An efficient key management scheme for content access control for linear hierarchies

Hani Ragab Hassen, Hatem Bettahar, Abdelmadjid Bouabdallah, Yacine Challal

An Efficient Key Management Scheme for Content Access Control for Linear Hierarchies

Hani Ragab Hassen

University of Kent, UK

Hatem Bettahar, Abdalmadjid Bouadbdallah, Yacine Challal

University of Technology of Compiegne, France

Abstract

The content access control problem appears in any context with a set of users and resources. The difference in access rights of the users defines classes where members of a given class have exactly the same access rights. A hierarchy can be defined on the classes. Linear hierarchies constitute a particulary interesting type of hierarchies. They appear in a wide range of applications such as secure multi-layered data streaming and communications within security corps. Many proposals have dealt with key management issues for tree hierarchies but they result in unjustified overhead when applied to linear hierarchies.

In this paper, we discuss the general problem of content access control in a hierarchy (CACH). Thereafter, we present the main requirements in key management to ensure confidentiality in linear hierarchies. In particular, we define a model to make a uniform and coherent description of the existing key management schemes. Thereafter, we propose an efficient key management scheme for linear hierarchies that not only provides mechanisms to manage membership changes but also hierarchy shape changes, and we describe it using our model. We conduct intensive simulations which show that our solution scales very well in terms of storage, bandwidth, and computation. Finally, we determine the complexity of some well-known key management schemes and compare them to the complexity of our scheme. This comparison shows that our scheme offers efficient compromises in complexity and overall overheads.

Keywords: Content access control, Group communication, Key management, Linear hierarchies, Network security.

Email addresses: h.ragab@kent.ac.uk (Hani Ragab Hassen), hatem.bettahar@utc.fr (Hatem Bettahar), bouabdal@utc.fr (Abdalmadjid Bouadbdallah), yacine.challal@utc.fr (Yacine Challal)

1. Introduction

Providing content access control is a critical security issue within hierarchies. A hierarchy is defined using a set of differentiating access rights (privileges), and each entity in the hierarchy has a subset of these access rights. Content access control consists of ensuring that users access only items to which they are entitled. The need for ensuring the Content Access Control in Hierarchies (CACH) appears in all hierarchically organized establishments, ranging from government departments to business corporations.

CACH is required in any context where a set of sensitive information items should be accessed with different access rights, in the sense that some members have access to particular items while others have not. For example, consider a multi-layered video streaming application with a basic layer quality, and N Enhancement Layers (EL). A user who paid for enhancement layer EL_i should be able to access enhancement layers of lower quality , but not to enhancement layers of better quality. Ensuring content access control in this example can be carried out by encrypting the different enhancement layers using different keys. The keys are called Content Encryption Keys (CEK). Each user will get the key of the enhancement layer to which they are entitled, as well as (some means to get) the keys for the enhancement layers of lower quality. Key management for CACH is about how to generate these keys, distribute them and renew them efficiently while ensuring the security requirements of content access control [1].

Key management for CACH is a challenging security problem. The key management problem has already been addressed for flat groups. In a flat group, there is only one data stream, which means that a member can either access all data or nothing. Adding a hierarchy to a group communication makes the key management much more complicated than the simple case of a flat group. In a hierarchy, there are more than one data stream, encrypted using different keys. Therefore a member having a given subset of keys is authorized to access corresponding data streams and not to others.

In this paper, we present a new and efficient key management scheme and we evaluate it. The paper is organized as follows. In the next section, we introduce the material required to understand the problem of CACH and its key management issues, and give an overview of the existing key management schemes. Section 3 presents our key management scheme. Simulation model and results are presented in section 4. In section 5, we study and compare the complexity of our scheme to the complexity we computed for some widely-known key management schemes. We present our conclusions in section 6.

2. Content Access Control in Hierarchies

This section introduces the key management for CACH. We start by giving a content access control model that we will use to describe both the existing key management schemes and our new scheme. We then classify hierarchies into three categories according to their shapes. Finally, we discuss main key management issues for hierarchies.

2.1. Security Classes Model

A hierarchy can be defined in two steps. Firstly, the entities within the system are divided into subgroups according to their access rights. Secondly, an ordering relation is defined on the subgroups. Ordering the subgroups is based on the access rights. These access rights allow to say that one subgroup is more privileged compared to another, thereby defining the hierarchy.

An access right statement gives one entity access to another. We distinguish two categories of entities: *subjects* and *objects*. A subject can access an object. That is, subjects are active entities (users, processes, ...), while objects are passive entities (messages, documents, system resources, ...). We give below an example of subjects and objects:

Example 2.1 *Group Communications in a Hierarchy*
Ensuring content access control is a critical issue within public security corps communications. For example, in a deployed military troop, commanders should be able to access communications within lower classes. The inverse should not be possible.

Subjects: senders and receivers within the system;

Objects: exchanged messages (voice, mail, ...).

We denote by S the set of all subjects, by O the set of all objects, , and by E the set of all the entities in the system. That is, $E = S \cup O$. We write $s \succeq o$ to say that subject s has access to object o. The set of entities E is organized into *security classes (SC)*. A security class contains subjects having exactly the same access rights. The operator '$\succeq$' can be easily extended to security classes. So we say that a security class SC_i covers (or dominates) another security class SC_j and denote it by $SC_i \succeq SC_j$ if and only if any subject in SC_i has access to all subjects in SC_j

In addition to subjects, each security class SC_i contains all objects that are only accessible by subjects in SC_i (and, of course, higher security classes SC_k).This means that members of a given class have access to objects of their own class, and all lower classes. We define as well the relation '$\succ$' by :

$$SC_i \succ SC_j \Leftrightarrow SC_i \succeq SC_j \text{ and } SC_i \neq SC_j.$$

Given two security classes SC_i and $SC_j \in SC$, such that $SC_i \succ SC_j$, we say that SC_i is an *ancestor* of SC_j, and SC_j is a *descendant* of SC_i. If, in addition, SC_i is a direct ancestor of SC_j, i.e there is no other class $SC_k \in SC$ such that $SC_i \succ SC_k$ and $SC_k \succ SC_j$, then SC_i is a *parent* of SC_j, and SC_j is a *child* of SC_i. In figure 1, SC_1 is an ancestor of SC_7 and is a parent of SC_2. SC_4 is a descendant of SC_1 and is a child of SC_2.

2.2. Hierarchy Categories

Hierarchies can be classified into three categories according to the hierarchy shape [1]. A very interesting type of hierarchy is the linear hierarchies (LH). In such a hierarchy, security classes form a directed chain. The first class in

the chain is the most privileged. Indexes can be assigned such that $i < j \Leftrightarrow SC_i \succ SC_j$. That is, SC_1 is the highest class. An example of a linear hierarchy is given in figure 2.a. As shown in figure 2.b, contrary to the linear hierarchy where each element in the hierarchy has only one parent (direct predecessor) and one child (direct successor), tree hierarchies allow elements to have many children. In a DAG (Directed Acyclic Graph) hierarchy (cf. 2.c), each security class has many children and parents.

In this paper, we will focus on linear hierarchies. Typical examples of linear hierarchies are military communications where an officer has access to communications within lower ranks, but not to higher ones. Multi-layered video streaming is another example of linear hierarchies. A user can access enhancement layers to which their subscribed while not to higher ones. In the rest of this section, we review the key management issues for content access control in hierarchies.

