Multi-session Key Management Scheme for Multimedia Group Communications

JOSE ROBERTO PEREZ CRUZ [1], *SAUL EDUARDO POMARES HERNANDEZ* [1], *GUSTAVO RODRIGUEZ GOMEZ* [1], *KHALIL DRIRA* [2,3] *and MICHEL DIAZ* [2,3]
[1]*Department of Computer Science,*
National Institute of Astrophysics, Optics and Electronics (INAOE),
Luis Enrique Erro No. 1, 72840, Tonantzintla, Puebla, Mexico
[2] *CNRS, LAAS, 7 avenue du colonel Roche, F-31077 Toulouse, France*
[3]*Universite de Toulouse, UPS, INSA, INP, ISAE, LAAS, F-31077 Toulouse, France*
{*jrpc, spomares, grodrig*}*@inaoep.mx*

Abstract

The Internet 2 deployment introduces new capabilities, such as multi-party collaboration, high-scale multimedia assembly and multicast communication. For this reason, the research concerning security is facing new challenges. One such challenge is to create secure multi-session frameworks to ensure the confidentiality of exchanged information. In a multi-session environment, several users are joined at two or more work sessions simultaneously. The confidentiality in these environments can be achieved using cryptographic methods. Unfortunately, the key management, necessary for such environments, creates two main problems: a high complexity in key distribution and a high storage cost. In this paper, we propose an efficient multi-session key management mechanism for dynamic multimedia group communication. Our solution proposes a functional architecture that exploits the overlapping of the user sessions to reduce the redundancy in key distribution. The proposed key management makes use of two key generation strategies: a key derivation technique to reduce the rekey overhead and a pseudorandom number generator that allows the users to generate an independent key per cipher packet.

Keywords: Key management, multi-session, multimedia, group communication, one key per packet

1 Introduction

The Internet 2 enables new capabilities, such as multi-party collaboration, high-scale multimedia assembly and multicast communication. The aim of these new capabilities is to develop communication platforms where there may be a high-scale associativity of users communicating by using multimedia data (audio, video, text, still images, etc.). To achieve this, the communication platforms contemplate the support of heterogeneous data management (transmission of discrete and continuous data) which must satisfy certain properties in order to not degrade the quality of service [1]. Specifically, for continuous data transmission, the communication should support the delay, the loss and the transposition of packets.

For these reasons, the new communication platforms are facing new challenges concerning security research. One such challenge is to create secure environments where several applications and several users maintaining work sessions in two or more applications can exist simultaneously. A description of a multi-session environment is given by the following example.

Suppose there are several users on the Internet using some or all of the following three applications: an interactive meeting through session 1, an iTV show through session 2, and a multimedia forum through session 3. Some users coincide in one, two or three of these applications, but there are many other users that do not coincide at all (see Figure 1).

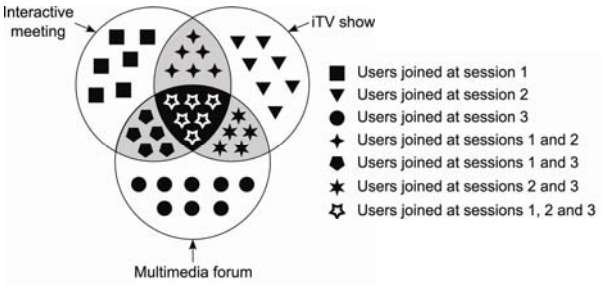

Figure 1: Multi-session scenario with three different applications

As shown in Figure 1, users in gray areas coincide in two applications respectively, while users in the black area coincide in all three applications. Users that coincide in some applications are said to have an *overlapping* in their sessions. With this scenario, users involved in sessions 1 and 3 may exchange information with users associated with these sessions, but users associated only with session 3 should not have access to the information exchanged in session 1 and vice versa. In other words, the exchanged information should be confidential, which means that the information can be accessed only by entities or groups of authorized entities [2, 3, 4].

A practical way to assure the confidentiality is with the use of cryptographic methods along with a selective key distribution technique [5]. Unfortunately, the key management in multi-session environments presents two main problems: a high complexity in key distribution and a high storage cost. These problems arise because users have to store independent keys for each joined session, and the number of keys required for rekeying depends on the number of users in each session.

In addition to the problems associated with key management in multi-session environments, special requirements must be met to preserve the quality of service for continuous data transmission. Since communication channels are not necessarily reliable nor ordered, there is no guarantee that all information is received correctly [6]. Therefore, systems should support the delay, loss and transposition of packets. For these requirements, it is desirable to use independent keys per transmitted packet, so that the lost, delayed or transposed packet has no adverse effects on the quality of service. Furthermore, if there are as many keys as transmitted packets, the level of confidentiality would be even greater.

Currently, some solutions exist that are designed for environments where there are several users with different access privileges or which are associated with different work sessions [6, 7, 10, 11]. Unfortunately, such solutions are not designed to support dynamic formation and decomposition of groups, since they rely on complex architectures to organize users and keys.

In this paper, we propose an efficient multi-session key management mechanism for dynamic multimedia group communication, which is characterized by the use of an independent key per cipher packet. Our mechanism uses a functional architecture that exploits the overlapping present in the user's sessions, creating groups composed of users with the same memberships to reduce the key distribution redundancy. Furthermore, the proposed architecture is designed to support the dynamic formation and decomposition of groups.

Along with the architecture, our mechanism uses two key generation strategies: a derivation technique and a pseudorandom number generator. Through the derivation technique, each formed group is organized into an independent hierarchy to manage the auxiliary keys used in the rekeying, while it allows the members of each group to derive the auxiliary keys by themselves, without the Key Distribution Center (KDC) having to generate, encrypt and distribute all the keys. With the pseudorandom number generator, our mechanism allows the users to generate independent keys for each transmitted packet. By the way in which independent keys are generated, in our solution, users can encrypt and decrypt different streams in an n to m communication, unlike many current solutions where users can only decrypt a specific stream in a 1 to n communication,.

The rest of the paper is organized as follows: in Section 2 we present the related work; in Section 3, we show a detailed description of our mechanism; in Section 4, we give the performance analysis of our solution and a comparison with the related work; and finally, in Section 5, we present the conclusions.

2 Related work

Our solution can be considered as a multi-group key management scheme since users are organized into multiple work groups. For this reason, in this section we present only related work oriented to the multi-group key management.

Multi-group key management has been little explored; however, some important solutions have emerged, which are mainly focused on multi-privileged group communication environments where users have different access privileges.

