SAKM: A Scalable and Adaptive Key Management Approach for Multicast Communications

Yacine Challal  
Compiegne University of Technology  
Heudiasyc lab. France  
yacine.challal@utc.fr

Hatem Bettahar  
Compiegne University of Technology  
Heudiasyc lab. France  
hatem.bettahar@utc.fr

Abdelmadjid Bouabdallah  
Compiegne University of Technology  
Heudiasyc lab. France  
madjid.bouabdallah@utc.fr

ABSTRACT
Multicasting is increasingly used as an efficient communication mechanism for group-oriented applications in the Internet. In order to offer secrecy for multicast applications, the traffic encryption key has to be changed whenever a user joins or leaves the system. Such a change has to be communicated to all the current users. The bandwidth used for such rekeying operation could be high when the group size is large. The proposed solutions to cope with this limitation, commonly called 1 affects n phenomenon, consist of organizing group members into subgroups that use independent traffic encryption keys. This kind of solutions introduce a new challenge which is the requirement of decrypting and reencrypting multicast messages whenever they pass from one subgroup to another. This is a serious drawback for applications that require real-time communication such as video-conferencing. In order to avoid the systematic decryption / reencryption of messages, we propose in this paper an adaptive solution which structures group members into clusters according to the application requirements in term of synchronization and the membership change behavior in the secure session. Simulation results show that our solution is efficient and typically adaptive compared to other schemes.

Keywords
Multicat, Security, Key Management, Scalability

1. INTRODUCTION
Multicasting is an efficient communication mechanism for group-oriented applications such as video conferencing, interactive group games and video on demand. IP multicast [10] saves bandwidth by sending the source traffic on a multicast tree that spans all the members of the group. In this communication model, groups are identified by a group address and any node of the network can join or leave the group freely (using the Internet Group Management Protocol (IGMP)[14]). This simplicity which makes the strength of multicast routing, presents however, many vulnerabilities [4]:

1. IP multicast does not support closed groups. In fact, multicast addresses are publicly known: joining or leaving a group does not require specific permissions [14]. Hence, any user can join a multicast group and receive messages sent to the group.
2. There is no access control to the group: an intruder can send data to the group without being a valid member, and disturbs the multicast session or eventually create bottlenecks in the network.
3. Data sent to the group may transit via many unsecure channels. Thus, eavesdropping opportunities are more important.

Assuring a certain level of security is a requirement for a large deployment of the multicast communication model: some applications need communication confidentiality (such as a pay per view application), other applications need source authentication (such as broadcasting stock quotes), etc.[5, 15, 18] In order to secure group communications, security mechanisms such as authentication, access control, integrity verification and confidentiality are required. Most of these mechanisms rely generally on encryption using one or several keys. The management of these keys, which includes creating, distributing and updating the keys, constitutes then a basic block to build secure group communication applications. In this paper, we focus on group key management [25] used in assuring group communication confidentiality. Group communication confidentiality requires that only valid users could decrypt the multicast data even if the data is broadcast to the entire network. We assume in what follows that data is encrypted to ensure confidentiality using a symmetric cryptosystem (such as DES [22] or AES [24]). In this case, a symmetric key is used to encrypt data by the source and to decrypt it by receivers. This key is generally called Traffic Encryption Key (TEK). The confidentiality requirements can be translated into four key distribution rules [29]:

- Non-group confidentiality : users that were never part
of the group should not have access to any key that can decrypt any multicast data sent to the group.

- Forward confidentiality: users which left the group should not have access to any future key. This ensures that a member cannot decrypt data after it leaves the group.
- Backward confidentiality: a new user that joins the session should not have access to any old key. This ensures that a member cannot decrypt data sent before it joins the group.
- Collusion freedom: any set of fraudulent users should not be able to deduce the current used key.

In order to meet the above requirements, a rekey process should be triggered after each join/leave to/from the secure group. It consists in generating a new TEK and distributing it to the members including the new one in case of a join or to the residual members in case of a leave. This process ensures that a new member cannot decrypt eventually stored multicast data (before its joining) and prevents a leaving member from eavesdropping future multicast data. A critical problem with any rekey technique is scalability: as the rekey process should be triggered after each membership change, the number of TEK update messages may be important in case of frequent join and leave operations, and induces what is commonly called: 1 affects n phenomenon [20]. Some solutions propose to organize the secure group into subgroups with different local traffic encryption keys. This reduces the impact of the key updating process (1 affects n), but needs decryption and reencryption operations at the border of subgroups. These operations may decrease the communication quality.

In this paper we propose a new Scalable and Adaptive Key Management Approach (SAKM). We tackle the scalability issue by subdividing the multicast group into clusters, where each cluster uses its own TEK. In contrast to existing solutions that use the same scheme, with SAKM: the organization of the group into clusters is updated periodically depending on the dynamism of the members during the session. Simulation results show that SAKM scales well to large groups by minimizing the 1 affects n phenomenon, while it reduces the decryption / re-encryption operations thanks to the adaptive dimensioning of the clusters depending on the membership dynamism.

The remaining of this paper is organized as follows: in section 2, we classify and discuss recent group key management protocols proposed in the literature and we motivate our work. In section 3 we give an overview of the proposed architecture, then we present the analytic model on which relies the SAKM protocol in section 4. In section 5 we give a formalization of the SAKM solution. Finally we present details of the SAKM protocol in section 6 and the simulation results in section 7.

2. RELATED WORK AND MOTIVATION

Key management protocols for secure multicast could be classified into two approaches: the common TEK approach and the TEK per subgroup approach. In the following we present each approach and we discuss its strengths and weaknesses.

2.1 Common TEK Approach

In this approach, all group members share a common TEK. The management of this single key could be centralized at a unique key server or distributed among different entities.

2.1.1 Centralized protocols

In this category, a single key server is responsible for computing and distributing a shared key to all group members. This key is used for encrypting and decrypting multicast messages. A naive solution N root/leaf pair-wise keys is presented in [31]. The group controller (GC) attributes a separate secret key to each group member. This secret key is used to establish a unicast secure channel between the GC and each member. When a member leaves the group, the GC generates a new TEK and sends it to each residual member via the secure channel. The number of transmitted messages required for rekeying is then (n – 1) where n is the number of group members. Another similar protocol with almost the same performances is presented in [16][17]. It is clear that this solution does not scale to large multicast groups. In order to reduce the number of required messages for rekeying, authors of [32] and [31] propose to use a key hierarchy solution, let us consider a multicast group with six members \{u_1, u_2, u_3, u_4, u_5, u_6\}. The key server builds an hierarchy of keys as shown in figure 1. Each member owns a secret key which is a leaf in the tree as well as all the keys on its path to the root. The root represents the TEK shared by the members. The other keys are used to reduce the required rekeying messages. According to figure 1: u_1 owns \{k_{1}, k_{12}, k_{1234}, \text{TEK}\}, u_2 owns \{k_{2}, k_{12}, k_{1234}, \text{TEK}\}, u_3 owns \{k_{3}, k_{34}, k_{1234}, \text{TEK}\}, u_4 owns \{k_{4}, k_{34}, k_{1234}, \text{TEK}\}, u_5 owns \{k_{5}, k_{56}, \text{TEK}\} and u_6 owns \{k_{6}, k_{56}, \text{TEK}\}.