2.3. Key Management Issues

The existing key management schemes model the hierarchy as a group of communicating members (subjects) [1]. In this context the objects are the exchanged messages. Key management typical tasks are content encryption keys (CEK) generation and renewal. The CEK renewal is necessary to ensure confidentiality when subjects (members) join or leave the hierarchy (group). That is, a new member must be prevented from accessing data (objects) exchanged before they join the group. This requirement is called *backward secrecy*. Inversely, a leaving member must not have access to data exchanged after they leave the session (*forward secrecy* requirement). A renewal should be carried out after their departure.

Many solutions have been proposed in the group communication literature to tackle the problem of key management within flat groups (i.e. groups with only one data stream). The proposed protocols can be divided mainly into three categories: centralized, decentralized and distributed architectures. In the centralized approach [2, 3, 4, 5, 6], a logical entity, namely the Group Controller and Key Server ($GCKS$) provides key management services (key generation, distribution and rekeying) [7]. In the decentralized approach [8, 9, 10, 11, 12, 13, 14], a set of managers share the labor of key management. Finally, in the distributed approach [15, 16, 17, 18, 19] a set of managers collaborate to agree on a group key and distribute the key material.

Most of the solutions proposed to meet the CACH requirements in key management are centralized solutions [1]. Key trees are widely used in centralized approach, and even in other approaches. This is mainly due to their well-known good performances [20]. A key tree is a graph with no cycles, in which keys are represented by nodes. Leaves are user individual keys. The root key is usually the CEK. All other keys are used to encrypt the rekeying material, and are so-called key encryption keys *KEK*.

The keys used to encrypt a given object should be renewed each time that a subject changes its access rights to the object. An event that causes a key

renewal is called *rekeying trigger*. Rekeying triggers will be further discussed in section 3.

2.4. Existing Key Management Schemes

We divide the existing key management schemes for CACH into two approaches: the *dependent-keys*, and the *independent-keys* approaches [1] as shown in figure 3. The first approach is issued from information systems (file systems, databases, ...) community. Earlier works in key management for hierarchies [21, 22, 23] were done in this community.

In the dependent-keys approach, in order to access a given information item (object), a legitimate user does not need to have the key with which it is encrypted. But using merely their own key, combined with some public parameters and/or functions, they can compute the key used to decrypt the item. The resulting key management schemes typically use complex cryptographic techniques. We further distinguish two categories in the dependent-keys schemes: indirect access schemes and direct access schemes. In indirect access schemes, if SC_i is an ancestor of SC_j, members of SC_i must compute all intermediate keys on the path from SC_i to SC_j to get the key of SC_j. Sandhu [24, 23], Gudes [21], Yang and Li [25], He et al. [26], Wang et al. [27], and Gawdan et al. [28] proposed indirect schemes that are based on one-way functions. In such schemes, the key of a class is computed from the key of its parent using a one-way function.

A direct scheme avoids the computation of intermediate keys, using additional public parameters. Most of the schemes proposed in this approach use prime numbers fundamental properties (Akl and Taylor [22], Ray et al. [29], Zou et al. [30]), and other theoretical cryptographic notions, including Chinese reminder theorem (Hardjono et al. [31] and Zheng et al.[32]). Shen and Chen [33], Zhang et al. [34], Tzang et al. [35] and Das et al. [36] use Newton's polynomial interpolation to correlate classes' indices and their keys. Whereas Aparna et al. [37] use threshold cryptography to reduce the cost of key renewal, and Liu et al. [38] use elliptic curves. In these schemes, the key of a class can be directly computed using the key of any of its ancestors combined with the parameters defined by the scheme.

The independent keys approach originates from multicast security community. More precisely, from works on key management. Thus these schemes use usual key trees and graphs techniques [20]. In order to access some information item, the user should have a copy of its decryption key.

Contrary to the dependent key approach, where the principle is to divide accessible resources and users into security classes and study the relations existing between the different classes, the independent key approach separates users from accessible resources, and studies relations between users and resources. From the resource point of view, users having access to a given resource r_i form a *resource group* RG_i. From the users point of view, users having exactly access to the same subset of resources form a *users service group* (*service group* for abbreviation).

In [**?**], Sun and Liu proposed to assign a key to each resource group and to each service group, and then use relations between resource groups and service groups to reduce the global number of used keys. Their solution takes benefits from the fact that nodes in a key tree maintain all keys of intermediate nodes on the path to the root. Ma et al. [39] associate to each resource r_i a key K_i^r. Each user owns a subset of resource keys which is called *access key (AK)*. This subset contains keys of all resources in the service group. Users in the same service group SG_i are arranged in a key tree, whose root is the set of keys AK_i of SG_i. In [40], Karandikar et al. suggest to maintain a resource list RL_i for each resource r_i. They use a technique for secure group conferencing which was proposed in [41]. It consists on building a centralized key tree which contains all users in leaves and uses intermediate keys in this tree to manage access to resources.

2.5. Discussion

Independent-keys schemes are quite simple to deploy. However, they do not offer an efficient support for hierarchy changes. On the other hand, dependent-keys schemes have the advantage of minimizing the number of keys maintained by users. This is carried out using well chosen public parameters. However, in most of the schemes proposed in this approach, number of public parameters is quite important. Which obsoletes the idea of reducing storage on user's side. One-way functions based techniques seem to be a good solution for storage overhead issues: relating keys using such functions allows to replace a set of keys by only one key. Nevertheless, changing one key in a set of related keys implies the renewal of the whole set. This generates an additional renewal overhead. In the next section we propose an improved one-way function-based solution that allows to avoid this overhead by using a key table mechanism.

Notations

Table 1 summarizes some important notations that we will use for key management schemes description and comparison.

3. Our Approach

Linear hierarchies are commonplace in communications within public security organisms and multilayered data streaming. A typical example of linear hierarchies is operational military battalions. As shown in figure 5, members communicate with others from the same class, and thereby should have the same content encryption key CEK. Furthermore, commanders should be able to supervise their subordinates, and hence need to know their respective CEKs.

Our paper focuses on linear hierarchies as most of the existing schemes have focused on tree and more general hierarchies. The proposed key management schemes are thus sophisticated and use many parameters in order to carry out efficient key management for such hierarchies. However, when applying these sophisticated schemes to the simple case of linear hierarchies, most of the used

parameters (that are explicitly or implicitly related to the nature of the hierarchy) become useless, making an unjustified resource overhead. What we propose in this paper is a scheme that is optimized for linear hierarchies.

3.1. Key Management for Linear Hierarchies

We focus here on the generation and renewal of content encryption keys for linear hierarchies. Every member of SC_i maintains a secret CEK K_c to be used as an encryption/decryption key for their class. The hierarchy (group) is divided into C security classes SC_c, $1 \leq c \leq C$. Every class SC_c contains N_c members. Classes follow a linear hierarchy, i.e $\forall i, j \in \{1, \ldots, C\} : SC_i < SC_j$ or $SC_i > SC_j$. We reassign indexes to SCs in such a way that: $SC_1 \succeq SC_2 \succeq \cdots \succeq SC_{C-1} \succeq SC_C$. Note that SC_1 is the highest class in the hierarchy.