The main solutions concerning key management for multi-privileged groups are based on the *integrated key graph* (IKG) defined in the Centralized Multi-Group Key Management Scheme (CMGKMS) [6, 7]. With the IKG, users with the same access privileges form a *Service Group* (SG), represented by a key tree. At that point, all the formed trees are connected by their roots and by other auxiliary keys to form a hierarchy of SGs (see Figure 2(b)). Thus, users may increase or decrese their privileges switching through the SGs.

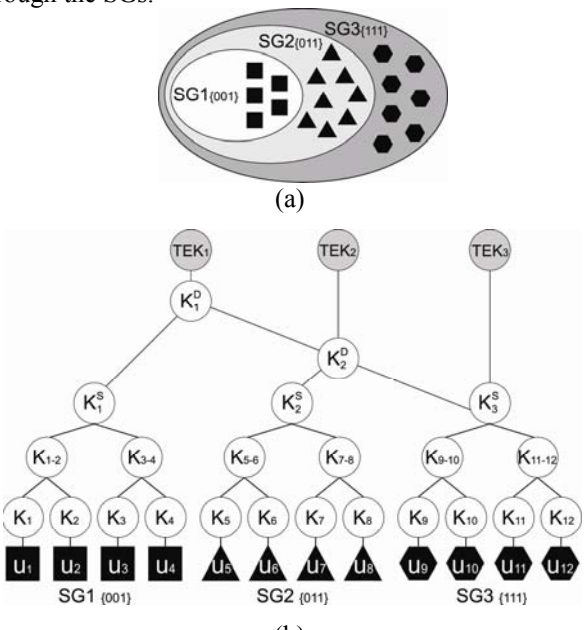

Figure 2: (a)Service Groups (b)Integrated key graph

For example, suppose that there is a video stream with three layers: low, medium and high quality. As shown in Figure 2(a), the users in the white area only have access to the lowest quality, while users in the darkest area have access to all the layers. These users are organized through the hierarchy of Figure 2(b) by three different SGs. Thus,

users in SG3 can access all the Transfer Encryption Keys (TEKs), which are used to encrypt and decrypt the different layers of the video stream, while users in SG1 can only access the first TEK, used to encrypt and decrypt the lowest layer.

The CMGKMS uses the IKG along with a Flat Table rekeying strategy, defined in the CFKM scheme [9], in which users can compute the new keys using one-way functions. Thus, the KDC only has to transmit some IDs and keys in order of the logarithm of the number of users in the affected SG and the height of the SG hierarchy.

The Multi-Group Key Management with Secret Sharing Scheme (MGKMSS) [10] is another solution based on IKG that uses a threshold cryptography to reduce the rekeying overhead, so that instead of keys, the KDC only has to send *secrets* that are smaller than a key.

Another solution that uses the IKG is the ID-based Hierarchical Key Graph Scheme (IDHKGS) [11], which uses a derivation technique based on the key indentification to allow the users to compute the keys by themselves, making the KDC only have to multicast some IDs and the keys that users cannot compute.

Although the IKG uses the overlapping of the user's privileges to reduce the redundancy in key distribution, its construction is too complex. Furthermore, access privileges can only decrease or increase in one way, making this scheme inefficient to manage highly dynamic groups.

The Dynamic Access Control for Multi-privileged Group Communications Scheme (DACMGS) [12] defines a Key Management Graph (KMG) formed by different key trees, which allows the dynamic formation of SGs and decreases the complexity related with the IKG construction. In the KMG, each SG forms an independent key tree, which has an additional vertex above the root to store the TEKs related with the SG. Therefore, if a new SG is formed, the KDC just builds a new tree without affecting the rest of the graph. Furthermore, in a rekey operation, the KDC only has to update the compromised TEKs and the compromised KEKs in the affected tree, allowing a reduction in rekey overhead. However, DACMGS is designed for users who act only as receiver entities, decrypting a specific stream in a 1 to n communication that is not suitable for environments where users exchange multimedia information among them.

3 Proposed solution

We propose a mechanism named Multimedia Multi-session Key Management Scheme (MM-MSKMS). The proposed scheme uses a key forest structure, similar to the Key Management Graph defined in the Dynamic Access Control for Multi-privileged Groups Scheme (DACMGS) [12]. MM-MSKMS differs from the DACMGS in three main aspects: the key forest is combined with a key derivation technique to reduce the rekeying overhead; the users are enabled to transmit multimedia information generating independent keys for each packet, by using a pseudorandom number generator; and finally, the users can exchange streams among them in an n to m communication. The fact that MM-MSKMS uses one key per packet allows the system to support the delay, the loss and the transposition of packets, since each packet is completely independent of the others, avoiding the need for the decryption to be done in a specific order.

3.1 Notation

The notation used through the paper is shown in Table 1:

Table 1: Notation used in this paper

n	number of users in the system
s	number of sessions in the system
OG_t	group of users with overlapped sessions
t	identifier of an OG
n_t	number of users in an OG
$m(s)$	maximum number of OGs
m_0	number of OGs in the system
i, j	indices of KEKs
$K_{i,j}^t$	KEK of tree t
Ω_t	set of Session Keys (SKs) related with an OG
SK_h	SK related with one session $1 \leq h \leq s$
d	degree of a KEK tree
b_h	order of the algebraic group
x_k	exponentiation base; generator in the algebraic group delimited by b_h
p_h, q_h	prime integers
$K'^t_{i,j}, SK'_h$	updated keys

3.2 Architecture

In the MM-MSKMS, the KDC maintains a key forest to organize the joined users according to their membership. The key forest will be composed by different key trees, each one associated with a group of users who have an exact match on their memberships. In other words, each tree represents a group of users who have a full overlapping in their sessions. Each group of users with overlapped sessions is named as *Overlapping Group* (OG). In this paper, in order to use a general notation, even those groups where users are involved with a single session are called OG.

In a system where there are s sessions, there will at most $m(s)$ Overlapping Groups, where $m(s)$ is determined by:

$$m(s) = \sum_{\gamma=1}^{s} \binom{s}{\gamma} \quad (1)$$

Thus, the key forest is formed by m_0 trees, with $1 \leq m_0 \leq m(s)$ trees (see Figure 3). Each tree is formed by two kinds of keys: the Key Encryption Keys (KEKs) and the Session Keys (SKs). KEKs are used as auxiliary keys for the rekeying operations. SKs are used to generate independent Data Encryption Keys (DEKs), which are used to encrypt and decrypt information related with sessions. In a system with s sessions, there are s SKs, one for each session.