Figure 1: key Hierarchy

Let us assume that u_6 leaves the group, GC updates k_{56} into k_{56}' and sends k_{56}' to u_6 encrypted with k_{g}. TEK is updated into TEK' and sent to \{u_1, u_2, u_3, u_4\} encrypted with k_{1234} and to u_6 encrypted with k_{56}' and hence only three messages are required instead of five messages for the previous solution. In [3], Balenson et al. proposed a similar approach, called One way Function Tree (OFT) which reduces the required...
rekeys messages to the half. In fact, with OFT, each internal node key is a function of its children and hence it is computed instead of transmitted. Woldvogel et al.[30] propose to replace the key hierarchy by a flat table in order to decrease the number of keys held by the group controller.

2.1.2 Distributed protocols
In this category, several entities are involved in calculating and distributing the shared key for all group members. In [6] authors proposed the Balas protocol which defines three types of entities: (i) The group controller (GC) which maintains a participant list (PL) and announces the secure session and distributes the KEK to the members via local controllers. (ii) The local controller (LC) (generally a local controller per local network) manages keys within its subnetwork. The GC sends the PL to LCs so that they could decide to accept or refuse membership requests. (iii) Group members are users whose identity is mentioned in the PL.

When a member leaves the group, the corresponding LC generates a new TEK and distributes it to its residual local members authenticated with their secret keys. Then it multicasts the new TEK to the other LCs and members encrypted with the old TEK. This distribution allows all members to share the same new traffic encryption key. This solution raises the problem of conflicts between simultaneous TEK updates by different LCs. To resolve this problem, the GC attributes a priority to each LC. In case of simultaneous TEK updates, members switch to the new TEK generated by the LC with the highest priority. In [21] authors proposed a similar distributed solution called DiRK (Distributed Registration and Key distribution). These protocols still suffer from the 1 affects n phenomenon and bring new challenges such as synchronization and conflict resolution. Setia et al. proposed the Kronos approach [27] where the multicast group is subdivided into many areas and each area is managed by an Area Key Distributor (AKD). The overall AKDs are synchronized using a specific protocol (NTP for example). Rather than rekeying the areas after each membership change, the AKDs make a batch rekeying periodically. This means that membership changes are batched during a period of time and a single rekey is made to take into consideration those membership changes. The synchronization between AKDs is required so that they commit to the same TEK after each period of time.

The single TEK approach suffers from the 1 affects n phenomenon, where a single group membership change (join or leave) results in a rekeying process that disturbs all group members to update the TEK. Moreover, centralized protocols are not scalable, and distributed ones bring new problems such as synchronization and conflict resolution. The Kronos approach attempts to reduce the 1 affects n phenomenon by batch rekeying, but then cannot be used with critical applications that requires to take into consideration the membership change instantly.

2.2 A TEK per subgroup approach
In order to cope with the 1 affects n problem, this approach proposes to organize the multicast group into multiple subgroups. Within each subgroup, a local controller manages a local traffic encryption key. Thereby, any modification in the membership does not affect all the members but only the members of the corresponding subgroup. In Iolus [20], the author proposes to connect several independent multicast groups (with different multicast addresses and eventually with different multicast routing protocols) into an hierarchy of subgroups which forms a virtual multicast group. Each subgroup is managed by a group security agent (GSA) which is responsible for rekeying its own subgroup. Each GSA is a member of its subgroup as well as its parent subgroup. A group security controller (GSC) is responsible for managing GSAs and announcing the secure session. When a member leaves the group, its corresponding GSA rekeys the residual members in the subgroup without affecting other subgroups. This way, Iolus reduces the 1 affects n problem. However, reducing 1 affects n phenomenon is not for free: the multicast messages should be decrypted and reencrypted by the GSAs whenever they pass from a subgroup to another. In [11][12][13], authors proposed a similar protocol called DEP (Dual Encryption Protocol). Unlike Iolus, DEP protocol considers the case where intermediaries (GSAs in Iolus) are not trusted entities and should then be prevented from having access to the multicast data. KHIP [28] protocol presents a similar solution with an elaborated trustiness model and presents an important improvement by restricting decryption/reencryption process to the headers of the IP multicast packets. Hao-hua Chu et al.[8] propose to use a secret encryption key per member that is shared with a group leader. A message is sent along with a random key k. k is encrypted with the secret key of the source. Upon receiving the message, receivers can decrypt it since they do not know the source secret key. When the leader receives the message, it decrypts k using the key that it shares with the source and constructs a validation message which consists of as many slots as there are receivers. Each slot contains k reencrypted with a secret key of a receiver. Upon receiving the validation message, each receiver decrypts k using its secret key and hence decrypts m which was encrypted using k. This protocol is a prominent solution for applications that do not need real-time data transmission, since it discards at all the 1 affects n phenomenon. Yang et al. proposed a reliable rekeying approach in [33]. In the proposed architecture, the multicast group is composed of many subgroups, where each subgroup is managed by a Key Server (KS). The role of a KS is to rekey the members in its subgroup periodically. In other words, the membership changes that occur during a specific period of time are batched, then the KS makes a single rekey that takes into consideration those membership changes. The overall KSs share a common secret key. When a KS receives a multicast message (sent by one of its subgroup members), it decrypts it and re-encrypts it using this secret key. Then, it multicasts the so re-encrypted message to the other KSs. In turn, these KSs decrypt the message using the secret key and re-encrypt it using their respective local TEKs. Then, each KS multicasts the so re-encrypted message to its subgroup.

In conclusion, the TEK per subgroup approach reduces the 1 affects n problem, which is benefic for highly dynamic multicast groups. However, in the case of static multicast groups, this approach becomes so expensive because of the multiple decryption/reencryption operations. Thereby, the efficiency of this approach depends on the dynamism of the group. Besides, some applications do not tolerate latencies, in transmitting data, due to decryption / reencryption op-
2.3 Motivation
Following the above description, we notice that both approaches suffer from great concerns depending on group dynamism: The common TEK approach suffers from 1 affects n phenomenon especially if the group is highly dynamic, while the TEK per subgroup approach suffers from the high decryption/reencryption overhead, especially if the group is quite static. In this case, we present the Scalable and Adaptive Group Key Management approach (SAKM) that takes advantage of the both approaches by dynamically adapting the key management process with respect to the frequency of membership changes. Within the same secure multicast session, SAKM begins with a common TEK approach behavior and dynamically partitions the subgroups into clusters with different local TEKs. The partitioning aims to minimize both decryption/re-encryption and rekeying overheads according to the membership behavior. To illustrate SAKM advantages, let us consider a secure group application (video streaming) that spans a large area (many domains with different countries in different continents). That is, the group members are located in different time beams. This difference in time implies a difference in the membership behavior of the members: the day activity varies from morning to night, a week day activity is more important than a week-end activity, etc...[1]. In this case, it would be interesting to use a protocol that restricts the rekey to the areas with frequent membership changes. Thus, SAKM is very efficient in such situations.

3. OVERVIEW OF SAKM ARCHITECTURE
From the discussion in the previous section, we conclude that current group key management proposals do not scale well with large and dynamic groups, either because of the 1 affects n problem or because of the excess of decryption/reencryption operations problem. Our approach aims to address these two problems by taking into consideration the dynamism of the group members. Figure 2 illustrates the main features and elements of a SAKM architecture: the multicast group is organized into multiple subgroups arranged into a tree structure. Each subgroup is managed by a SAKM Agent which is responsible for the key management process. An SAKM Agent could be in two possible states: active or passive. An active SAKM Agent uses an independent TEK for its subgroup and thus it has to decrypt and reencrypt received messages before forwarding them to local members. A passive SAKM Agent uses the same TEK as its parent subgroup and hence forwards received messages to local members without decryption/reencryption. So, the whole SAKM Agents’ states induces a partition of the subgroups into a set of clusters. Each cluster is composed of a set of subgroups that share the same TEK. The cluster’s root agent is active and all internal agents are passive. Messages are decrypted and reencrypted only at the clusters’ roots.