3.1.1. Confidentiality Requirements

In addition to the backward and forward secrecy mentioned above, we have determined two linear hierarchies specific requirements: *Upward Secrecy*, and *Downward Secrecy*. They correspond to promotions and degradations of class members. *Promotion* of a member is passing from a class to a higher one. Inversely, passing from a class to a lower one, is called *degradation*. Thus, the confidentiality requirements are defined as follows:

1. **Upward Secrecy:** The promotion of a member from a class u to a higher class t will give them access to current communications of classes between t and u. However, they should be prevented from accessing old communications of the classes to which they had not access before the promotion(i.e. any class x verifying $SC_u < SC_x \leq SC_t$);
2. **Downward Secrecy:** The degradation of a member from a class t to a lower class u should prevent them from accessing future communications of the classes to which they have no more access after the degradation (i.e. any class x verifying $SC_u < SC_x \leq SC_t$).

3.1.2. Rekeying Triggers

A key renewal can be triggered by a join/leave of a host, or by promotion/degradation of a member.

Join: Backward Secrecy

The arrival of a new member to class SC_t causes a key renewal of all classes SC_u, $SC_u \leq SC_t$. Otherwise, once that the new member has obtained her key set containing $\{K_t, K_{t+1}, ..., K_C\}$, they will be able to decrypt old traffic sent before their arrival. Therefore, the rekeying pattern consists in the renewal of all the keys of class SC_t and all the lower classes.

Leave: Forward Secrecy

Similarly to the previous case, a departure of a member of class t causes a rekeying of all the classes SC_u where $SC_u \leq SC_t$. The rekeying pattern is the same.

Promotion: Upward Secrecy

The promotion of a member of class SC_u to class SC_t, $SC_u < SC_t$, requires a slightly more complicated rekeying pattern. Backward secrecy of classes implies that we should renew keys K_c, such that $SC_u < SC_c \leq SC_t$. Therefrom, the rekeying pattern consists on the renewal of all keys of all the classes SC_c such that $SC_u < SC_c \leq SC_t$.

Degradation: Downward Secrecy

When a member is degraded from class SC_t to class $SC_u < SC_t$, they should no longer be able to decrypt traffic of their old class, as well as traffic of intermediate classes, i.e all classes c such that $SC_u < SC_c \leq SC_t$. This is due to the constraint of future confidentiality of these classes. The rekeying is exactly the same as promotion rekeying.

3.1.3. Rekeying Patterns

We have shown that forward and backward secrecy imply the same rekeying steps. Upward and downward secrecy share the same rekeying steps as well. This suggests to divide rekeying processes into two different generic patterns:

Partial Renewal R_t : consists in renewing the keys of the classes SC_t, SC_{t+1}, $\ldots$, SC_C and updating the system accordingly. A partial renewal R_t is triggered when a member joins/leaves class SC_t.

Bounded Renewal $R_{t,u}$: consists in renewing the keys of the classes SC_t, $SC_{t+1}, \ldots, SC_{u-1}$ and updating the system accordingly. A partial renewal $R_{t,u}$ is triggered when a member is promoted (resp. degraded) from u to t (resp. t to u).

Any key management scheme for linear hierarchy will need to clearly specify how these two patterns are carried out. We specify them for our scheme in 3.3.2

3.1.4. Key Management Schemes for Linear Hierarchies

There is no linear hierarchies-specific key management schemes. All the existing schemes were proposed for tree and DAG hierarchies. However, it is interesting to note that almost all independent and dependent indirect schemes are reduced to only two schemes when applied to linear hierarchies: the independent scheme, and the dependent scheme.

Dependent Scheme

In this scheme, all the keys are related using a one way function f such that the key K_i of class SC_i is: $K_{i+1} = f(K_i)$, where the key of SC_1, K_1, is generated randomly. Any membership or hierarchy change induces a complete renewal. This scheme has the advantage of reducing the storage per user to one key, and thus the number of CEKs sent to each user to renew their set of keys to one CEK.

Independent Scheme

This scheme randomly generates a key for each class. Each member of a

particular class stores the key of their own class as well as the keys of all the lower classes. In the case of a membership or hierarchy change, only concerned CEKs are renewed. This scheme does not require particular computation overhead for CEKs management contrary to the dependent scheme.

We will use these two schemes when comparing our scheme, using simulations, to the existing schemes.

3.2. Our Scheme: KTLH

We propose a key management scheme for group communications within hierarchical environnements: *Key Tables-based key management scheme for Linear Hierarchies* (KTLH). KTLH relates keys in such a way that, knowing their own class key, a member can compute keys of lower classes. Key tables are used in KTLH to maintain this keys relation as described below.

Initialization

In KTLH, keys are initialized as follows. First, KTLH randomly generate a key K_1. Then, KTLH uses a hash function H to compute a chain of keys using the formula: $K_{t+1} = H(K_t)$, where K_t is the key of the t^{th} class. Then each key K_t is sent to its corresponding class t. Thus, only one key per member is sent. Once a member u_t of class SC_t receives their key, they can, if required, compute the key of any class SC_u, $SC_u < SC_t$, by simply applying $(u-t)$ times H to K_t.

Since the keys are renewed several times, we denote by K_t^p the key of SC_t after the $(p-1)^{th}$ key renewal. That is, the initial key of SC_t is denoted SC_t^1. In what follows we specify the rekeying patterns for our scheme.

3.2.1. Partial Rekeying (R_t): Join/Leave

The partial rekeying R_t in KTLH is carried out as follows:

Partial Rekeying Algorithm in KTLH

Step 1: The key generation mechanism (a Group Controller and Key Server, *GCKS*, for instance) randomly generates a new key K_t^{p+1} for class SC_t
Step 2: The *GCKS* computes the new keys K_c^{p+1}, $SC_C \leq SC_c < SC_t$ using H: K_c^{p+1}=$H(K_{c-1}^{p+1})$
Step 3: The *GCKS* sends every key K_c^{p+1}, $SC_C \leq SC_c \leq SC_t$ to its corresponding class and sends an update message to superior classes SC_s ($SC_s > SC_t$) containing the couple (t , K_t^{p+1})
Step 4: Members of each class update their key tables according to the received messages

Update messages are used to maintain the key chain. For instance, when a partial rekey occurs at the t^{th} class, it is clear that, if we do not use the

update messages, members of higher classes will have no idea about K_t. This is because it was not computed based on K_{t-1} using H. To overcome chain discontinuity problem, members should maintain a key table. Each time that a member receives an update message, they make necessary updates to their key table. Later, when they need to compute the key of a lower class u, they look in their table for the biggest class index t such that $t \leq u$.