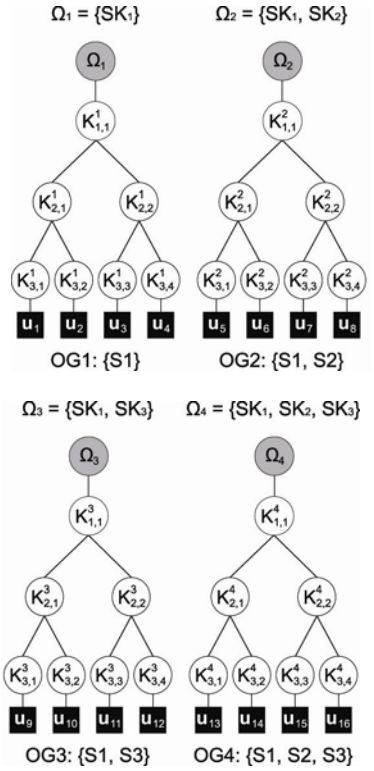

Figure 3: Key forest for 4 OGs related with three sessions

KEKs are organized in balanced trees, where each key is denoted as $K_{i,j}^{t}$, where t indicates the OG associated with the tree ($1 \leq t \leq m_0$), i indicates the vertex level, and j indicates the most left position relative to level i. KEKs situated in the $K_{1,1}^{t}$ position are called root-KEKs (rKEKs) and are used to distribute the SKs, while KEKs in the lowest level are the individual keys of the OG members.

All the SKs associated with the sessions related with an OG_t form a set Ω_t, which is represented by an additional vertex in a level above the KEK-tree.

With such key organization, each OG member must store all the keys along the tree path where it is joined, from its individual key to the rKEK, along with the SKs in the corresponding Ω_t. Unlike KEKs and SKs, DEKs are not stored by users because they are generated before the transmission of a packet.

An example of a key forest structure used to organize different OGs is shown in Figure 3. In such example, it is assumed that in the system there are 16 users grouped according to their memberships into 4 of the 7 possible OGs, related with three different sessions. Thus, the members of OG_1 have access to the information of session S_1, using SK_1; the members of OG_2 have access to the information of sessions S_1 and S_2, using SK_1 and SK_2, the members of OG_3 have access to the information of sessions S_1 and S_3, using SK_1 and SK_3, while the members of OG_4 have access to the information of sessions S_1, S_2 and S_3, using SK_1, SK_2 and SK_3.

3.3 Key generation

As we mentioned in Section 3.2, members in each OG store two kinds of keys: KEKs and SKs. KEKs are keys generated through a derivation technique to avoid having the KDC generate, encrypt and transmit all the keys related to the rekeying. SKs are keys designed to allow the users to generate independent DEKs and to transmit multimedia packets, using one key per packet.

3.3.1 Generation of KEKs

For the generation of KEKs, our mechanism uses a key derivation technique similar to the one defined in the SKD protocol [15]. With this technique, each user can compute the new KEKs using a function $f(\cdot)$ and previous keys. Thus, the KDC only has to transmit the keys that some users cannot derive, decreasing the computational effort in KDC and the use of bandwidth. For key derivation, function $f(\cdot)$ could be a one-way function, a pseudo-random number generator or a trap-door function.

3.3.2 Generation of SKs

Each SK_h $(1 \leq h \leq s)$ is a packet formed by variables used by the Blum Blum Shub algorithm (BBS) [16]. The BBS algorithm has the purpose of generating a pseudo-random number series free of patterns that can be discovered with any reasonable amount of calculations. With some bits of each of the generated numbers, issuers construct an individual DEK to encrypt a packet. Receptors can recover any DEK generating the corresponding number series from the packet index and a *seed*. Thus, users can use individual keys to encrypt each transmitted packet.

With the BBS algorithm, each number is generated by:

$$x_{k+1} = (x_k)^2 \bmod b, \qquad (2)$$

where $b = pq$, being p and q two large primes congruent with $3 \bmod 4$ ($p \equiv 3 \bmod 4$ and $q \equiv 3 \bmod 4$).

To start the number generation, a *seed* must be chosen; such seed can be a random number x_0 that is a relative prime with b. Knowing x_0, any user can compute the kth generated number using the equation:

$$x_k = x_0^{(2^k \bmod ((p-1)(q-1)))} \bmod b. \qquad (3)$$

Thus, each key SK_h will be a packet formed by the variables p_h, q_h, x_{0_h} and b_h, which will be computed by the KDC.

3.3.3 Generation of DEKs

Generating independent DEKs. When an user starts the transmission of multimedia information for each packet P_r $(r = 1,2,\ldots)$, the user locally generates an independent DEK in the following way:

1. Using the variables of $SK_h = \{p_h, q_h, x_{0_h}, b_h\}$ and equation (2), the user generates a pseudo random number

series of k elements, depending upon the required key size (for example, if the system uses AES-128, then one series of 64 elements should be generated).

2. At most $log_2 log_2 b_h$ bits of each generated number are taken to form k bit sequences.
3. DEK is formed concatenating the k bit sequences.

Finally, each packet is encrypted with the generated DEK, using the specified cypher algorithm.

Recovering independent DEKs. When a user receives a packet P_r, that user will use the packet index to recover the DEK to decrypt information in the following way:

1. Using the variables of $SK_h = \{p_h, q_h, x_{0_h}, b_h\}$ and equation (3), the user locally generates a pseudo number series of k elements, starting at element $(r-1)k+1$
2. At most, $log_2 log_2 b_h$ bits of each generated number are taken to form k bit sequences.
3. The corresponding DEK is created concatenating the k bit sequences.

Finally, each packet is decrypted with the DEK, using the specified cypher algorithm.

3.4 Rekeying operations

Rekeying operations must be started by the KDC when a membership change takes place. We understand as a membership change when a user joins an OG or leaves any OG in order to leave the whole system or simply to change its OGs and its joined sessions.

3.4.1 User join

When a user requests to join the system, the KDC decides which OG must hold that user, according to its requested sessions. Then, the KDC randomly generates an individual key for the new member and sends it through a secure channel. By secure channel we mean any unicast communications channel secured using a unicast security protocol. The particular protocol is not important; any unicast security protocol that provides mutual authentication with key exchange can be used. Moreover, the KDC updates the compromised KEKs and SKs.

Updating KEKs. The KDC assigns a new vertex in the KEK-tree to store the individual key of the new member. As each KEK-tree maintained by the KDC must be balanced, each new vertex is inserted in the shortest paths of the KEK-tree.

