied Mathematics & Computing, International Journal of Computers & Applications, International Journal of Data Analysis Techniques and Strategies etc. His research interest includes inventory management under fuzzy environment, optimization, database systems, distributed and object-oriented databases, data mining techniques under fuzzy environment, and information security.