Example 3.1. *Let's consider a group with five security classes ($C = 5$). Initially, every member needs to know merely their own class key. Assume that a member of SC_3 leaves the group. Keys of SC_3, SC_4 and SC_5 must be renewed. That is, the partial renewal R_3 should be triggered. If we assume that we use a centralized key distributer S, then it should generate a new key K_3^2 (2^{nd} version of K_3), and compute from it, using the one-way function H, K_4^2 and K_5^2. Security classes SC_3, SC_4 and SC_5 will receive K_3^2, K_4^2 and K_5^2 respectively. While classes SC_1 and SC_2 will receive the pair (K_3^2 , 3), to indicate that K_3^2 is the new key of SC_3.*

Once a member of SC_1 for example receives the pair $(K_3^2, 3)$, they update their table which becomes as shown in TAB. *2.*
If a member of SC_1 wants to access an information item within SC_2, then they will compute K_2^2 by applying H one time on K_1^2. If they want to access SC_5's information items, then they will use the second entry of her key table: K_3^2. K_5^2 is computed by applying H two times consecutively on K_3^2.
Members of SC_4 will have TAB. *3. It contains their class's key, and they can calculate K_5^2 when needed by applying $K_5^2 = H(K_4^2)$*

3.2.2. Bounded Rekeying ($R_{t,u}$): Promotion/Degradation

As we have seen in section 3.1, a bounded rekeying is required each time a member moves up or down in the hierarchy. The bounded rekeying $R_{t,u}$ is implemented in KTLH as follows:

Bounded Rekeying Algorithm in KTLH

Step 1: The *GCKS* randomly generates a new key K_t^{p+1} for class t
Step 2: The *GCKS* computes the new keys K_c^{p+1}, $SC_u \leq SC_c < SC_t$ using H
Step 3: The *GCKS* sends every key K_c^{p+1}, $SC_u \leq SC_c \leq SC_t$ to its corresponding class and sends two update messages.

- The first to superior classes s ($SC_s > SC_t$) containing the couple $(t,\ K_t^{p+1}$
- The second message contains $(u+1,\ k_{u+1}^{p+1})$ to all classes s such that $SC_s \geq SC_u$

Step 4: Members of each class update their key tables according to the received messages

Example 3.2. *Let's reconsider the previous example in its initial status:* $C = 5$ *and each member initially knows only the key of their own class. Assume that a member of* SC_4 *has just been promoted to* SC_2*. Upward secrecy requires changing keys of* SC_2 *and* SC_3*. The system generates a new key* K_2^2 *for* SC_2*. Then S computes* K_3^2 *by:* $K_3^2 = H(K_2^2)$*. That is, the new content encryption keys are* $K_1^2 = K_1^1$ *(has not changed),* K_2^2*,* K_3^2*,* $K_4^2 = K_4^1$ *(has not changed) and* $K_5^2 = K_5^1$ *(has not changed). The system sends* K_2^2 *to members in* SC_2 *and* K_3^2 *to those in* SC_3*. It also sends key tables update messages to* SC_1*,* SC_2 *and* SC_3*; members of* SC_2 *and* SC_3 *will receive only one message containing the pair* $(K_4^2, 4)$*. Whereas members of* SC_1 *will receive two messages: the first containing* $(K_2^2, 2)$*; and the second containing* $(K_4^2, 4)$*.*
Upon receiving these messages, members of SC_1 *will build key tables of the form* TAB. *4. While members of* SC_3 *will build* TAB. *5.*

3.2.3. Hierarchy Change Management

Possible hierarchy change events for linear hierarchies are class addition and class deletion. We show here how KTLH achieves them

1- Class Addition: Let SC_t be the class after which a new class SC'_t is inserted. KTLH carries out the class addition as follows:

Class Addition Algorithm in KTLH

Step 1: The number of classes C is incremented
Step 2: indexes are reassigned such that SC_{t+1} becomes SC_{t+2}, SC'_t becomes SC_{t+1} and so on
Step 3: make the partial rekeying R_{t+1}

2- Class Deletion: Suppose that SC_{t+1} will be deleted. The class deletion is made as follows:

Class Deletion Algorithm in KTLH

Step 1: The number of classes C is decremented
Step 2: indexes are reassigned such that SC_{t+2} becomes SC_{t+1} and so on
Step 3: make the partial rekeying R_{t+1}

Hierarchy change management is carried out by partial rekeying. The cost of a hierarch change is exactly the same as the cost of a join/leave event which is the cost of a partial rekeying. This is a great advantage for our scheme compared to several existing schemes where resource consuming operations should be executed for each hierarchy change.

4. Simulation

We conducted intensive simulations to compare our scheme with the independent and dependent schemes. We make the comparison according to the three main system resources: the storage needed on user side, the bandwidth required to update the system after a renewal , and the computation overhead on the key generator's (a $GCKS$ for example) side:

1. Storage: number of content encryption keys stored on each member, their own class key included;
2. Bandwidth: number of messages sent per member in order to make rekeying when a membership changement arises.
3. Computation: number of the applications of the one-way function by the the key generator per membership or hierarchy change.

4.1. A First Scenario

We will consider a group containing 5 classes which is a real example that we find in several cases (military communications, multi-layered video streaming). Members arrival follows a Poisson law and their membership duration follows an exponential distribution [42].

A typical member session starts by a *join* event, which can be followed by one or more *rise* and/or *degrade* events before a *leave* occurs. At the end of their membership in some class, a member leaves the group with probability $prob\_leave$, or changes the class with probability $prob\_change(= 1 - prob\_leave)$. A member who changes their class, is degraded according to probability $prob\_degrade$, and is promoted with probability $prob\_rise = (1 - prob\_degrade)$.

We will consider a session of 3 hours. Inter-arrival average λ is of 20 seconds, and average membership duration μ is 30 minutes. Table 6 summarizes simulation parameters.

Results

Figure 6 compares key storage per class between KTLH and the dependent and independent schemes. In the independent solution, storage is constant per class, because every member stores, in addition to their class key, keys of all lower classes. In KTLH, every member permanently stores initially only their class's key, and according to group membership evolution, number of stored keys can increase or decrease. KTLH decreases the required key storage by 36,76%. On the other hand, since the dependent scheme allows each member to compute keys of lower classes, the key storage per class is equal to 1.

Figure 7 compares bandwidth overhead per class between the three schemes. It gives, for each class, the number of messages required to update its key table. The independent scheme sends a number of keys equal to number of key stored on each member. KTLH reduces number of sent messages to 0, 1 or 2 according to rekeying type and class position. Thus, KTLH allows to save 45,19% of the bandwidth overhead. Note that, for KTLH, the average bandwidth for lower classes is inferior to 1. This is logical because many rise/degrade events do not require to update their keys. According to our simulations, the average bandwidth required by KTLH to update keys of a given class is 1.00. The dependent scheme sends one key to each class for any membership change; hence KTLH has the same average bandwidth as the dependent schemes.

Figure 8 compares computation overhead per membership change of the three schemes through successive simulation runs. Since it uses no one-way function, the computation overhead of the independent scheme is 0. The computation overhead of the dependent scheme is constant as well and is equal to 4 one-way function computations per membership change. This is due to the fact that all the keys need to be changed after any membership change because they are related‘. Whereas the average computation overhead per membership change for KTLH is 1.28.

Table 7 summarizes the simulation results for a hierarchy of five classes. It shows that KTLH trades very well the three criteria and gives best overall results. Comparing KTLH to each of the schemes shows that it tunes very well the system resources.

4.2. Scalability

In this subsection, instead of focusing on the average cost, we will consider how the total cost of a communication session for each approach varies as a function of each parameter. This allows us to study and emphasize the scalability of each solution according to each parameter. In what follows, we study how KTLH scales according to three parameters:

1. Number of classes within the group;
2. Group size: the number of members within the group;
3. Group dynamicity: the number of rekeying triggers per time unit. It is interesting to note that increasing the group dynamicity increases the arrival rate and that group dynamicity is inversely proportional to the membership duration within each group class.

