Group Key Management Protocols: A Novel Taxonomy

Yacine Challal, Hamida Seba

Open Science Index, Computer and Information Engineering Vol:2, No:10, 2008 waset.org/Publication/11804

***Abstract*—Group key management is an important functional building block for any secure multicast architecture. Thereby, it has been extensively studied in the literature. In this paper we present relevant group key management protocols. Then, we compare them against some pertinent performance criteria.**

***Keywords*—Multicast, Security, Group Key Management.**

I. Introduction

THE phenomenal growth of the Internet in the last few years and the increase of bandwidth in today's networks have provided both inspiration and motivation for the development of new services, combining voice, video and text "over IP". Although unicast communications have been predominant so far, the demand for multicast communications is increasing both from the Internet Service Providers (ISPs) and from content or media providers and distributors. Indeed, *multicasting* is increasingly used as an efficient communication mechanism for group-oriented applications in the Internet such as video conferencing, interactive group games, video on demand (VoD), TV over Internet, e-learning, software updates, database replication and broadcasting stock quotes. Nevertheless, the lack of security in the multicast communication model obstructs the effective and large scale deployment of such strategic business multi-party applications. This limitation motivated a host of research works that have addressed the many issues relating to securing the multicast, such as *confidentiality, authentication, watermarking and access control*. These issues must be seen in the context of the *security policies* that prevail within the given circumstances. For instance, in a *public* stock quotes broadcasting, while *authentication* is a fundamental requirement, *confidentiality* may not be. In the contrary case, both *authentication* and *confidentiality* are required in *video-conference* applications. In this paper, we focus on a keystone component of any secure multicast architecture over wired networks: *group key management*.

II. Group Communication Confidentiality

In this section, we will use a simple scenario to introduce the challenging issues relating to group confidentiality and key management. We consider a *source* that sends data to a set of *receivers* in a multicast session. The security of the session is managed by two main *functional entities*: a *Group Controller (GC)* responsible for authentication, authorization and access control, and a *Key Server (KS)* responsible for the maintenance and distribution of the required *key material*. Note that these two functions can be implemented over a single physical entity or over different physical entities depending on the key management architecture. Figure 1 depicts this simple scenario.

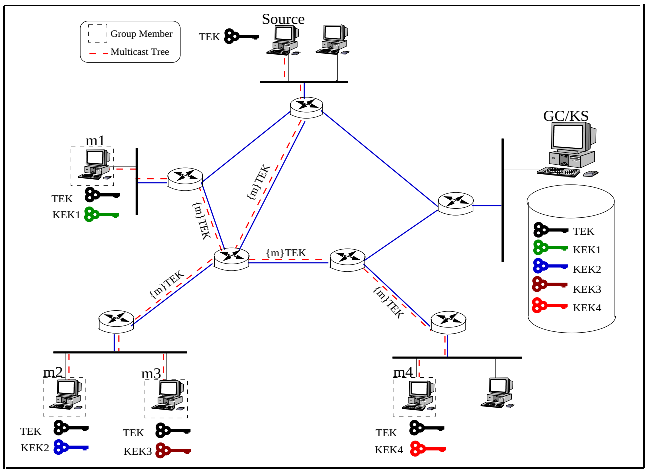


Fig. 1. Simple scenario of group confidentiality components

To ensure confidentiality during the multicast session, the sender (source) shares a secret symmetric key with all valid group members, called *Traffic Encryption Key (TEK)*. To multicast a secret message, the source encrypts the message with the TEK using a symmetric encryption algorithm. Upon receiving the encrypted multicast message $\{m\}_{TEK}$, each valid member that knows the TEK can decrypt it with TEK and recover the original one. To avoid that a leaving or an ejected member from the group, continues to decrypt the secret multicast messages, the KS must generate a new TEK and securely distribute it to the all remaining group members except the leaving one. This operation is called ***re-keying***. The KS shares a secret key called *Key Encryption Key* (KEK_i) with each member m_i (cf. figure 1). To re-key the group following a leave from the group, the KS generates a new TEK: TEK', and sends it to each member m_i (except the leaving one) encrypted with its corresponding KEK_i. Thereby, the leaving member cannot know the new TEK' and hence will not be able to decrypt future multicast messages of this session.