Periodically, SAKM Agents exchange dynamism information about their subgroups. Based on these information, each agent estimates two costs: the first cost is the overhead cost induced if the agent becomes active (decryption/reencryption cost) and the second cost is the overhead cost induced if the agent becomes passive (the 1 affects n overhead). By comparing the two costs, the SAKM Agent decides whether to become active or passive (i.e. if the first cost is lower then the agent becomes active, else it becomes passive). If an agent becomes passive it merges with its parent cluster and so it uses its parent TEK. If an agent becomes active it forms a new separate cluster and so uses an independent local TEK. After each periodic information exchange about subgroups’ dynamism, we may obtain a new partition of the group into clusters due to split and/or merge operations. This new partition suites better the current membership behavior of the group in terms of both decryption/reencryption cost and 1 affects n overhead. Hence, SAKM approach offers an efficient and adaptive scheme that maintains good performance during the whole secure multicast session. In the following sections, we give detailed description of the SAKM approach by presenting an analytic model and a set of theoretical results on which rely the heuristic and protocols presented in section 6.

4. SAKM ANALYTIC MODEL
In this section we present an analytic model of an elementary system composed of two SAKM subgroups i and j; j is the parent subgroup of i. Each subgroup is managed by an SAKM Agent. By presenting the model, we aim to find out the criterion which should be used by an agent to decide whether it has to be in active or passive state. The two situations: active agent (or split subgroups) and passive agent (or merged subgroups) induce different overheads regarding decryption/reencryption operations and rekeying messages. By comparing the overheads in the two situations, the agent makes the best decision. We use these results as building blocks to propose the Scalable and Adaptive Key Management Protocol for a secure group communication in section 6.

4.1 Preliminaries and nomenclature
In what follows, we will quantify the overhead induced by decryption/reencryption and rekeying in two cases:

- case 1: in the case where the two subgroups i and j are merged to use the same keying material. We denote the induced overhead in this case by \(O_{i}^{m} \) ;
- case 2: in the case where the two subgroups i and j are split into two different clusters, and hence each of them uses its own keying material. We denote the induced overhead in this case by \(O_{i,j}^{s} \).

![Figure 2: SAKM Architecture](image_url)
Thus, an SAKM Agent compares the two quantities $O_{i,j}^{(m)}$ and $O_{i,j}^{(s)}$. If $O_{i,j}^{(m)} > O_{i,j}^{(s)}$ then SAKM agent becomes active (i.e. it is more efficient to separate the subgroup from its parent subgroup so that each of them uses its own keying material: a split operation), else it becomes passive (i.e. it is more efficient to merge the subgroup with its parent subgroup so that they use the same keying material: a merge operation). In this section, we calculate the two overheads: $O_{i,j}^{(m)}$ and $O_{i,j}^{(s)}$.

Let $C_{i,j}^{(m)}$ be the average cost of rekeying in the case $i$ and $j$ are merged (case 1). And $C_{i,j}^{(s)}$ the average cost of rekeying in the case $i$ and $j$ are split (case 2). Suppose that the decryption / reencryption overhead (per time unit) depends only on the encryption system ($\tau$), the computation power of the agent that does the operation ($P_i$) and the rate of messages ($r$) that characterizes the application. The whole decryption/reencryption overhead is given by $\tau(P_i, r)$. Table 1 gives the nomenclature used to present the analytic model.

\[
\begin{align*}
&O_{i,j}^{(m)} \quad \text{The overhead incurred by merging subgroups } i \text{ and } j \\
&O_{i,j}^{(s)} \quad \text{The overhead incurred by splitting subgroups } i \text{ and } j \\
&r \quad \text{Application rate} \\
&P_i \quad \text{Computational power of SAKM agent of subgroup } i \\
&\tau(P_i, r) \quad \text{Decryption / reencryption overhead (per time unit)} \\
&C_{i,j}^{(m)} \quad \text{Cost of rekeying in case subgroups } i \text{ and } j \text{ are merged} \\
&C_{i,j}^{(s)} \quad \text{Cost of rekeying in case subgroups } i \text{ and } j \text{ are split} \\
&E(C_{\lambda, \mu}) \quad \text{Expected number of rekey messages per unit of time in a system where members arrive at a rate } \lambda \text{ and stay in the group for an average of } \mu \text{ time units} \\
&E[J_k] \quad \text{Expected number of rekey messages per unit of time due to a join in a system that contains } k \text{ concurrent members} \\
&E[L_k] \quad \text{Expected number of rekey messages per unit of time due to a leave in a system that contains } k \text{ concurrent members} \\
&M_{(i,j)} \quad \text{Mutual impact rekeying cost due to merging subgroups } i \text{ and } j
\end{align*}
\]

Table 1: Nomenclature

The overhead incurred by split subgroups is the sum of both rekeying and decryption/reencryption overheads, thus

\[
O_{i,j}^{(s)} = C_{i,j}^{(s)} + \tau(P_i, r)
\]

(1)

Where (a) is the factor characterizing the weight given to a decryption/reencryption operation compared to a rekey message. This parameter will be further discussed in the following paragraphs.

The overhead induced by merging subgroups corresponds to the rekeying overhead alone since there is no decryption/reencryption in the case of merged subgroups, and thus

\[
O_{i,j}^{(m)} = C_{i,j}^{(m)}
\]

(2)

In order to approximate $C_{i,j}^{(m)}$ and $C_{i,j}^{(s)}$, we present first the following analytic model.

4.2 Analytic model

We consider that users arrive in a multicast subgroup according to a Poisson process with rate $\lambda$ (arrivals/time unit), and that the membership duration of a member in the group follows an exponential distribution with a mean duration $1/\mu$ time units [1][2]. The average number of concurrent users in a subgroup is given by $\frac{1}{\mu}$. Unlike [7], we do not suppose that the subgroups are likely to be joined by the members. Instead, each subgroup is characterized by its own parameters $\lambda$ and $\mu$. Moreover, we suppose that the parameters $\lambda$ and $\mu$ change over time and thus each SAKM agent adjusts its estimations of $\lambda$ and $\mu$ every $\theta$ time units in order to approximate better the rekeying overhead. Each subgroup can, therefore, be modeled by a Markov process [7]. Let $Q = \{0, 1, 2, 3, \ldots\}$ denote the state system corresponding to the number of concurrent users in a subgroup. Let $\pi_k$ be the steady state probability that $Q = k$. It is well known [19] that

\[
\pi_k = \frac{(\frac{1}{\mu})^k}{k!} e^{-\frac{1}{\mu}}
\]

(3)

Let $J_k$ and $L_k$ be the costs for a user joining and leaving the subgroup in state $k$, respectively. Note that $J_k$ and $L_k$ are random variables depending on where the user joins or leaves the subgroup. Let $E[J_k]$ and $E[L_k]$ be the expected values of $J_k$ and $L_k$, respectively. We denote by $E[C_{\lambda, \mu}]$ the expected number of rekey messages per time unit. By the steady state properties of the Markov chain, $E[C_{\lambda, \mu}]$ can then be expressed by

\[
E[C_{\lambda, \mu}] = \lambda \sum_{k=0}^{\infty} \pi_k (E[J_k] + E[L_k])
\]

(4)

In order to calculate $E[C_{\lambda, \mu}]$, we need to simplify the expression of $\pi_k$ by approximation. Authors in [7] showed that approximating $\pi_k$ as a $\delta$-function at its mean is a good approximation, i.e.,

\[
\pi_k \approx \delta \left( k - \frac{1}{\mu} \right)
\]