4.2.1. Number of Classes

Figure 9(a) shows how the storage overhead varies as a function of the number of classes. We note that the storage overhead is constant for 30 classes and above and is equal to about 2.7 keys. It is very interesting to note that even for 100 classes, the average storage overhead per class is less than 3 keys. We observe the same thing for the bandwidth overhead in figure 9(b). The average bandwidth is constant and is equal to 1.007, that is why the dependent approach curve (which is constant and equal to 1) overlaps with the KTLH curve. Figure

9(c) shows that the computation overhead of KTLH is about one third of the overhead of the independent scheme. Of course, the computation overhead of the independent approach is equal to zero.

4.2.2. Group Size

Figure 10(a) shows that the total storage of KTLH is linear, and that its overhead is about the average of the two other schemes. As we can see in figure 10(b), the bandwidth overhead of KTLH is the same as the dependent scheme. The computation overhead is shown in figure 10(c), the computation overhead of the independent scheme is obviously equal to zero. All KTLH overheads are linear and thus scale very well when the group size increases.

4.2.3. Group Dynamicity

We vary the group dynamicity by varying the arrival rate λ and the membership duration μ. Figure 11(a) shows that the total storage of KTLH is about the average of the two other schemes. As for the group size, figure 11(b) shows that the the bandwidth overhead of KTLH is the same as the dependent scheme. The computation overhead is shown in figure 11(c), the computation overhead of the independent scheme is nil. As for the group size, KTLH scales very well for highly dynamic groups.

4.3. Discussion

Simulations have shown that KTLH reduces members storage by more than 35%, and bandwidth by more than 45% relatively to the independent solution. It has the same bandwidth overhead as the dependent scheme, and an average storage overhead less than three keys per class. We showed also that KTLH requires a small amount of computation compared to the dependent solution. KTLH scales very well, particularly when the number of classes changes, which is not the case for many of the existing schemes as we will see in the next section.

5. Complexity Comparison

In this section, we study the complexity of a key renewal in KTLH. We study the computation, bandwidth and storage complexity in terms of the number of classes C. We then compare KTLH to the schemes we presented in subsection 2.2.4.

As shown in figure 9(b), the required bandwidth is constant. The bandwidth complexity is $O(1)$. Whereas figure 9(c) shows that the computation overhead goes linearly with the number of classes. The computation overhead is $O(C)$. Figure 9(a) shows that the average number of keys stored per class is constant for 30 classes and over. That is, the storage overhead is inferior to a constant α ($\alpha < 3$). Therefore the storage complexity is $O(1)$.

We have studied the complexity of all the schemes presented in 2.2.4 when applied to a linear hierarchy. Table 8 summarizes the results of our study.

This table clearly shows that KTLH tunes very well the three criteria and gives best combined performances. Table 9 makes an "abstract" comparison between our scheme and the three approaches: independent, direct dependent and indirect dependent key management schemes. The only approach that gives better performance on one of the three criteria is the independent approach. But KTLH is better than this approach for the two other criteria.

6. Conclusion

In this paper, we introduced the linear hierarchical group model which, as far we can say, is a very promising communication model. We identified two new security requirements: *upward secrecy* and *downward secrecy*.

Furthermore, we classified the existing key management schemes into two categories and showed that most of this schemes are reduced to one of two schemes in the case of linear hierarchies: the independent and the dependent schemes, then we described each of them. Thereafter, we presented our solution KTLH which is a key tables and hash functions-based solution. We conducted intensive simulations to compare KTLH to the two key management approaches and showed that KTLH has good performances. We also computed the key renewal complexity for several key management schemes and compared them to KTLH. This comparison had also shown that KTLH tunes very well storage, bandwidth and computation overheads.

[1] H. R. Hassen, A. Bouabdallah, H. Bettahar, Y. Challal, Key management for content access control in a hierarchy, Comput. Netw. 51 (11) (2007) 3197–3219. doi:http://dx.doi.org/10.1016/j.comnet.2006.12.011.

[2] C. K. Wong, M. Gouda, S. S. Lam, Secure group communications using key graphs, IEEE/ACM Transactions on Networking 8(1) (2000) 16–30.

[3] H. Harney, C. Muckenhirn, RFC 2093 - Group Key Management Protocol (GKMP) Architecture (July 1997).

[4] D. Balenson, D. McGrew, A. Sherman, Key Management for Large Dynamic Groups: One-Way Function Trees and Amortized Initialization, draft-balenson-groupkeymgmt-oft-00.txt, internet-Draft (February 1999).

[5] A. Penrig, D. Song, D. Tygar, Elk, a new protocol for efficient large-group key distribution, in: Security and Privacy, 2001. S P 2001. Proceedings. 2001 IEEE Symposium on, 2001, pp. 247 –262. doi:10.1109/SECPRI.2001.924302.

[6] C. K. Wong, S. S. Lam, Keystone: A group key management service, in: in: International Conference on Telecommunications, ICT, 2000.

[7] T. Hardjono, B. Weis, RFC 3740 - The Multicast Group Security Architecture (2004).

[8] A. Ballardie, RFC 1949 - Scalable Multicast Key Distribution (1996).

[9] T. Hardjono, B. Cain, I. Monga, Intra-domain Group Key Management for Multicast Security (September 2000).

[10] S. Mittra, Iolus: A framework for scalable secure multicasting, 1997, pp. 277–288.

[11] L. R. Dondeti, S. Mukherjee, A. Samal, Scalable secure one-to-many group communication using dual encryption, Computer Communications 23(17) (2000) 1681–1701.

[12] S. Rafaeli, D. Hutchison, Hydra: a decentralised group key management, in: Enabling Technologies: Infrastructure for Collaborative Enterprises, 2002. WET ICE 2002. Proceedings. Eleventh IEEE International Workshops on, 2002, pp. 62 – 67. doi:10.1109/ENABL.2002.1029990.

[13] S. Setia, S. Koussih, S. Jajodia, E. Harder, Kronos: a scalable group re-keying approach for secure multicast, in: Security and Privacy, 2000. S P 2000. Proceedings. 2000 IEEE Symposium on, 2000, pp. 215 –228. doi:10.1109/SECPRI.2000.848459.

[14] B. Briscoe, Marks: Zero side effect multicast key management using arbitrarily revealed key sequences, in: PROC. FIRST INTERNATIONAL WORKSHOP ON NETWORKED GROUP COMMUNICATION (NGC'99), 1999.

[15] M. Steiner, G. Tsudik, M. Waidner, Diffie-Hellman key distribution extended to group communication, 3rd ACM Conference on Computer and Communications Security (1996) 31–37.

[16] Y. Kim, A. Perrig, G. Tsudik, Simple and fault-tolerant Key Agreement for Dynamic Collaborative groups, 7th ACM Conference on Computer and Communications Security (2000) 235–244.

[17] C. Becker, U. Wille, Communication complexity of group key distribution, ACM Press, 1998.

[18] O. Rodeh, K. P. Birman, D. Dolev, Optimized group rekey for group communication systems, in: In Proceedings of ISOC Network and Distributed Systems Security Symposium, 2000.

[19] C. Boyd, On key agreement and conference key agreement, Information Security and Privacy: Australasian Conference LNCS(1270) (1997) 294–302.

[20] C. K. Wong, M. Gouda, S. Lam, Secure group communications using key graphs, Networking, IEEE/ACM Transactions on 8 (1) (2000) 16 –30. doi:10.1109/90.836475.

[21] E. Gudes, The design of a cryptography based secure file system, IEEE Transactions on Software Engineering. SE-6, 5 (1980) 411–420.