When a new member joins the session, it must be authenticated by the GC. After checking the rights of the new member to access the group, the KS proceeds to a new group *re-keying* to avoid that the new member decrypts previous exchanged messages using the current key. Therefore, the KS generates a new TEK: TEK', encrypts it with the old TEK: $\{TEK'\}_{TEK}$, and multicasts it to

Yacine Challal, and Hamida Seba are with Compiegne University of Technology, Heudiasyc lab., France (phone:33-3-44-23-44-23; e-mail: ychallal@hds.utc.fr, seba@hds.utc.fr).

Open Science Index, Computer and Information Engineering Vol:2, No:10, 2008 waset.org/Publication/11804

the group. Hence all old members can recover the new TEK: TEK'. Then, the KS encrypts TEK' with the secret KEK_j that it shares with the new member m_j and sends it to him to recover TEK' which is required to decrypt the multicast messages.

The maintenance and the distribution of the keys involved in *re-keying* and encryption is commonly called : *Group Key Management*. In this illustrative protocol, re-keying induces a $O(n)$ re-key messages after each *leave* from the group, where n is the number of group members. It induces also a storage of $O(n)$ keys (1 *TEK* + n KEK_i) at the KS during the whole secure multicast session. Since each membership change in the group requires re-keying and the group may be highly dynamic, one of the challenges of *group key management* is how to assure re-keying using the minimum bandwidth overhead without increasing the storage overhead. Proposed solutions in the literature, as we will see in the following sections, trade bandwidth overhead for storage overhead to achieve the best overall performance.

III. Group Key Management Requirements

Efficient group key management protocols should take into consideration miscellaneous requirements. Figure 2 summarizes these requirements from four points of view: security, quality of service, KS's resources, and group members' resources.

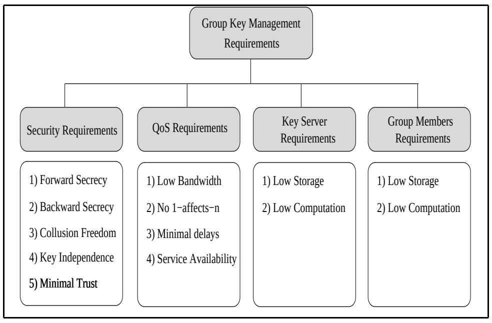


Fig. 2. Group Key Management Requirements

Security requirements:

1. *Forward secrecy* requires that users who left the group should not have access to any future key. This ensures that a member cannot decrypt data after it leaves the group. To assure forward secrecy, a re-key of the group with a new TEK after each leave from the group is the ultimate solution.
2. *Backward secrecy* requires that 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. To assure backward secrecy, a re-key of the group with a new TEK after each join to the group is the ultimate solution.
3. *Collusion freedom* requires that any set of fraudulent users should not be able to deduce the current traffic encryption key.
4. *Key independence*: a protocol is said *key independent* if a disclosure of a key does not compromise other keys.
5. *Minimal trust*: the key management scheme should not place trust in a high number of entities. Otherwise, the effective deployment of the scheme would not be easy.

Quality of service requirement:

1. *Low bandwidth overhead*: the re-key of the group should not induce a *high number of messages*, especially for dynamic groups. Ideally, this should be independent from the group size.
2. *1-affects-n*: a protocol suffers from the *1-affects-n* phenomenon if a single membership change in the group affects all the other group members. This happens typically when a single membership change requires that all group members commit to a new TEK.
3. *Minimal delays*: many applications that are built over the multicast service (typically, multimedia applications) are sensitive to jitters and delays in packet delivery. Therefore, any key management scheme should take this into consideration and hence minimizes the impact of key management on the *delays of packet delivery.*
4. *Service availability*: the failure of a single entity in the key management architecture must not prevent the operation of the whole multicast session.