Assuming that K^t_{v,j_v}, the root vertex of KEK subtree X^t_{v,j_v} is the last internal vertex on the joined path:

- if X^t_{v,j_v} is not full, the new vertex $K'^t_{v+1,j_{v+1}}$ is inserted,
- if X^t_{v,j_v} is full, the left most vertex $K^t_{v+1,j_{v+1}}$ is moved to a lower level, becoming the new vertex $K'^t_{v+2,j_{v+2}}$ and its old position is replaced by a new intermediate vertex $K'^t_{v+1,j_{v+1}}$. Thus $K'^t_{v+1,j_{v+1}}$ will be the parent of $K'^t_{v+2,j_{v+2}}$ and the vertex associated with the individual key of the new user, $K^t_{v+2,j_{v+2}+1}$.

In both cases, all the new KEKs, found in unchanged vertices between the vertex asociated with the key of the new user and $K^t_{1,1}$, are computed by:

$$K'^t_{i,j_i} = f(K^t_{i,j_i}),$$

where K^t_{i,j_i} is the previous KEK of that position, named as derivation key.

If the new intermediate vertex $K'^t_{v+1,j_{v+1}}$ is inserted, the new KEK is computed by:

$$K'^t_{v+1,j_{v+1}} = f(K'^t_{v+2,j_{v+2}} \oplus K^t_{1,1}),$$

where $K'^t_{v+2,j_{v+2}}$, the previous KEK $K^t_{v+1,j_{v+1}}$ is the derivation key, while the rKEK $K^t_{1,1}$, named the *salt* value, is used to ensure that the derived key is different even when the same derivation key is used since $K^t_{1,1}$ will be different each time.

For the remaining OG members, the KDC multicasts a message to inform the position of the new user. Thus, each user can compute the necessary KEKs. As the new user does not know the KEKs involved with its path, the KDC sends to it a unicast message with the related keys.

Consider the OG_2 of the system shown in Figure 3. Suppose that user u_{17} joins the system. The KDC moves the vertex associated with the individual key of user u_5 to a lower level, and replaces that position with the new vertex $K'^2_{3,1}$. With this modification, the new intermediate vertex is the parent of $K^2_{4,1}$ and $K^2_{4,2}$ (see Figure 4), where $K^2_{4,2}$ is the individual key of the new user.

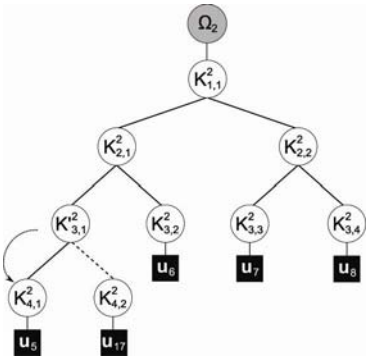

Figure 4: An example of user join

The compromised KEKs are recomputed by:

$$K'^2_{1,1} = f(K^2_{1,1}),$$
$$K'^2_{2,1} = f(K^2_{2,1}),$$
$$K'^2_{3,1} = f\left(K^2_{4,1} \oplus K^2_{1,1}\right).$$

The new user u_{17} cannot derive the KEKs related with its path. For this reason, the KDC unicasts such keys in the following way:

$$KDC \rightarrow u_{17}: \{K'^2_{1,1}\}_{K'^2_{2,1}} \parallel \{K'^2_{2,1}\}_{K'^2_{3,1}} \parallel \{K'^2_{3,1}\}_{K^2_{4,2}}$$

Updating SKs. Once the updating of the corresponding

KEKs has finished, the KDC updates the SKs of the set Ω_t to preserve the backward secrecy.

To generate each SK_h, the KDC generates the necessary variables for the BBS algorithm: two primes p_h and q_h, congruent with 3mod4 and one number x_{0_h}, relative prime with $b_h = p_h q_h$, which will be the seed of the BBS generator. Thus, the new key SK_h is the packet $\{p_h, q_h, x_{0_h}, b_h\}$.

To finish the rekeying, the KDC multicasts the new SKs to all the involved OGs, encrypting each packet with the corresponding rKEKs.

In the example shown in Figure 4, as OG_2 members are involved in sessions S_1 and S_2, the KDC has to generate the new SK_1 and SK_2, which are elements of Ω_2. Finally, the KDC multicasts the new SKs to the OGs involved with these keys, using the corresponding rKEKs to encrypt those messages. The KDC transmits the new SKs through the following messages:

$$KDC \rightarrow OG_1: \{SK'_1\}_{K^1_{1,1}}$$
$$KDC \rightarrow OG_2: \{SK'_1\}_{K^2_{1,1}} \parallel \{SK'_2\}_{K^2_{1,1}}$$
$$KDC \rightarrow OG_3: \{SK'_1\}_{K^3_{1,1}}$$
$$KDC \rightarrow OG_4: \{SK'_1\}_{K^4_{1,1}} \parallel \{SK'_2\}_{K^4_{1,1}}$$

The rekeying for the user join process is detailed in Algorithms 1 and 2.

Algorithm 1 User join algorithm on KDC's side

Input: join_request_message($user, \Theta$) /*Θ *is a set with the requested sessions*/*
Output: Updated Keys
Ω_{new_user} =get_related_SKs_with(Θ)
t = choose_an_OG_where($\Omega_t = \Omega_{new\_user}$)
$user\_key$ = generate_key()
unicast($user\_key, user$)
$height$ = get_height_of(t)
(i, j) = get_last_internal_vertex(t)
if subtree(i, j) is not full **then** */*verifies if the last internal vertex can hold a new vertex*/*
  (i, j) = get_right_most_leaf($t, height + 1$)
  $K^t_{i,j+1} = user\_key$
else
  if $j < d^{height-1}$ **then** */*insert a new vertex under the next available vertex*/*
    (i, j) = get_left_most_leaf($t, height$)
  else */*create a new KEK-tree level*/*
    (i, j) = get_left_most_leaf($t, height + 1$)
  end if
  */*new intermediate key derivation*/*
  $K^t_{i+1,d(j-1)+1} = K^t_{i,j}$
  $K^t_{i,j} = f(K^t_{i,j} \oplus K^t_{1,1})$
  $K^t_{i+1,d(j-1)+2} = user\_key$
end if
multicast(join_notification($i + 1, d(j-1) + 2), OG_t$)
$i = i - 1$
while $i > 0$ **do** */*update of the compromised KEKs*/*
  $j = \lceil j/d \rceil$
  $K^t_{i,j} = f(K^t_{i,j})$
  $i = i - 1$
end while
unicast(updated_KEKs(), $user$)
for each $SK_h \in \Omega_t$ **do** */*update of the compromised SKs*/*
  (p, q) =generate_two_primes_congruent_with(3mod4)
  $x =$ generate_a_relative_prime_with($b = pq$)
  $SK_h = \{p, q, x, b\}$
end for
for each $\beta_g \in \{OG_a | \Omega_a \cap \Omega_t \neq \emptyset \wedge a \in [1, m_0]\}$ **do**
  multicast($\{\Omega_g \cap \Omega_t\}_{K^g_{1,1}}, \beta_g$)
end for