[22] S. Akl, P. Taylor, Cryptographic solution to a problem of access control in a hierarchy, ACM Transactions on Computer Systems (1983) 239–248.

[23] R.Sandhu, Cryptographic implementation of a tree hierarchy for access control, Information Processing Letters 27, no. 2 (1988) 95–98.

[24] R. Sandhu, On some cryptographic solutions for access control in a tree hierarchy, IEEE (1987) 405–410.

[25] C. Yang, C. Li, Access control in a hierarchy using one-way functions, Elseveir Computers & Security 23 (2004) 659–664.

[26] Z. hua He, Y.-S. Li, Dynamic key management in a user hierarchy, in: Anti-counterfeiting, Security and Identification, 2008. ASID 2008. 2nd International Conference on, 2008, pp. 298 –300. doi:10.1109/IWASID.2008.4688404.

[27] Z. Wang, X. Du, Y. Sun, G. Xu, Group key management based on information flow policy for multi-privileged groups, in: Information Theory and Information Security (ICITIS), 2010 IEEE International Conference on, 2010, pp. 485 –489. doi:10.1109/ICITIS.2010.5689555.

[28] I. Gawdan, C.-O. Chow, T. Zia, Q. Sarhan, A novel secure key management module for hierarchical clustering wireless sensor networks, in: Computational Intelligence, Modelling and Simulation (CIMSiM), 2011 Third International Conference on, 2011, pp. 312 –316. doi:10.1109/CIMSim.2011.63.

[29] I. Ray, I. Ray, N. Narasimhamurthi, A cryptographic solution to implement access control in a hierarchy and more, SACMAT'02 (2002) 65–73.

[30] X. Zou, B. Ramamurthy, S. S. Magliveras, Chinese remainder theorem based hierarchical access control for secure group communication, ICICS '01: Proceedings of the Third International Conference on Information and Communications Security (2001) 381–385.

[31] Y. Zheng, T. Hardjono, J. Seberry, New solutions to the problem of access control in a hierarchy, Tech. rep., Department of Computer Science, University of Wollongong, Australia (1993).

[32] Zheng, Hardjono, Pieprzyk, Sibling intractable function families and their applications, in: ASIACRYPT: Advances in Cryptology – ASIACRYPT: International Conference on the Theory and Application of Cryptology, LNCS, Springer-Verlag, 1991.

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

[34] J. Zhang, J. Li, X. Liu, An efficient key management scheme for access control in a user hierarchy, Information Technology and Computer Science, International Conference on 1 (2009) 550–553. doi:http://doi.ieeecomputersociety.org/10.1109/ITCS.2009.119.

[35] S.-F. Tzeng, C.-C. Lee, T.-C. Lin, A novel key management scheme for dynamic access control in a hierarchy, International Journal of Network Security 1 (2011) 68–70.

[36] M. L. Das, A. Saxena, V. P. Gulati, D. B. Phatak, Hierarchical key management scheme using polynomial interpolation, SIGOPS Oper. Syst. Rev. 39 (1) (2005) 40–47. doi:http://doi.acm.org/10.1145/1044552.1044556.

[37] R. Aparna, B. Amberker, Key management scheme for multi-layer secure group communication, in: Communication Systems and Networks and Workshops, 2009. COMSNETS 2009. First International, 2009, pp. 1 –6. doi:10.1109/COMSNETS.2009.4808860.

[38] G. Liu, L. Li, J. Zheng, Z. Li, A hierarchical key management scheme in role-based access control, in: Computer Design and Applications (ICCDA), 2010 International Conference on, Vol. 5, 2010, pp. V5–581 –V5–584. doi:10.1109/ICCDA.2010.5541164.

[39] D. Ma, R. H. Deng, Y. Wu, T. Li, Dynamic access control for multiprivileged group communications, in: International Conference on Information and Communication Systems, 2004.

[40] Y. Karandikar, X. Zou, Y. Dai, Secure group communication based scheme for differential access control in dynamic environments, in: Parallel and Distributed Systems, 2005. Proceedings. 11th International Conference on, Vol. 2, 2005, pp. 448 –452. doi:10.1109/ICPADS.2005.260.

[41] X. Zou, S. Magliveras, B. Ramamurthy., Key tree based scalable secure dynamic conferencing schemes., proceedings of International Conference on Parallel and Distributed Computing and Systems (PDCS 2004),.

[42] K. Almeroth, M. Ammar, Multicast group behavior in the internet's multicast backbone (mbone), Communications Magazine, IEEE 35 (6) (1997) 124 –129. doi:10.1109/35.587716.

Notation	**Meaning**
$GCKS$	The Group Controller and Key Server
CEK	Content Encryption Key
E	The set of all entities within the system
S	The set of all subjects
O	The set of all objects
SC	The set of all security classes
$s \succeq o$	Subject s has access to object o
$SC_i \succeq SC_j$	Security class SC_i dominates SC_j

Table 1: Used notations.

Security Class	Key
1	$K_1^2 (= K_1^1)$
3	K_3^2

Table 2: Key table maintained by SC_1 members.

Security Class	Key
4	K_4^2

Table 3: Key table maintained by SC_4 members.

Security Class	Key
1	$K_1^2 (= K_1^1)$
2	K_2^2
4	$K_4^2 (= K_4^1)$

Table 4: Key table maintained by SC_1 members.

Security Class	Key
3	K_3^2
4	$K_4^2 (= K_4^1)$

Table 5: Key table maintained by SC_3 members.

Parameter	Role	Value
prob_leave	probability to leave the group	0.95
prob_change	probability to change class	0.05
prob_rise	probability to have a promotion	0.5
prob_degrade	probability to have a degradation	0.5
λ	Inter-arrival average	20 sec
μ	Average membership duration	30 min
T	Session duration	3 h

Table 6: List of simulation parameters.

Scheme	**Storage**	**Bandwidth**	**Computation**
Independent	3	1.81	0
KTLH	1.90	1	1.28
Dependent	1	1	4

Table 7: Schemes Comparison in the case of a hierarchy of 5 classes.

Scheme	**Computation**	**Bandwidth**	**Storage**
KTLH	***O(C)***	***O(1)***	***O(1)***
Sandhu[23]	$O(C)$	$O(C)$	$O(1)$
Gudes[21]	$O(log(C))$	$O(log(C))$	$O(log(C))$
Yang & Li[25]	$O(C)$	$O(C)$	$O(1)$
Akl & Taylor[22]	$O(C^2)$	$O(C)$	$O(1)$
Ray et al.[29]	$O(C^2)$	$O(C)$	$O(C)$
Shen& Chen[33]	$O(C^2 \times log(C))$	$O(C)$	$O(log(C))$
Das et al.[36]	$O(C^2 \times log(C))$	$O(N_{sc})$	$O(log(C))$
*Sun & Liu. [**?**]*	$O(log(C))$	$O(log(C))$	$O(log(C))$
Ma et al. [39]	$O(C)$	$O(C)$	$O(C)$
Karandikar et al. [40]	$O(log(C))$	$O(log(C))$	$O(1)$

Table 8: A comparison of key renewal cost of CACH key management schemes.

Approach	**Computation**	**Bandwidth**	**Storage**
Indirect	$O(C)$	$O(C)$	$\geq O(log(C))$
Direct	$\simeq O((C^2))$	$O(C)$	**O(1)**
Indep.	**O(log(C))**	$O(log(C))$	$O(log(C))$
KTLH	$O(C)$	**O(1)**	**O(1)**

Table 9: A comparison of overheads of key management approaches.