Other requirements:

1. The key management scheme must not induce neither high *storage of keys* nor high *computation overhead* at the key server or group members.

IV. A Taxonomy of Group Key Management

A critical problem with any re-key technique is scalability: as a re-keying 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. Thereby, some solutions propose to organize the secure group into subgroups with independent local *TEKs*. This reduces the impact of re-keying, but requires data transformation at the borders of subgroups as we will see in the following sections. Therefore, we can classify existing solutions into two approaches: the *Common TEK* approach and the *TEK per sub-group approach* as illustrated in figure 3. In what follows, we present each class of protocols and we further refine the classification in order to highlight the underlying common concepts and mechanisms. We will illustrate each identified sub-category with relevant protocols from the literature.

V. Common TEK Approach

In this approach, all group members share a *common* Traffic Encryption Key (TEK). The management of this single key can be further classified into three classes: *centralized*, *decentralized* or *distributed.* Figure 3 illustrates this classification.

Open Science Index, Computer and Information Engineering Vol:2, No:10, 2008 waset.org/Publication/11804

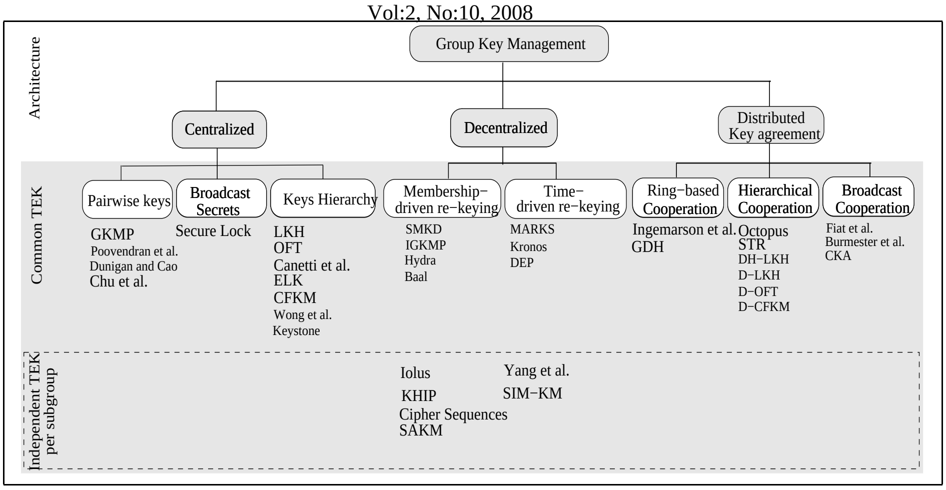


Fig. 3. Taxonomy of *Common TEK* Group Key Management Protocols

A. Centralized Protocols

In this approach, the key distribution function is assured by a single entity which is responsible for generating and distributing the traffic encryption key (TEK) whenever required. In figure 3, centralized protocols are further classified into three sub-categories depending on the technique used to distribute the TEKs. In what follows, we present each sub-category:

A.1 Pairwise Keys

In this sub-category of protocols, the *Key Server* shares a secret key with each group member. These keys are generally called: *Key Encryption Keys (KEK)* and are used to establish secure channels between the KS and each member in order to re-distribute the current TEK securely whenever required.

Group Key Management Protocol (GKMP): Harney and Muckenhirn [25, 26] proposed the Group Key Management Protocol (GKMP) that uses this approach. The key server shares a secret key with each valid group member (*KEKs*). In GKMP, the key server generates a Group Key Packet (GKP) that contains two keys: a Group TEK (GTEK) and a Group KEK (GKEK). The GTEK is used to encrypt the traffic and the GKEK is used to secure the distribution of a new GKP whenever required. When a new member joins the session, the key server generates a new GKP (which contains a new GTEK to assure backward secrecy) and sends it securely to the new member encrypted with the KEK established with this new member, and sends it to the other members encrypted with the old GTEK. The key server refreshes the GKP periodically and uses the GKEK for its distribution to the group members. When a member leaves the group, the key server generates a new GKP and sends it to each remaining member encrypted with the KEK that it shares with each member. Thus to assure forward secrecy, GKMP requires $O(n)$ re-key messages for each leave from the group. Therefore, this solution does not scale to large groups with highly dynamic members.