Algorithm 2 User join algorithm on user's side

Input: join_notification(i, j)
Output: Updated Keys
$x = i - 1$
$y = \lceil j/d \rceil$
*/*verifies if a new intermediate vertex has been inserted*/*
if local_user_individual_key()= $K^t_{x,y}$ **then**
  */*new intermediate key derivation*/*
  $K^t_{x+1,d(j-1)+1}$ =local_user_individual_key()
  $K^t_{x,y} = f(K^t_{x,y} \oplus K^t_{1,1})$
  $i = i - 1$
  $j = \lceil j/d \rceil$
end if
$i = i - 1$
while $i > 0$ **do** */*update of the compromised KEKs*/*
  $j = \lceil j/d \rceil$
  if local_user_holds($\{K^t_{i,j}\}$) **then**
    $K^t_{i,j} = f(K^t_{i,j})$
  end if
  $i = i - 1$
end while
*/*update of the compromised SKs*/*
wait_until_the_reception_of($\{\Omega_t\}_{K^t_{1,1}}$)
decrypt($\{\Omega_t\}_{K^t_{1,1}}$)

3.4.2 User leave

Updating KEKs. When a user leaves the system, the KDC removes the corresponding vertex in the KEK-tree of the affected OG and updates the compromised keys.

Assuming that K^t_{v,j_v} is the root vertex of the affected KEK subtree X^t_{v,j_v}, the KEKs updating is performed in one of two ways:

- if K^t_{v,j_v} has at least two children, K^t_{v,j_v} is only updated.
- if K^t_{v,j_v} has only a child, K^t_{v,j_v} is replaced by its child ($K'^t_{v,j_v} = K^t_{v+1,j_{v+1}}$).

In both cases, the new KEKs K'^{t}_{i,j_i} of the compromised path are computed using the previous keys K^{t}_{i,j_i} along with the left most key of the lower level $i+1$, located in the opposite path of the removed vertex, as follows:

$$K'^{t}_{i,j_i} = f(K^{t}_{i+1,j_{i+1}} \oplus K^{t}_{i,j_i}),$$

where $K^{t}_{i+1,j_{i+1}}$ is the derivation key and the previous key K^{t}_{i,j_i} is used as *salt* value.

For the remaining OG members, the KDC multicasts a message to inform the position of the removed user. Thus, each user can start the rekeying.

As the derivation strategy only benefits users in the opposite path of the removed vertex, the KDC has to send the updated KEKs to users that cannot derive those keys.

Consider the OG_2 of Figure 3. Assuming that user u_{17} leaves the system, the KDC modifies the KEK-tree, moving the vertex $K^{2}_{4,1}$ to an upper level, replacing the vertex $K^{2}_{3,1}$ as shown in Figure 5.

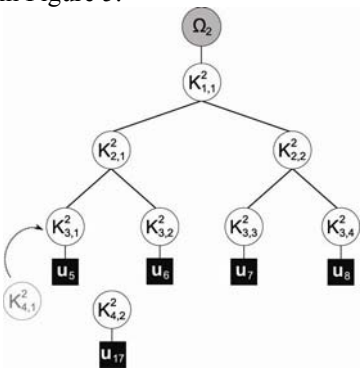

Figure 5: An example of user leave

The new KEKs are computed by:

$$K'^{2}_{1,1} = f(K^{2}_{2,2} \oplus K^{2}_{1,1}),$$
$$K'^{2}_{2,1} = f(K^{2}_{3,2} \oplus K^{2}_{2,1}).$$

Since not all users can derive the keys, the KDC sends the following messages to complete the updating process of KEKs:

$$KDC \rightarrow u_5 : \{K'^{2}_{2,1}\}_{K^{2}_{3,1}}$$
$$KDC \rightarrow u_5 - u_6 : \{K'^{2}_{1,1}\}_{K^{2}_{2,1}}$$

Updating SKs. To finish the rekeying, the KDC updates the SKs of the set Ω_t to preserve the forward secrecy. For each $SK_h \in \Omega_t$, the KDC computes the corresponding values p_h, q_h, x_{0_h} and b_h, and then transmits the new SKs to all the involved OGs, encrypting each packet with the corresponding rKEKs.

In the example shown in Figure 5, to finish the rekeying, the KDC updates the SKs of the set Ω_2, and multicasts those keys to the members in the affected OGs through the following messages:

$$KDC \rightarrow OG_1 : \{SK'_1\}_{K^{1}_{1,1}}$$
$$KDC \rightarrow OG_2 : \{SK'_1\}_{K^{2}_{1,1}} \| \{SK'_2\}_{K^{2}_{1,1}}$$
$$KDC \rightarrow OG_3 : \{SK'_1\}_{K^{3}_{1,1}}$$
$$KDC \rightarrow OG_4 : \{SK'_1\}_{K^{4}_{1,1}} \| \{SK'_2\}_{K^{4}_{1,1}}$$

The rekeying for the user leave process is detailed in Algorithms 3 and 4.

Algorithm 3 User leave algorithm on KDC's side

Input: leave_request_message(OG_t, i, j)
Output: Updated Keys
multicast(leave_notification$(i,j), OG_t$)
delete_vertex(i,j,t)
if number_of_children_of$(K^{t}_{i-1,\lceil j/d \rceil}) = 1$ **then**
  $K^{t}_{i-1,\lceil j/d \rceil} = K^{t}_{i,j}$ */*move the key to a upper level*/*
  $i = i - 1$
  $j = \lceil j/d \rceil$
end if
$y = i$
while $i > 1$ **do** */*update of the compromised KEKs*/*
  (h,v) =get_left_most_sibling_of(i,j,t)
  $K^{t}_{i-1,j/d} = f(K^{t}_{h,v} \oplus K^{t}_{i-1,\lceil j/d \rceil})$ */*KEKs derivation*/*
  $i = i - 1$
  $j = \lceil j/d \rceil$
end while
multicast(updated_KEKs(), users_that_cannot_derive())
for each $SK_h \in \Omega_t$ **do** */*update of the compromised SKs*/*
  (p,q) =generate_two_primes_congruent_with (3mod4)
  $x =$ generate_a_relative_prime_with$(b = pq)$
  $SK_h = \{p, q, x, b\}$
end for
for each $\beta_g \in \{OG_a | \Omega_a \cap \Omega_t \neq \emptyset \wedge a \in [1, m_0]\}$ **do**
  multicast$(\{\Omega_g \cap \Omega_t\}_{K^{g}_{1,1}}, \beta_g)$
end for