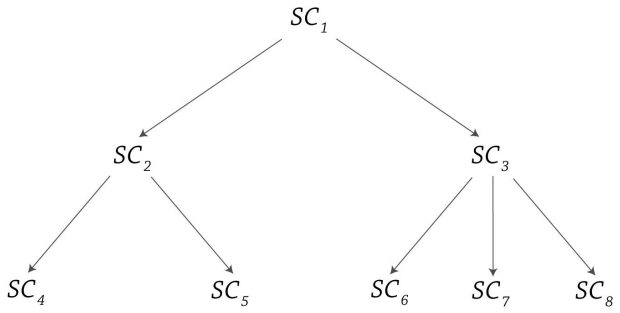


Figure 1: An example of a Hierarchy

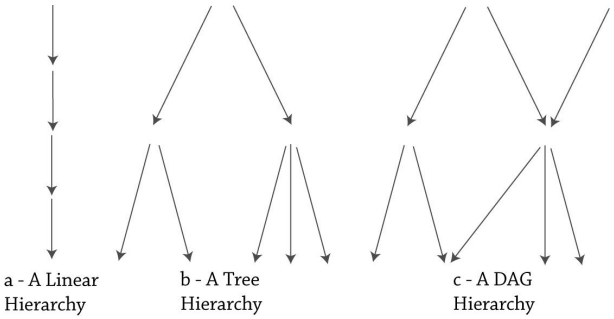


Figure 2: Hierarchy Shapes

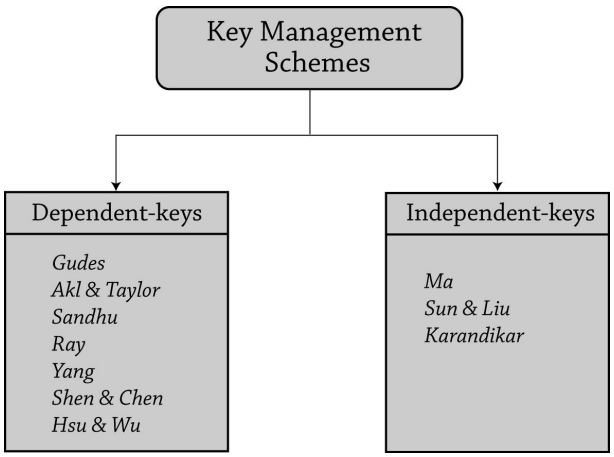


Figure 3: A classification of CACH key management schemes

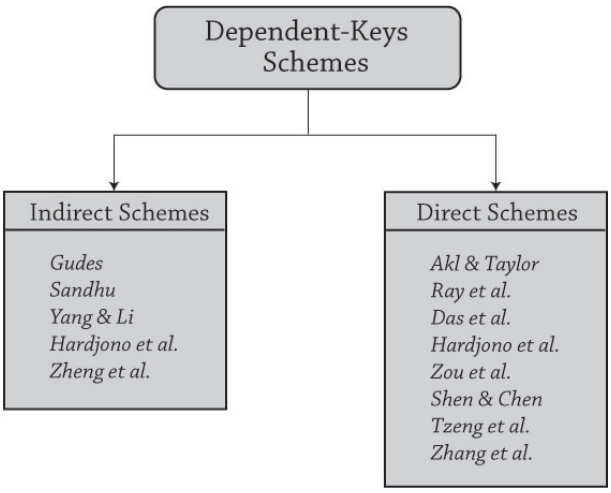


Figure 4: Indirect and direct key management schemes

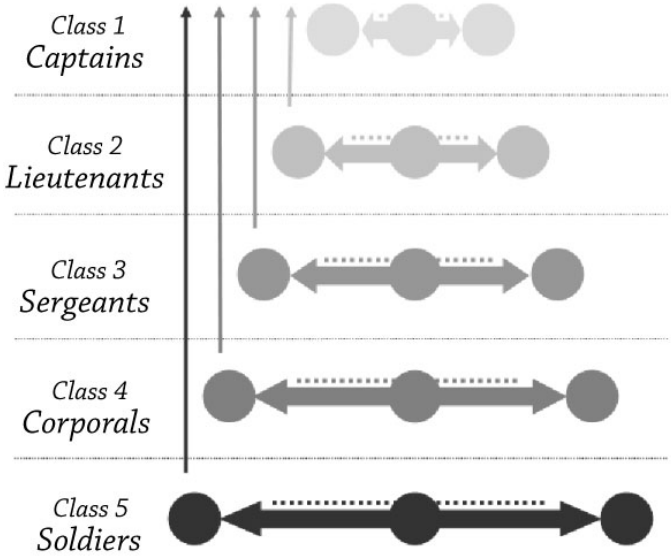


Figure 5: A linear hierarchy within a military group

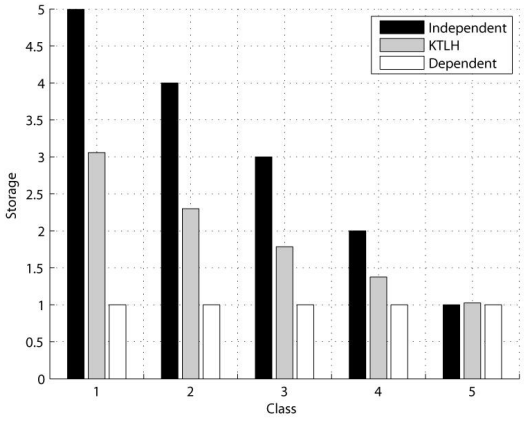


Figure 6: Per-class storage for each approach

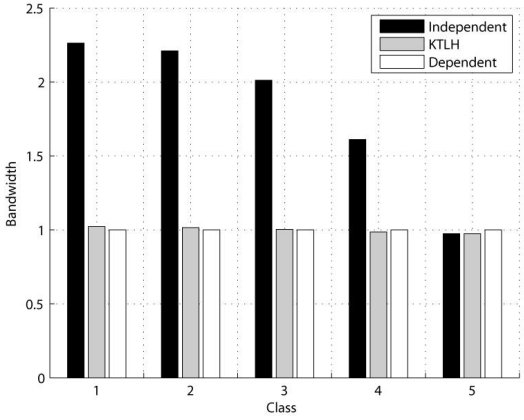


Figure 7: Per-class bandwidth for each approach

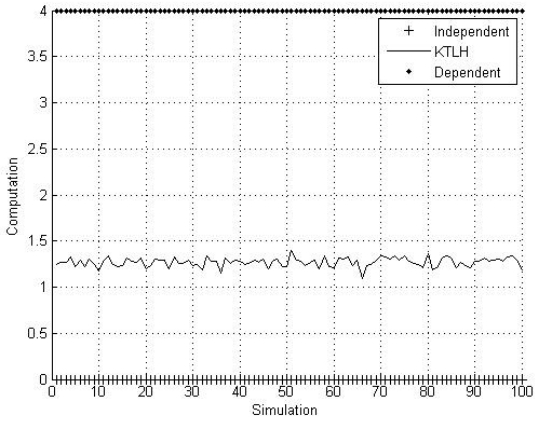
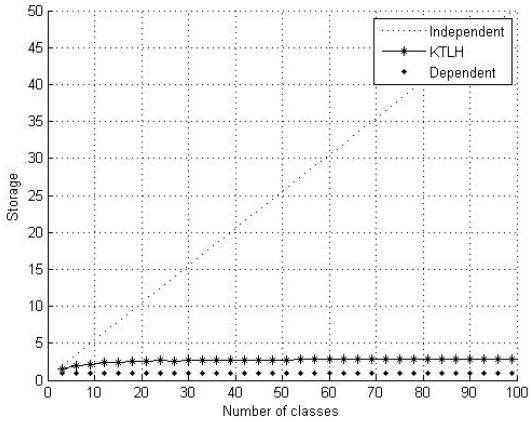
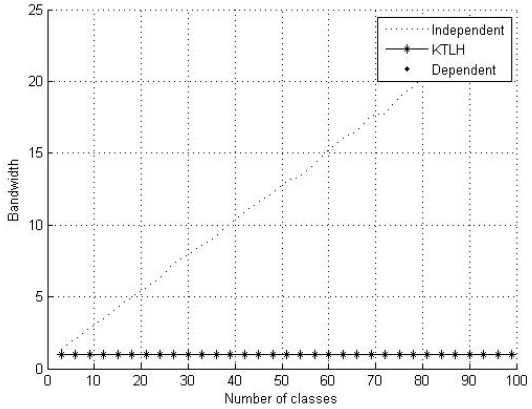


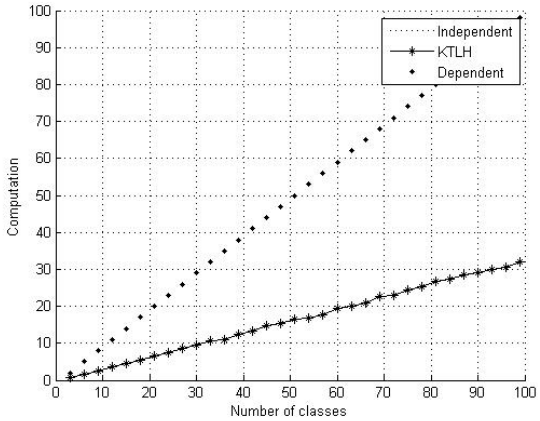
Figure 8: Computation comparison