(5)

where

\[
\delta \left( k - \frac{1}{\mu} \right) = \begin{cases} 1 & \text{if } k = \frac{1}{\mu} \\ 0 & \text{otherwise} \end{cases}
\]

(6)

Therefore

\[
E[C_{\lambda, \mu}] \approx \lambda \sum_{k=0}^{\infty} \delta \left( k - \frac{1}{\mu} \right) (E[J_k] + E[L_k])
\]

\[
= \lambda (E[J_{\frac{1}{\mu}}] + E[L_{\frac{1}{\mu}}])
\]

(7)

In case of split subgroups, each membership change implies a
rekey process in the subgroup where the membership change occurs and thus we can approximate $C_{i,j}^{(s)}$ by

$$C_{i,j}^{(s)} \approx E[C_{\lambda_i,u_i}] + E[C_{\lambda_j,u_j}]$$  \hspace{1cm} (8)

and hence, from (1) and (8), we obtain

$$O_{i,j}^{(s)} \approx E[C_{\lambda_i,u_i}] + E[C_{\lambda_j,u_j}] + \alpha \tau(P_t,r)$$  \hspace{1cm} (9)

In case of merged subgroups, each membership change implies a rekey process in both subgroups that share the same TEK. We say that a membership change that occurs in subgroup $i$ has an impact on subgroup $j$ (which is merged with $i$), and a membership change that occurs in subgroup $j$ has an impact on subgroup $i$. Therefore, there is a mutual impact rekeying in case of merged subgroups because of sharing a same TEK. Now, we define some terminology to simplify the following discussion:

**Definition 1.** Consider a SAKM-cluster $C$. We mean by mutual impact rekeying in $C$, the sum of required rekey messages sent by SAKM-agents of the SAKM-subgroups $C_{\{i\}}$ when a membership change occurs in subgroup $i$ (for each $i \in C$). We denote by $E[M_{C}]$, the expected number of mutual impact rekeying messages per time unit. In what follows, we call $E[M_{C}]$ the mutual impact rekeying cost of cluster $C$.

In our case, the cluster contains two merged subgroups $i$ and $j$. Thus, according to this definition, we approximate $C_{i,j}^{(m)}$ by

$$C_{i,j}^{(m)} \approx E[C_{\lambda_i,u_i}] + E[C_{\lambda_j,u_j}] + E[M_{i,j}]$$  \hspace{1cm} (10)

and hence, from (2) and (10), we obtain

$$O_{i,j}^{(m)} \approx E[C_{\lambda_i,u_i}] + E[C_{\lambda_j,u_j}] + E[M_{i,j}]$$  \hspace{1cm} (11)

According to (9) and (11), we notice that to compare $O_{i,j}^{(m)}$ and $O_{i,j}^{(s)}$, it is sufficient to compare the two quantities:

$$E[M_{i,j}]$$  \hspace{1cm} (12)

and

$$\alpha \tau(P_t,r)$$  \hspace{1cm} (13)

In conclusion, we notice that $O_{i,j}^{(s)}$ (equation 9) is equal to the expected number of rekey messages per time unit in each subgroup plus the overhead induced by decryption / reencryption operations (see figure 3). $O_{i,j}^{(m)}$ (equation 11) is equal to the number of rekey messages per time unit in each subgroup plus the impact (in term of messages per time unit) of each subgroup on the other subgroup in case of a merge (see figure 4). As the number of rekey messages in each subgroup is common to the two situations, it suffices to compare the differences, i.e. the quantities given by the formulas (12) and (13).

Now we can see that the parameter $\alpha$ plays a key role in the operation and the performance of the proposed scheme. Indeed, the parameter $\alpha$ allows to trade the application level requirement in term of synchronization between the source and receivers, for minimizing the $I$-affects-$N$ phenomenon. In other words:

- If the application requires high synchronization between the source and receivers (such as videoconferencing), then we give to $\alpha$ a large value, so that decryption / re-encryption operations will be minimized by favoring merging over splitting.
- If the application induces high dynamism (such as video on demand), we give to $\alpha$ a small value, so that $I$-
4.3 Application of the analytic model to actual rekey strategies

The rekey overhead depends on the used rekey strategy. As SAKM is an open architecture, each SAKM agent is free to use the most suitable rekey strategy for its subgroup. For clarity reasons, we suppose that all SAKM agents use the same rekey strategy. In order to give illustrative examples we consider mainly two strategies: the "n root/leaf pairwise" [31] protocol and the "hierarchical key graph" [32] protocol. For each of these two strategies we compute the rekeying and the mutual impact rekeying costs in both situations: split and merge of two subgroups. We use these costs to compute \( O_{i,j}^{(m)} \) and \( O_{i,j}^{(n)} \) for each strategy. Table 2 summarizes the results of this section.

4.3.1 Rekeying cost

The n root/leaf pairwise approach: In this approach, all the members share the same TEK. When a member joins the group, the agent multicasts the new TEK (encrypted with the old one) to the old members and unicasts it to the new member encrypted with the secret key that it shares with it. Thus, the cost of a join rekey is two messages

\[
E \left[ J_{\frac{h}{m}} \right] = 2
\]

In case of a leave, the old TEK is compromised and hence the agent sends the new TEK to each remaining member encrypted with the old key that it shares with it, and hence the number of messages in a leave rekey is equal to the size of the group. Thus,

\[
E \left[ L_{\frac{h}{m}} \right] = \frac{\lambda}{\mu}
\]

The hierarchical key graph approach: If we approximate the key graph as a full tree at any time (with degree \( d \)) in case of a join, the agent should redistribute each key that is on the path from the leaf that represents the new member to the root which represents the TEK. Thus, each key from the leaf of the new member to the root (there are \( \log_{\frac{d}{\mu}} \) nodes) is sent twice: it is sent to the new member by unicast encrypted with the child key that is known by the new member, and sent by multicast to the members that share the node encrypted with its old version. Therefore, we obtain

\[
E \left[ J_{\frac{h}{m}} \right] = 2 \log_{d} \frac{\lambda}{\mu}
\]

In case of a leave, the keys on the path from the leaf that represents the leaving member to the root are compromised and hence each of them should be updated. Each key from the leaf (of the leaving member) to the root (there are \( \log_{\frac{d}{\mu}} \) nodes) is sent by multicast (encrypted with its \( d \) child keys) to the residual members that share its \( d \) child keys. Thus

\[
E \left[ L_{\frac{h}{m}} \right] = d \log_{d} \frac{\lambda}{\mu}
\]

4.3.2 Mutual impact rekeying cost

To illustrate the computation of the mutual impact rekeying cost, we consider the two subgroups \( i \) and \( j \) and we compute the mutual impact rekeying cost in the case when these subgroups are merged into the same cluster. In what follows, \( l \) designate either \( i \) or \( j \), and \( l \) designates the counterpart of \( l \) (i.e. if \( l = i \) then \( l = j \) and vice versa). The member arrival rate at subgroup \( l \) is \( \lambda_l \). Let \( IJ_{l}^{i} \) and \( IL_{l}^{i} \) be the impact costs at subgroup \( l \), in state \( k \) when a member joins or leaves subgroup \( l \), respectively. By the steady state properties of the Markov Chain,

\[
E \left[ M_{(i,j)} \right] = \lambda_l \sum_{k=0}^{\infty} \pi_k (E \left[ IL_{k}^{i} \right] + E \left[ IL_{k}^{l} \right]) + \lambda_l \sum_{k=0}^{\infty} \pi_k (E \left[ IL_{k}^{i} \right] + E \left[ IL_{k}^{l} \right])
\]