Algorithm 4 User leave algorithm on user's side

Input: leave_notification(i,j)
Output: Updated Keys
  $i = i - 1$
  $j = \lceil j/d \rceil$
while $i > 1$ **do** */*update of the compromised SKs*/*
  (h,v) =get_left_most_sibling_of(i,j)
  if local_user_ holds$(\{K^{t}_{i-1,\lceil j/d \rceil}, K^{t}_{h,v}\})$ **then**
    $K^{t}_{i-1,\lceil j/d \rceil} = f(K^{t}_{h,v} \oplus K^{t}_{i-1,\lceil j/d \rceil})$ */*KEKs derivation*/*
  end if
  $i = i - 1$
  $j = \lceil j/d \rceil$
end while
*/*update of the compromised SKs*/*
wait_until_the_reception_of$(\{\Omega_t\}_{K^{t}_{1,1}})$
decrypt$(\{\Omega_t\}_{K^{t}_{1,1}})$

3.4.3 User switch

When a user requires to leave an OG_y to join an OG_z, the KDC has to modify the KEK-trees of the affected OGs and update the compromised SKs.

The updating of KEKs is performed as described in Sections 3.4.1 and 3.4.2. For the KEK-tree of OG_y, the operations related with the user leave event will be performed, while for the KEK-tree of OG_z, the process will be similar to the user join event, with the only difference being that the KDC does not assign a new individual key to the user. The KDC only modifies the user key index in order to incorporate it into the new KEK-tree.

To finish the rekeying, the KDC updates the SKs that sets Ω_y and Ω_z do not have in common. In other words, the KDC updates the SKs in $\Omega_y \Delta \Omega_z$. The renewal of the SKs in the symmetric difference of Ω_y and Ω_z, is intended to ensure backward and forward secrecy in each of the system's OGs, using the common SKs in order not to raise the rekeying overhead.

After the KDC computes the new SKs, those keys will be sent to all the memebers of the OGs involved with the SKs.

For example, consider the OG_2 and the OG_4 in Figure 3. Assuming that user u_{15} leaves the OG_4 in order to join the OG_2, first, the KDC modifies the KEK-tree of OG_4, removing the corresponding vertex of the individual user key. Then, the KDC modifies the KEK-tree of the OG_2 in order to asign a new vertex for the individual key of u_{15}. We illustrate this process in Figure 6.

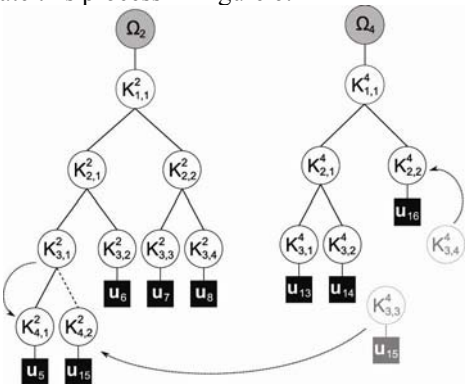

Figure 6: An example of user switch

Compromised KEKs are computed by:

$$K'^4_{1,1} = f(K^4_{2,1} \oplus K^4_{1,1})$$
$$K'^2_{1,1} = f(K^2_{1,1})$$
$$K'^2_{2,1} = f(K^2_{2,1})$$
$$K'^2_{3,1} = f(K^2_{4,1} \oplus K^2_{1,1})$$

As users u_{15} and u_{16} cannot derive the KEKs, the KDC sends those keys through the following messages:

$$KDC \rightarrow u_{16}: \{K'^4_{1,1}\}_{K^4_{2,2}}$$
$$KDC \rightarrow u_{15}: \{K'^2_{1,1}\}_{K'^2_{2,1}} \parallel \{K'^2_{2,1}\}_{K'^2_{3,1}} \parallel \{K'^2_{3,1}\}_{K^2_{4,2}}$$

To finish the rekeying, the KDC computes the new SK_3, which is the key in $\Omega_2 \Delta \Omega_4$. Then the KDC transmits the new SK'_3 to all the members of OG_3 and OG_4, using the following messages:

$$KDC \rightarrow OG_3: \{SK'_3\}_{K^3_{1,1}},$$
$$KDC \rightarrow OG_4: \{SK'_3\}_{K^4_{1,1}}$$

4 Performance analysis

In this section we analyze the performance of the MM-MSKMS, focusing on storage and rekey overheads in order to demonstrate the efficiency of our solution. Then, we compared it directly with DACMGS, which is the work we take as main reference.

4.1 Storage overhead

As we mentioned above, in the MM-MSKMS we use trees as storage structures to organize the keys and the members of the different OGs present in the system. Particularly, trees used in this work can be viewed as a graph composed of a KEK-tree connected with an additional vertex used to store the SKs. Each KEK-tree is maintained as balanced as possible by positioning the joining users on the shortest paths.

Let n denote the number of users joined at the whole system and n_t the number of users involved in a tree (OG). We use $l_d(n_t)$ to denote the length of the branches of a tree of d degree. Since each KEK-tree is balanced and it is possible that not all the branches have the same length at some point, $l_d(n_t)$ is either L or $L+1$, where $L = \log_d n_t$. Particularly,

- the number of users who are on branches with length L is $d^L - \left\lceil \frac{n_t - d^L}{d-1} \right\rceil$,
- and the number of users who are on branches with length $L+1$ is $n_t - d^L + \left\lceil \frac{n_t - d^L}{d-1} \right\rceil$.

Therefore, the total number of keys in a KEK-tree is determined by:

$$TK(n_t) = n_t + \frac{d^L - 1}{d-1} + \left\lceil \frac{n_t - d^L}{d-1} \right\rceil \quad (4)$$

As the KDC holds the s SKs related to the system sessions and maintains the KEK-trees of the $m(s)$ OGs, the total number of keys stored by the KDC is determined by:

$$TK_{KDC} = \sum_{t=1}^{m(s)} TK(n_t) + s \quad (5)$$

Each user joined at an OG_t has to store the $l_d(n_t)$ KEKs involved with its branch and the $|\Omega_t|$ SKs related with its OG. Thus, the total keys stored by each user is determined by:

$$TK_{u \in OG_t} = l_d(n_t) + |\Omega_t| \quad (6)$$

Assuming the worst case, where all the combinations of the s sessions exist, we can take $m(s)$ as a fixed value ($m(s) = m_0$) throughout the communication process. If we also assume that all the OGs have the same number of users ($n_t = n_0$), the number of users in the whole system is $n = m_0 \cdot n_0$. Using (5), the KDC's storage overhead is calculated as:

$$TK_{KDC} = m_0 \cdot TK(n_0) + s \quad (7)$$

Using (6), we have that the user's storage overhead is:

$$TK_{u \in OG_t} = l_d(n_0) + |\Omega_t| \quad (8)$$

From (4), we have that $\lim_{n_0 \to \infty} TK(n_0) = \frac{d}{d-1} n_0$. Therefore, as s is a fixed value throughout the communcation process and $s \ll n_0$ when $n_0 \to \infty$, using (7) we can calculate the KDC's asymptotic storage overhead as:

$$TK_{KDC} \sim O\left(\frac{d}{d-1} m_0 \cdot n_0\right) = O\left(\frac{d}{d-1} n\right) \quad (9)$$