(a) Storage function of classes number

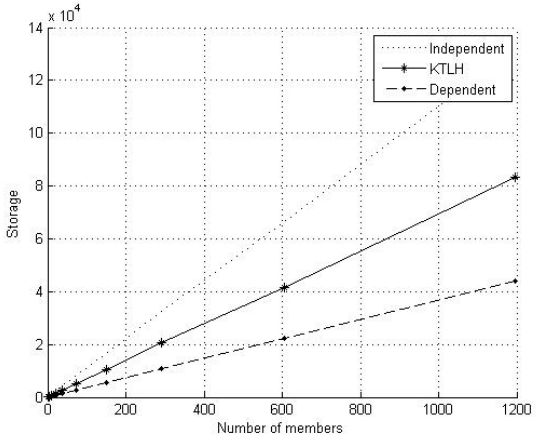


(b) Bandwidth function of classes number

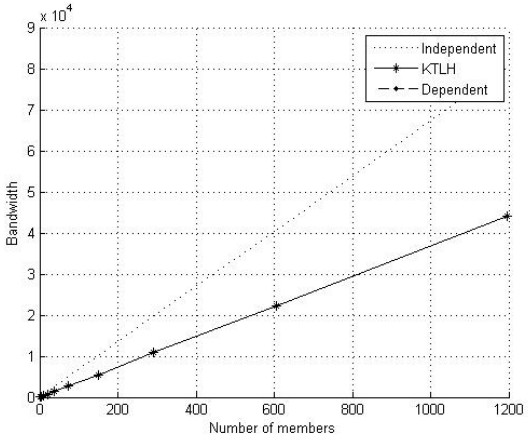


(c) Computation function of classes number

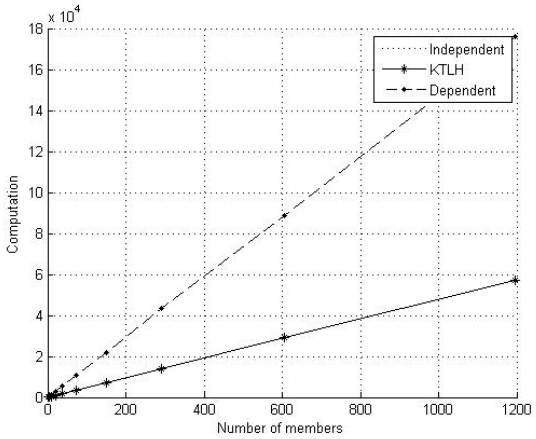
Figure 9: Performance criteria function of classes number



(a) Storage function of group size

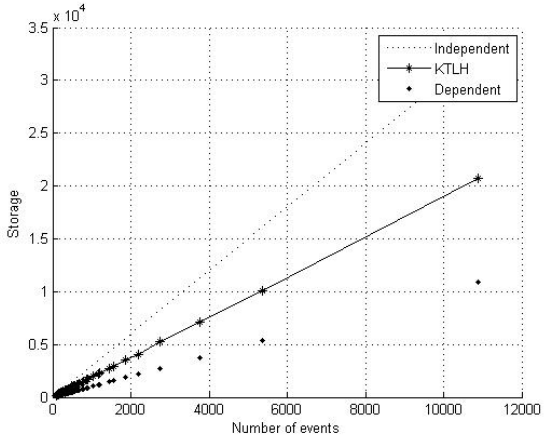


(b) Bandwidth function of group size

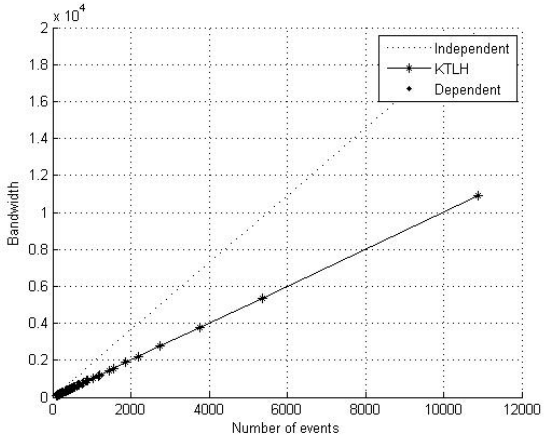


(c) Computation function of group size

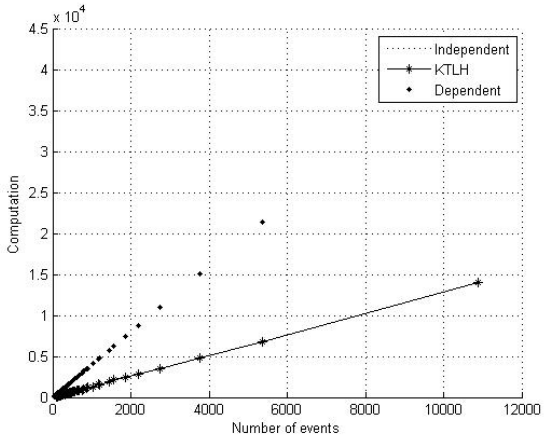
Figure 10: Performance criteria function of group size



(a) Storage function of group dynamicity



(b) Bandwidth function of group dynamicity



(c) Computation function of group dynamicity

Figure 11: Performance criteria function of group dynamicity