When a member joins subgroup \( l \), SAKM-agent of subgroup \( l \) sends the received new TEK by multicast (one message) to its members encrypted with the old TEK which is not compromised. Hence, \( E \left[ IJ_{l}^{i} \right] = 1 \). Thus, equation 18 becomes

\[
E \left[ M_{(i,j)} \right] = \lambda_l \sum_{k=0}^{\infty} \pi_k (1 + E \left[ IL_{k}^{i} \right]) + \lambda_l \sum_{k=0}^{\infty} \pi_k (1 + E \left[ IL_{k}^{i} \right]) + \lambda_l \sum_{k=0}^{\infty} \pi_k (1 + E \left[ IL_{k}^{i} \right])
\]

By considering the approximation of \( \pi_k \approx \delta \left(k - \frac{\lambda}{\mu} t\right) \) [7], equation 19 becomes

\[
E \left[ M_{(i,j)} \right] = \lambda_l \left(1 + E \left[ IL_{1}^{i} \right]\right) + \lambda_l \left(1 + E \left[ IL_{1}^{i} \right]\right)
\]

The computation of \( E \left[ IL_{1}^{i} \right] \) depends on the actual rekey strategy:

The n root/leaf pairwise approach: In case of a leave at subgroup \( l \), the old TEK shared between subgroups \( i \) and \( j \) is compromised and hence the SAKM-agent of subgroup \( l \) sends the new TEK encrypted with each secret key that it shares with its \( \frac{d}{\mu} \) members. Therefore, with this rekey strategy

\[
E \left[ IL_{1}^{i} \right] = \frac{\lambda_l}{\mu}
\]

The hierarchical key graph approach: We approximate the key tree as a full tree at any time (with degree \( d \)). In case of a leave at subgroup \( l \), the old TEK is compromised and hence the agent sends the new TEK by multicast and encrypted with each root’s child key (there are \( d \) keys: the degree of the tree) that are shared by the members of subgroup \( l \). Therefore, with this rekey strategy

\[
E \left[ IL_{1}^{i} \right] = d
\]
5. SAKM Problems Statement

In the above sections, we discussed the overhead induced by merging two subgroups. In a general case, we deal with merging many SAKM subgroups whenever splitting them is more expensive than merging them together to use the same keying material. According to definition 1 of mutual impact rekeying, we have $M_C = \sum_{i,j \in C} M_{i,j}$, where $C$ is a cluster of merged SAKM-subgroups, and the expected number of mutual impact rekeying messages per time unit can be expressed by

$$E[M_C] = \sum_{i,j \in C} E[M_{i,j}]$$

5.1 Illustrative example

Let $G = \{x, y, z\}$, where $x$, $y$ and $z$ are SAKM subgroups and form a virtual hierarchy (see figure 5.2). To this tree, we can associate four partitions $\{x, z\}$, $\{x, y\}$, $\{x, y, z\}$ and $\{y, z\}$ (see figure 5.3). The cost associated to the partition $\{x, y, z\}$ is

$$E[M_{x,y}] + \alpha \tau(P_s, r)$$

In the same way, we establish the table 3 which associates a cost to each G-partition.

<table>
<thead>
<tr>
<th>Partition</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>${x, y, z}$</td>
<td>$\alpha \tau(P_s, r)$</td>
</tr>
<tr>
<td>${x, z}$</td>
<td>$E[M_{x,z}] + \alpha \tau(P_s, r)$</td>
</tr>
<tr>
<td>${y, z}$</td>
<td>$E[M_{y,z}] + \alpha \tau(P_s, r)$</td>
</tr>
<tr>
<td>${x, y}$</td>
<td>$E[M_{x,y}] + \alpha \tau(P_s, r)$</td>
</tr>
</tbody>
</table>

Table 3: Costs associated to G-partitions

SAKM clusters $\{G_1, G_2, \ldots, G_i, \ldots\}$ of the given SAKM hierarchy. To each cluster $G_i$, we associate a cost $C(i)$ which is the sum of decryption / reencryption cost at the $G_i$’s root and the mutual impact rekeying cost due to merging SAKM subgroups into the cluster $G_i$.

$$C(i) = E[M_{G_i}] + \alpha \tau(P_s, r)$$

We are interested in finding a sub-family of $E$: $F = \{T_1, T_2, \ldots, T_i, \ldots\}$, so that the corresponding SAKM clusters $\{G_1, G_2, \ldots, G_i, \ldots\}$ form a partition of $G$ with a total minimal cost. In other words, we want to:

$$\min \sum_{G_i \in F} C(i)$$

with $F \subseteq E$

and $\forall G_i, G_j \in F, G_i \cap G_j = \emptyset$ if $i \neq j$.

The following lemma gives the size of the SAKM problem.

Lemma 1. If $|G| = p$ then the number of G-partitions is $2^{p-1}$.

Proof: Each G-partition cluster is made up of a root SAKM subgroup whose SAKM agent is active and child subtrees of SAKM subgroups whose SAKM agents are passive. Thus, each G-partition defines a combination of active / passive SAKM agents. And, vice versa, each combination of active / passive SAKM agents (except the root which should be always active) defines a G-partition. It follows that the number of possible combinations (of $p - 1$ SAKM agents which can hold either passive or active state) is $2^{p-1}$, which corresponds to the number of possible G-partitions.

6. SAKM Protocol

In order to find out the optimal G-partition, a naive solution would be to enumerate the $2^{p-1}$ possible configurations and to pick up the one with the smallest cost. This would incur a $O(2^{p-1})$ overhead which is not acceptable. In our case, we propose a heuristic to approach the optimal configuration with reasonable delays and overheads. The heuristic used by SAKM to partition the virtual hierarchy into clusters with independent keying material relies on the following results.

Lemma 2. Let $x$ and $y$ be SAKM subgroups, with $x$ parent of $y$. If $\alpha \tau(P_s, r) \leq E[M_{x,y}]$, then $\alpha \tau(P_s, r) + E[M_{x,z}] + E[M_{y,z}] < E[M_{x,y}] + E[M_{x,z}] + E[M_{y,z}]$ for any set of SAKM subgroups that can be merged with $x$ and $y$, any set of SAKM subgroups that can be merged with $y$ (see figure 6).
Proof This lemma states that if the decryption/ reencryption cost at an SAKM agent $y$ is smaller than the mutual impact rekeying cost induced by merging $y$ with its parent subgroup $x$, then the overhead induced by splitting $x$ and $y$ is less important than the overhead induced by merging many subgroups that include $x$ and $y$. In fact, we have $E[M_{(x,y)}] = E[M_{(x)}] + E[M_{(y)}] + \sum_{j \in [y]} E[M_{(i,j)}]$. It follows then, that if $\alpha(P_y, r) \leq E[M_{(x,y)}]$, then

$$
\alpha(P_y, r) + E[M_{(x)}] + E[M_{(y)}] \\
\leq E[M_{(x,y)}] + E[M_{(x)}] + E[M_{(y)}] + \sum_{j \in [y]} E[M_{(i,j)}] \\
+ E[M_{(x)}] + E[M_{(y)}] + \sum_{j \in [y]} E[M_{(i,j)}] \\
= E[M_{(x)}] + E[M_{(y)}] + \sum_{j \in [y]} E[M_{(i,j)}] \\
= E[M_{(x,y)}] + E[M_{(y)}].
$$

Corollary 1. Let $x$ and $y$ be SAKM subgroups with $x$ parent of $y$. If $\alpha(P_y, r) \leq E[M_{(x,y)}]$, then $y$ would be a cluster's root in the optimal SAKM-partition.