Since $|\Omega_t|$ is fixed for each OG, using (8) we can calculate the user's asymptotic storage overhead as:

$$TK_{u \in OG_t} \sim O(\log_d n_0) \quad (10)$$

4.2 Communication overhead

The communication overhead is determined by the number of messages transmitted at rekey operations. Therefore, we estimate the number of messages involved in the different rekeying processes (join, leave and switch) to determine the communication overhead.

Let $m(s_r)$, the number of OGs involved with the updated SKs in a rekeying operation. When a new user joins the system, the KDC unicasts to the new user a message with all the KEKs of its branch and the SKs related to its OG. Moreover, the KDC multicasts a join notification with the information of the join branch to the remaining users, and $m(s_r)$ messages to transmit the SKs to the involved OGs. Therefore, the number of messages sent out by the KDC is determined by:

$$M_{join} = m(s_r) + 2 \quad (11)$$

As each OG_t is related to $|\Omega_t|$ SKs, the total number of SKs sent by the KDC to the $m(s_r)$ involved groups in a rekey operation is determined by:

$$N_K = \sum_{l=1}^{m(s_r)} |\Omega_t \cap \Omega_l| \quad (12)$$

where Ω_t denotes the set of SKs related to an OG_t and Ω_l denotes the set of SKs related to an OG_l.

Therefore, the number of keys sent out by the KDC in the user join process, is determined by the following equations:

$$NK_{J_{unicast}} = l_d(n_t) + |\Omega_t| \quad (13)$$

$$NK_{J_{multicast}} = N_K \quad (14)$$

where $NK_{J_{unicast}}$ denotes the number of keys sent out in a unicast way and $NK_{J_{multicast}}$ denotes the number of keys sent out in a multicast way.

When a user leaves the system, the KDC multicasts $(d-1)l_d(n_t)$ messages with the KEKs of the affected branch to the users which cannot derive them, a message with a leave notification, and also multicasts $m(s_r)$ messages with the updated SKs to the involved OGs. Therefore, the number of messages sent out by the KDC is determined by:

$$M_{leave} = (d-1)l_d(n_t) + m(s_r) + 1 \quad (15)$$

As for the user leave process only the remaining users are involved, the KDC does not send any message in a unicast communication. Thus, using 12 the number of keys sent out by the KDC in the user leave process, is determined by:

$$NK_{L_{multicast}} = (d-1)l_d(n_t) + N_K \quad (16)$$

As the rekey for the user switch process involves the join and leave processes, from (11) and (15) we know that $(d-1)l_d(n_t) + 3$ messages are necessary to update the compromised KEKs, 2 messages to update the KEKs in the joined group and $(d-1)l_d(n_t) + 1$ to update the KEKs of the left group. Assuming that a user switches from OG_y to OG_z, the KDC has to update the SKs in $\Omega_y \Delta \Omega_z$. Let $m(s_r)$ be the number of messages to update such SKs, the number of messages sent out by the KDC is determined by:

$$M_{switch} = (d-1)l_d(n_t) + m(s_r) + 3 \quad (17)$$

Using 12, 13, 14 and 16, the total number of keys sent out by the KDC in the rekeying, needed for the user switch process is determined by:

$$NK_{S_{unicast}} = l_d(n_t) + |\Omega_t| \quad (18)$$

$$NK_{S_{multicast}} = dl_d(n_t) + N_K \quad (19)$$

In this case, N_K involves the number of SKs sent out by the KDC to the $m(s_r)$ groups, related to the $|\Omega_y \Delta \Omega_z|$ updated SKs.

As the switch process involves join and leave operations, we can use it to determine the highest bound of transmitted messages.

Assuming the worst case, when the user switch process involves an OG related to all the s sessions, and an OG related with one session, we have that $m(s_r) = m(s-1)$. Furthermore, if we also assume that all the OGs have the same number of users, $n_t = n_0$, using (17) the total number of messages involved in the rekeying process is given by:

$$M = (d-1)l_d(n_0) + m(s-1) + 3 \quad (20)$$

If $n_0 \to \infty$, as s is fixed throughout the communication process and $m(s-1) < n_0$, we can see that the asymptotic communication overhead is:

$$M \sim O\big(d \log_d(n_0)\big) \quad (21)$$

4.3 Comparison

In this section we compare the CMGKMS [8], DACMGS [12] and MM-MSKMS, focusing on two measures: the storage overhead and the communication overhead. The communication overhead is compared using the costs of join, leave and switch processes separately.

In Table 2 we summarize the measurements, which are expressed in bits. These results are based on the results of CMGKMS and DACMGS, and on the results obtained in

Table 2 Performance comparison for storage and communication overhead

		CMGKMS	DACMGS	MM-MSKMS
Storage cost				
KDC		$(\frac{d}{d-1}n + 2s)S_K$	$(\frac{d}{d-1}n + s)S_K$	$\frac{d}{d-1}nS_K + sS_{sk}$
User		$(\log_d(n_t) + N_K + 1)S_K$	$(\log_d(n_t) + \lvert\Omega_t\rvert)S_K$	$\log_d(n_t)S_K + \lvert\Omega_t\rvert S_{sk}$
Communication Cost				
Join	Unicast	$(\log_d(n_t) + N_K + 1)S_K$	$(\log_d(n_t) + 1)S_K$	$\log_d(n_t)S_K + \lvert\Omega_t\rvert S_{sk}$
	Multicast	0	0	$\log_d(n_t) + N_K S_{sk}$
Leave	Unicast	0	0	0
	Multicast	$(d\log_d(n_t) + N_K)S_K$	$d(\log_d(n_t) - 1)S_K + N_K S_{ck}$	$(d-1)\log_d(n_t)S_K + N_K S_{sk}$
Switch	Unicast	$(\log_d(n_t) + N_K + 1)S_K$	$(\log_d(n_t) + 1)S_K$	$\log_d(n_t)S_K + \lvert\Omega_t\rvert S_{sk}$
	Multicast	$(d\log_d(n_t) + N_K)S_K$	$d(\log_d(n_t) - 1)S_K + N_K S_{ck}$	$d\log_d(n_t)S_K + N_K S_{sk}$