Proof Suppose that $\alpha(P_y, r) \leq E[M_{(x,y)}]$. If $y$ is not a cluster's root in the optimal partition, then it should be an internal node to a cluster in the optimal partition. Let $\{x, y\} \cup Z_x \cup Z_y$ be that cluster (see figure 6). $x$ is parent of $y$. $Z_x$ and $Z_y$ are SAKM subgroups' sets. The cost associated to this cluster in the optimal partition is: $E[M_{(x,y)}] + \alpha(P_t, r)$ where $t$ is the $Z_x$'s root. Now, we show that if $\alpha(P_y, r) \leq E[M_{(x,y)}]$, then there exists a partition with a smaller overhead than the one that contains the cluster $\{x, y\} \cup Z_x \cup Z_y$.

In fact, if we split this cluster into two clusters by making $y$'s SAKM agent active, we would have two clusters with a total cost of $\alpha(P_y, r) + E[M_{(y)}] + \alpha(P_t, r) + E[M_{(x)}]$. According to lemma 2, we have

$$
\alpha(P_y, r) + E[M_{(y)}] + E[M_{(x)}] < E[M_{(x,y)}] + \alpha(P_t, r)
$$

which means that we found a partition with a cost smaller than the one pretended to be optimal.

Corollary 2. Let $x$ and $y$ be SAKM subgroups with $x$ parent of $y$. If $\alpha(P_y, r) \leq E[M_{(x,y)}]$ then the $y$'s SAKM agent can decide to become active and this would be an optimal decision.

Proof If $\alpha(P_y, r) \leq E[M_{(x,y)}]$ then it follows from corollary 1 that $y$ must be a cluster's root in the optimal partition, which is equivalent to say that $y$ must be active (assures decryption/ reencryption in the optimal configuration).

6.1 Overview of SAKM protocol

We remind that the SAKM architecture is made up of an hierarchy of multicast subgroups. The whole hierarchy forms an SAKM multicast group. The keying material inside a subgroup is managed by an SAKM agent. Two adjacent subgroups may merge to use the same parent's keying material if the mutual impact rekeying cost is less than the decryption/ reencryption cost at the child SAKM agent. In the contrary case, the two adjacent subgroups are split and each of them uses its own keying material. The SAKM aim is partitioning the hierarchy into clusters so that both decryption/ reencryption and rekeying overheads are minimized. Each cluster is a set of SAKM subgroups using the same keying material. SAKM uses a heuristic to approach the optimal solution. Corollary 2 states that if the decryption/ reencryption cost at a child SAKM agent $y$ is less important than the mutual impact rekeying cost induced by
its merge with its parent subgroup, then \( y \) can decide to become active and that this decision is optimal. Relying on this corollary, SAKM proceeds as follows: periodically, each SAKM agent \( x \) computes new estimations of the two parameters \( \lambda_x \) (the mean arrival rate of members at the agent’s subgroup) and \( \mu_x \) (where \( \frac{\lambda_x}{\mu_x} \) is the mean membership duration of the members of the agent’s subgroup). \( x \) sends these parameters \( (\lambda_x, \mu_x) \) to its child SAKM agents \( y \). Each child SAKM agent \( y \) compares then the two costs: \( \sigma_x(P_{y,r}) \) and \( E[M_{x,y}] \). If \( \sigma_x(P_{y,r}) \leq E[M_{x,y}] \), then \( y \) becomes active (corollary 2). If \( \sigma_x(P_{y,r}) > E[M_{x,y}] \), then \( y \) becomes passive. The decision to become passive may be not optimal. In fact, each child decides to become passive without taking into consideration the mutual impact rekeying cost due to the other children that might merge with their parent subgroup. However, if \( x \)'s children \( (y) \) verify the inequation \( \sigma_x(P_{y,r}) > E[M_{x,y}] \), it means that the membership of each of them is so stable (upper bounded by roughly the same constant: \( \sigma_x(P_{y,r}) \)) that we can merge them into the same cluster to use the same \( x \)'s keying material even if this may not be the best configuration (see simulation results in section 7).

Each SAKM agent \( y \) holds two Traffic Encryption Keys (TEKs): \( TEK_y \) used in its own subgroup and \( TEK_{xy} \) used in its parent subgroup \( x \). Note that if \( y \) is passive then \( TEK_y = TEK_x \). If \( y \) is active then, it decrypts received messages using \( TEK_x \) and reencrypts them toward its own subgroup using \( TEK_y \).

Five types of messages are involved in the protocol:

- **NEW-TEK_RQ:** this type of message is sent by a passive agent when a membership change occurs in its subgroup. This message specifies the membership change type (join or leave) and it is sent to the cluster’s root agent which is responsible for delivering TEKs for the cluster.

- **IM-ACTIVE:** this type of message is sent by an agent when it becomes active. The message is sent to the cluster’s root agent. Upon receiving the message, this later distributes a new TEK to be used by the remaining subgroups in the cluster.

- **NEW-TEK:** this message type is used by an active agent to distribute a new TEK to its cluster. It specifies the new TEK and the agent’s identity which is necessary for the passive agents to request new TEKs whenever membership changes occur in their subgroups. It specifies also the membership change type which caused delivering the new TEK, because the distribution scheme of the new TEK depends on the membership change type (join or leave). The keys used to encrypt the new TEK depend also on the membership change type (see subsection 4.3).

- **JOIN_LEAVE:** these messages are sent by the members to join or leave the virtual multicast group. An expellee of a member by an agent is considered as receiving a LEAVE message. This type of message specifies the membership change type along with the required member authentication information.

\[
(\text{NEW-TEK}_RQ) = (m) \quad \text{NEW-TEK} = (m)_{k-1} \quad \text{JOIN_LEAVE} = (m)_{k-1}
\]

\[
(\text{NEW-TEK}_RQ) \quad \text{NEW-TEK} \quad \text{JOIN_LEAVE}
\]

\[
\begin{align*}
(m)_k & \text{ means the message } m \text{ is encrypted with } k \text{ using a symmetric encryption algorithm such as DES [22] or AES [24].} \\
(m)_{k-1} & \text{ means } m \text{ is signed with the private key } k^{-1} \text{ using a signing algorithm such as RSA [26] or DSS [23].} \\
\text{verify}(m) & \text{ is a primitive that verifies whether the message's signature is correct.} \\
\text{authorized-}(request) & \text{ is a primitive that authenticates the joining member and verifies its access rights.} \\
\text{tek}(request) & \text{ is a primitive that extracts the encrypted TEK from the message request.} \\
\text{type}(request) & \text{ is a primitive that returns the type of the requested membership change (join or leave).} \\
\text{rekey}(type) & \text{ is a primitive that assures a rekey process depending on the adopted rekey strategy and the type of membership change (see subsection 4.3).} \\
\text{mutual-} & \text{ mutual-} \\
\text{Impact-} & \text{Impact-} \\
\text{Rekey(type)} & \text{Rekey(type)} \\
\text{Impact} & \text{Impact} \\
\text{NEW_DYN_INF} & \text{NEW_DYN_INF} \\
\text{NEW_DYN_INF} & \text{NEW_DYN_INF} \\
\text{NEW_DYN_INF} & \text{NEW_DYN_INF}
\end{align*}
\]

\[
\begin{align*}
\text{NEW_DYN_INF} & \text{NEW_DYN_INF} \\
\text{NEW_DYN_INF} & \text{NEW_DYN_INF} \\
\text{NEW_DYN_INF} & \text{NEW_DYN_INF} \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>Table 4: Notation and primitives</th>
</tr>
</thead>
</table>

- **NEW_DYN_INF**: this message is sent by SAKM agents to refresh their dynamic parameters estimation. Each agent sends this message periodically to its child agents. It specifies the new estimations of the two parameters \( \lambda \) and \( \mu \). Upon receiving this message, the child agents decide whether to become passive or active.

### 6.2 Notation and primitives

Table 4 shows the notation and primitives that are used to write algorithms in the remaining of the paper.

### 6.3 Merge / Split Protocol

In this phase of SAKM, the hierarchy is partitioned into clusters that use the same keying material. Periodically (let say after each \( \theta \) time units), each SAKM agent \( x \) sends the new estimations of \( \lambda_x \) and \( \mu_x \) to its children. \( x \) signs the message with its private key \( k_x^{-1} \) to ensure the authenticity and the integrity of the sent parameters. Upon receiving \( \lambda_x \) and \( \mu_x \) (the parent’s parameters), each SAKM agent compares \( \sigma_x(P_{y,r}) \) to \( E[M_{x,y}] \) and takes the decision to become active or passive according to the results of that comparison. Figure 7 summarizes this phase.

When \( y \) becomes active, it generates a new \( TEK_y \) and multicasts it to the members of its subgroup (line 1). Note that as \( y \)'s child SAKM agents are members in \( y \)'s subgroup, they receive the new \( TEK_y \). When a passive child SAKM agent receives the new \( TEK_y \), it forwards its distribution to its subgroup and the process continues until all the members in the cluster that will use \( y \)'s TEK receive the new \( TEK_y \). \( y \) informs then the cluster’s root agent (t) about the decision to become active (line 2) using the “IM-ACTIVE” message. Upon receiving the message, \( t \) distributes a new \( TEK_t \) to the remaining subgroups in the cluster. This update of \( TEK_t \) is compulsory to ensure backward and forward secrecy. In the case where \( y \) becomes passive, it multicasts its parent’s \( TEK \) to the members of its subgroup (line 4) and changes its state to passive. This TEK is forwarded by \( y \)'s children to all the members in the \( y \)'s old cluster and informs SAKM agents of the cluster about the new cluster’s root.

### 6.4 Membership change protocol
Let $x$ and $y$ be SAKM agents with $x$ parent of $y$.

Agent $x$

$x \rightarrow y: NEW\_DYN\_INF, \lambda_x, \mu_x, \text{timeStamp} > k-1$ /* periodically */

Agent $y$

If (state=PASSIVE and $\alpha(P_y, r) \leq E[M(x,y)])$

1. $y \rightarrow SUBG\_ADD: < NEW\_TEK, \{\text{newTEK}\}_{\text{oldTEK}}, JOIN, my\_ID, \text{timeStamp} > k-1$;
2. $y \rightarrow CLUSTER\_ROOT\_ADD: < IM\_ACTIVE, \text{timeStamp} > k-1$;
3. state=ACTIVE;
end if

if(state=ACTIVE and $\alpha(P_y, r) > M(x,y))$

4. $y \rightarrow SUBG\_ADD: < NEW\_TEK, \{\text{parentTEK}\}_{\text{oldTEK}}, JOIN, \text{clusterRoot}, \text{timeStamp} > k-1$;
5. state=PASSIVE;
end if

The main idea in this phase is to restrict rekeying to the cluster where occurs the membership change (join or leave). This minimizes the $I$ affects $N$ phenomenon as only the members of the cluster are concerned by the rekey. As the members of a cluster use the cluster’s root TEK, when a membership change occurs in a subgroup, the join / leave information is sent to the cluster’s root which is responsible for generating and distributing a new TEK for the valid members in the cluster. When a membership change occurs in a subgroup, the SAKM agent responsible for that subgroup reacts as follows: If the agent is active (which means that it is a cluster’s root), it generates and distributes a new TEK to its subgroup and hence to its cluster (the distribution is forwarded by its child SAKM agents). If the agent is passive (an internal agent in a cluster), it sends a request to the cluster’s root asking for a new TEK for the cluster ("NEW\_TEK\_RQ"). Then the cluster’s root agent generates and distributes a new TEK for the cluster. All the agents of the cluster distribute the new TEK to their subgroup members according to their rekey strategy and depending on the membership change type (join or leave). In figure 8, the numbers show the order of the steps to be taken in order to rekey a cluster following a membership change.

6.5 Agent’s dynamic behavior

As shown above, the behavior of an SAKM agent depends on its state (active or passive): this is due to the adaptive and dynamic aspect of the protocol. An SAKM agent can take three states:

- active: in the case the agent is a cluster’s root. In this state the agent assures decryption / reencryption operations, and is responsible for generating a new TEK whenever a membership change occurs in its cluster.
- passive: in the case the agent is a cluster’s internal node. In this state, the agent just forwards messages without decryption / reencryption, forwards also up to date TEKs and asks the cluster’s root agent new TEKs whenever membership changes occur in its subgroup.
- "waiting TEK": this state is factual and introduced just to simplify writing the protocol processed by an SAKM agent. An agent is in this state when it is passive and waiting an up to date TEK from the cluster’s root agent.

The state chart of figure 9 depicts the SAKM agent’s behavior (operations and actions triggered by events and received messages according to the state of the agent). The numbers in figure 9 correspond to the numbered code portions and are commented in the following paragraphs. Each procedure deals with a message type according to the agent’s state. We suppose that the procedures are processed by the agent $y$ whose parent is $x$.

6.5.1 Receipt of NEW\_DYN\_INF message

In actions (1)(fig.10), an active agent becomes passive: it multicasts (to its subgroup) its parent’s TEK encrypted with the TEK used in its subgroup (oldTEK). In actions (2)(fig.10), a passive agent becomes active: it generates a new TEK and multicasts it to its subgroup (and hence to the new cluster for which it becomes a root), then it informs the cluster’s root about the change in its state, so that the latter updates the TEK for the remaining subgroups in the cluster. If the agent is waiting for a new TEK because of a membership change in its subgroup and receives a NEW\_DYN\_INF message which states that it should become active ($\alpha(P_y, r) \leq E[M(x,y)])$, then the agent generates a
new TEK and rekey its subgroup according to the adopted strategy and the membership change type. The agent informs then the cluster’s root agent about the change of state (actions (3)-Fig.10).

6.5.2 Receipt of NEW_TEK_RQ message
Action (4) (Fig.11) consists of a mutual impact rekeying which depends on the adopted strategy and the membership change type for which the new TEK is requested (see subsection 4.3)

6.5.3 Receipt of IM_ACTIVE message
When a cluster’s root agent receives an IM_ACTIVE message, it generates a new TEK and multicasts it to its subgroup (and hence to the remaining subgroups in the cluster) encrypted with the old TEK (action (5)-Fig.12)

6.5.4 Receipt of NEW_TEK message
When a membership change occurs in a subgroup whose SAKM agent is passive, the agent asks the cluster’s root agent for a new TEK (action (6)-Fig.13). If the agent is waiting for a new TEK, it then makes a rekey of its subgroup and resumes the passive state (actions (7)-Fig.13). Finally, if the agent is a cluster’s root, it takes note that its parent updated its TEK (action (8)-Fig.13)

6.5.5 Receipt of JOIN_LEAVE message
When a membership change occurs in a subgroup whose SAKM agent is passive, the agent asks the cluster’s root agent for a new TEK (action (9)-Fig.14). However, if the agent is active (a cluster’s root) then it generates and distributes a new TEK according to the adopted rekey strategy and the membership change that occurred in its subgroup (action (10)-Fig.14)

7. SIMULATION RESULTS
In this section, we provide an overview of our simulation model and some of the results we obtained by comparing SAKM with Iolus and a centralized solution. We selected Iolus [20] as a TEK per subgroup approach representative protocol. In this approach, the multicast group is divided into
multiple subgroups, in a static manner, with independent TEKs and thus it suffers from the high number of decryption / reencryption operations. We selected the centralized solution as a representative protocol of the common TEK approach. In this approach, members share a same TEK and thus suffer from the 1 affects n phenomenon. We study the 1 affects n behavior of each simulated protocol and the number of decryption / reencryption operations required for the communication.