Sections 4.2 and 4.1. In Table 2, S_K denotes the KEK's and the TEK's size, S_{ck} denotes the size of a secret that is smaller than a KEK, while S_{sk} denotes the size of a SK. If we use a cryptosystem with KEKs of 128 bits, the size of the SKs should be $O(192)$ bits, using the BBS algorithm with 32-bit integers. In addition, we use N_K to denote the number of keys that the KDC has to distribute and are related to the sessions or the privileges of a user. In the case of DACMGS and MM-MSKMS, N_K is the number of keys sent by the KDC to the $m(s_r)$ involved groups, $N_K = \sum_{a=1}^{m(s_r)} |\Omega_t \cap \Omega_a|$. For CMGKMS, N_K is approximately two times the number of resources that the user can access ($N_k \approx 2|\Omega_t|$).

In Table 2 we can observe that CMGKMS and DACMGS have lower communication costs than MM-MSKMS. However, those costs are lower because CMGKMS and DACMGS are designed for users who work only as receivers in a 1 to n communication. For that reason, the KDC does not have to transmit the new keys in the rekeying process because the new version of the keys are indicated in the received packets. On the other hand, MM-MSKMS is designed for n to m communications which support the delay, the loss and the transposition of packets. In addition, each user is a transceiver entity that generates independent keys from the SKs, sent by the KDC. For this reason, the KDC has to transmit some of the updated keys to avoid inconsistencies in transmissions. However, the cost of MM-MSKMS does not differ significantly compared with the cost of CMGKMS and DACMGS.

5 Conclusion

In this paper we have proposed an efficient multi-session key management scheme for dynamic multimedia group communication. The proposed scheme is characterized by the use of an independent key per cipher packet and allows the users to exchange streams between them in an *n* to *m* communication. Our solution proposes a functional architecture that exploits the overlapping of the user sessions to reduce the redundancy in key distribution, and makes use of two key generation strategies: a key derivation technique to reduce the rekey overhead and a pseudorandom number generator that allows the users to generate independent keys for each transmitted packet. The proposed mechanism offers good storage and communication costs, comparable with the existing mechanisms based on multi-privileged groups. According to our knowledge, the MM-MSKMS presented in this paper is the only one oriented towards the support of multi-group multimedia environment with *n* to *m* communication. The MM-MSKMS can be used for environments as presented by [17].

References

[1] Dapeng Wu, Yiwei Thomas Hou, Wenwu Zhu, Member, Ya-Qin Zhang, Jon M. Peha, Streaming Video over the Internet: Approaches and Directions, IEEE Transactions on circuits and systems for video technology, Vol. 11, No. 3, 2001, pp. 282-300.

[2] Thomas Hardjono, Lakshminath R. Dondeti, Multicast and Group Security, Artech House, 2003, pp. 17-43.

[3] Borko Furht, Darko Kirovski, Multimedia Security Handbook, CRC Press, 2004, pp. 100-120.

[4] Alfred J. Menezes, Paul C. van Oorschot, Scout Vanstone, Handbook of Applied Cryptography, CRC Press, 1996, pp. 230-260.

[5] Sandro Rafaeli, David Hutchison, A Survey of Key Management for Secure Group Communication, ACM Computing Surveys, 2003, pp. 309-329.

[6] Hao-hua Chu, Lintian Qiao, Klara Nahrstedt, A Secure Multicast Protocol with Copyright Protection, ACM SIGCOMM Computer Communications Review, Volume 32 , Issue 2, 2002, pp. 42-60.

[7] Yan Sun, K. J. Ray Liu, Multi-Layer Key Management for Secure Multimedia Multicast Communications, Proceedings of the 2003 International Conference on Multimedia and Expo (ICME 2003), Vol. 1, IEEE

Computer Society Press, 2003, pp. 205-208.
[8] Yan Sun, K. J. Ray Liu, Scalable Hierarchical Access Control in Secure Group Communications, Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004), Vol. 2, IEEE, 2004, pp. 1296-1306.
[9] Marcel Waldvogel, Germano Caronni, Dan Sun, Nathalie Weiler, Bernhard Plattner, The VersaKey Framework: Versatile Group Key Management, IEEE Journal on Selected Areas in Communications, Vol. 17, IEEE, 1999, pp. 1614-1631.
[10] Scott Dexter, Roman Belostotskiy, Ahmet M. Eskicioglu, Multi-layer multicast key management with threshold cryptography, Proceedings of SPIE Security and Watermarking of Multimedia Contents VI, Vol. 5306, SPIE, 2004.
[11] Guojun Wang, Jie Ouyang, Hsiao-Hwa Chen, Minyi Guo, Efficient group key management for multi-privileged groups, Computer Communications, Vol. 30, Issue 11-12, Elsevier, 2007, pp. 2497-2509.
[12] Di Ma, Robert H. Deng, Yongdong Wu, Tieyan Li, Dynamic Access Control for Multi-Privileged Group Communications, 6th International Conference on Information and Communications Security (ICICS 2004), Lecture Notes in Computer Science (LNCS) 3269, Springer-Verlag, 2004, pp. pp. 508-519.
[13] Ruidong Li, Jie Li, and Hisao Kameda, Distributed Hierarchical Access Control for Secure Group Communications, International Conference on Computer Network and Mobile Computing 2005 (ICCNMC 2005), Lecture Notes in Computer Science (LNCS) 3619, Springer-Verlag, 2005, pp. 539-548.
[14] Qijun Gu, Peng Liu, Wang-Chien Lee, Chao-Hsien Chu, KTR: an efficient key management scheme for secure data acces control in wireless broadcast services, IEEE Trans. Dependable Sec. Comput., 2009, pp. 188-201.
[15] Jen-Chiun Lin, Kuo-Hsuan Huang, Feipei Lai, Hung-Chang Lee, Secure and efficient group key management with shared key derivation, Computer Standars and Interfaces, Vol. 31, Issue 1, Elsevier, 2009, pp. 192-208.
[16] Lenore Blum, Manuel Blum, Michael Shub, A Simple Unpredictable Pseudo-Random Number Generator, SIAM Journal on Computing, Vol. 15, No. 2, 1986, pp. 364-383.
[17] Bo Cheng, Xiaoxiao Hu, Junliang Chen, Design and Implementation for Multimedia Conferencing Communication Services and Orchestration on Internet, Journal of Internet Technology, Vol. 12 No. 6, pp. 865-874.