### 7.1 Simulation model

In our simulation, we use a virtual SAKM multicast group made up of five multicast groups organized as shown in figure 8. We suppose that the group is composed of 100 dynamic members in the average. In order to show the ability of SAKM to cope with different application requirements, we make the simulation for three types of applications characterized by their requirement in term of synchronization between the source and receivers (see table 5).

<table>
<thead>
<tr>
<th>Application</th>
<th>Synchronization</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Low synchronization required</td>
</tr>
<tr>
<td>T2</td>
<td>High synchronization required (small latencies are allowed)</td>
</tr>
<tr>
<td>T3</td>
<td>Real-time (no latency allowed)</td>
</tr>
</tbody>
</table>

Table 5: Three application types

In this approach, members share a same TEK and thus suffer from the 1 affects n phenomenon. We study the 1 affects n behavior of each simulated protocol and the number of decryption / reencryption operations required for the communication.

#### 7.2 Split / merge criteria

Experiments [9] show that with a Celeron 850 MHz processor, to encrypt or decrypt a message of 1 MBytes we have the results of table 6.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Encryption time</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES</td>
<td>78ms</td>
</tr>
<tr>
<td>AES</td>
<td>33ms</td>
</tr>
<tr>
<td>IDEA</td>
<td>88ms</td>
</tr>
</tbody>
</table>

Table 6: Required time to encrypt a message of 1 MBytes

If we consider a flow with a rate $r$ ($2^{-3}$Mbytes/s or 16kbps of encrypted data), and we use DES to assure its secrecy, then each $r$ Mbytes of received data, we would have an overhead of $2.r.DES_t$ (2 for decryption and reencryption, $DES_t = 78ms$ in our case) seconds for decryption / reencryption. Thus the decryption / reencryption overhead becomes

$$\alpha \tau(P, r) = 2.\alpha.r.DES_t$$

We will show that $\alpha$ plays a key role in controlling the behavior of SAKM regarding synchronization requirements of the application. Consider two adjacent subgroups of an SAKM hierarchy $i$ and $j$ with $i$ parent of $j$. Suppose that both SAKM agents of subgroups $i$ and $j$ use the hierarchical key graph scheme for relaying and that the graph is a full binary tree. Thus the mutual impact relaying cost would be

$$E[M_{i,j}] = 3(\lambda_i + \lambda_j)$$

(see table 2). Hence, for $j$ to take a decision about its state, it has to compare between the two costs $2.\alpha.r.DES_t$ and
\[ E[M_{i,j}] = 3(\lambda_i + \lambda_j) \] after each \( \theta \) units of time (15mins in our simulation).

### 7.3 Simulation results and discussion

In a first stage, we consider a T1 application that do not require much synchronization. Since members arrival at subgroups is not uniform, SAKM aims to minimize decryption / reencryption overhead as well as the \( t \) affects \( n \) phenomenon according to the members' dynamism. Figures 15,16,17 show the results obtained with a T1 application: figure 15 measures the number of decryption / reencryption operations which corresponds to the number of clusters in SAKM and to the number of subgroups in Iolus (5 in our case). Figure 16 measures the number of affected members: with the centralized solution all the members are affected, with Iolus only the members of a subgroup are affected, that is why the results of Iolus are much smaller, SAKM makes a trade off between decryption / reencryption overhead and \( t \) affects \( n \). Figure 17 gives the same results of figure 16 in the average which means that it divides the number of affected members by the number of active clusters. As T1 tolerates latencies, we give to the weight \( \alpha \) a small value (4 in our case), so that SAKM creates as much clusters as it needs to attenuate \( t \) affects \( n \) and to minimize decryption / reencryption overhead compared to Iolus which makes it systematically at each Iolus agent. At a first sight, we remark that even if SAKM makes only three decryption / reencryption operations in the average (see figure 15), it maintains as good performances as those of Iolus (see figures 16,17). SAKM starts with a centralized behavior (a single cluster) \((0 < t < 700\text{seconds})\). As the group size grows, the group dynamism increases and thus SAKM creates new clusters following the split / merge process; at \( t = 700\text{s} \) SAKM creates only 3 clusters (see figure 15) and reaches with that Iolus performances regarding \( t \) affects \( n \) attenuation (see figures 16, 17), which means that SAKM saves decryption / reencryption overhead as well rekeying messages overhead, which is not the case with Iolus. At each time the group dynamism reaches a certain degree, SAKM creates a new cluster and hence the number of decryption / reencryption operations increases by one and in the counter part \( t \) affects \( n \) is much more attenuated (see for example: figure 15 and figure 16 at 3800s < \( t < 5500\text{s} \) and at

**Figure 15: Dec / Re-enc operations (T1)**

Whenever decryption / reencryption cost \( 2.\alpha.r.DES_i \) exceeds the mutual impact rekeying cost, SAKM destroys as much clusters to reach a better whole partition cost (see figure 15 and 16 at \( t = 5500\text{s} \) and \( t = 9000\text{s} \)). In this way, SAKM assures a trade off between decryption / reencryption cost and rekeying cost so that the whole cost is minimized.

In a second stage, we consider a T2 application that requires high synchronization between the source and receivers. Iolus do not fit with this kind of applications since decryption / reencryption operations would introduce latencies that are not desirable. With SAKM, we give to the factor \( \alpha \) a value (8 in our case) that prevents it from creating a lot of clusters except for situations where the membership changes in a sharply way and it would be better to limit the rekey to the subgroup(s) where the membership changes. Such a situation is very rare and do not affect performances of T2 that tolerates some latencies (it is not a serious problem when we receive a slightly slow sequence for a while when seeing a movie on the Internet). In figure 18 we remark that the state of SAKM agents does not change frequently, and that it happens only when the membership changes in a sharp way (see figures 18, 19 at 4500s < \( t < 8000\text{s} \)) where
SAKM creates three clusters because of a sharply change in the membership change.

Finally we consider a T3 application, where it is out of question to do decryption / reencryption of the data which should reach receivers in real-time. Iolus does not support this kind of requirement. With SAKM it suffices to put the factor \(a\) to infinite theoretically (16 in our case) to prevent SAKM from creating clusters. And hence SAKM becomes typically a centralized solution without intermediaries (see figures 20, 21).

8. CONCLUSION

Security mechanisms are an urgent requirement for multicasting in order to ensure a safe and large deployment for confidential group communications. Key management protocols play a key role in the whole secure multicast architecture. In real multicast sessions, members can join and leave the group dynamically during the whole session. This dynamicity affects considerably the performances of the key management protocol. Most proposed solutions in the literature do not take this parameter into consideration and so suffer either from the 1 affects n phenomenon or from the important decryption / reencryption overhead. In this paper, we considered a special class of group key management which subdivides the multicast group into subgroups with independent traffic encryption keys so that the 1 affects n phenomenon and hence the rekeying overhead are minimized. In this kind of architectures, multicast messages should be decrypted and reencrypted at the boundaries of subgroups. The decryption / reencryption operations create a new overhead which could be disastrous for some kind of applications that require a real-time or highly synchronized data transmission. We showed that it is possible to make a trade off between the two overheads (decryption / reencryption and rekeying overheads) by making an adaptive clustering of subgroups. We proposed a heuristic to approach the optimal configuration of this clustering and the simulation results show that it is a good approach compared to two other approaches from the literature.

9. REFERENCES