Dunigan and Cao [21] proposed a similar protocol that suffers from the same issues. Poovendram et al. [38] have also proposed a similar scheme to GKMP, where authentication and authorization functions are delegated to other group members rather than centralized at the same group controller entity.

Hao-hua Chu et al. protocol: In the solution proposed by Hao-hua Chu et al. in [14], a *Group Leader* shares a secret *Key Encryption Key (KEK)* with each group member. To send a secret multicast message m, the sender encrypts m with a random key k. Then, the sender encrypts k with the secret KEK that it shares with the group leader, and sends it to the group along with the encrypted message. Upon receiving the message, receivers cannot decrypt it since they do not know the random key k. When the leader receives the message, it decrypts k using the key that it shares with the source and constructs a validation message which contains k encrypted with each KEK that the leader shares with a valid group member (excluding the departing members). Upon receiving the validation message, each receiver decrypts k using its KEK and hence decrypts m which was encrypted using k. This protocol has the drawback to require the transmission of the *validation* multicast message by the group leader, with a size in the order of $O(n)$ (n being the number of current valid group members), after each time the source sends a message to the group.

A.2 Broadcast Secrets Approach

In this sub-category of protocols, the re-keying of the group is based on broadcast messages instead of peer to peer secret transmissions.

Secure Locks: Chiou and Chen [13] proposed Secure Lock; a key management protocol where the key server requires only a single broadcast to establish the group key or to re-key the entire group in case of a leave. The protocol relies on the following theorem:

Theorem 1: Chinese Remainder Theorem Let $m_1, \ldots, m_n$ be pairwise relatively prime positive integers, and let $a_1, \ldots, a_n$ be any integers. Then the system of linear congruences in one variable given by:

$$x \equiv a_1 \bmod m_1$$

$$\ldots$$

$$x \equiv a_n \bmod m_n$$

has a unique solution modulo $M = m_1 \times m_2 \times \ldots m_n$. The unique solution is: $x = \sum_{i=1}^{n} a_i M_i y_i \bmod M$, where $M_i = M/m_i$ and $y_i = M_i^{-1} \bmod m_i$.

In this protocol, the key server assigns a positive integer m_i to each member and shares a secret value k_i with each of them. When the server wants to send a message to the group, it generates a random value K and uses it to encrypt the message. Then, it encrypts K with each secret k_i and obtains the set $\{K_i\}$ of the encryptions of K ($K_i = \{K\}_{k_i}$). Then the server computes a lock M which is the solution to the equation system:

$$M \equiv K_1 \bmod m_1$$

$$\ldots$$

$$M \equiv K_n \bmod m_n$$

Then the server multicasts the lock M as well as the encrypted message with K. Upon reception of the lock M, each member recovers the encryption key $K = \{M \bmod m_i\}_{k_i}$, and hence decrypts the received message. Only members whose secret k_i and its corresponding positive integer m_i are included in the computation of the lock M, can recover the decryption key K.

This protocol minimizes the number of re-key messages. However, it increases the computation at the server due to the Chinese Remainder calculations before sending each message to the group.

A.3 Hierarchy of Keys Approach

We showed that in the *pairwise keys* approach, re-keying induces a high number of update messages (in the order of $O(n)$, with n being the number of group members). This is due mainly to the fact that the key server establishes a secure channel individually with each member and uses this channel to distribute the TEK updates. In order to reduce the number of update messages, in this sub-category of protocols, the key server shares secret keys with subgroups of the entire secure group in addition to the individual channels. Then, when a member leaves the secure session, the key server uses the secret sub-group keys, that are unknown by the leaving member, to distribute the new TEK. Thereby, sub-group secret keys allow to reduce the required number of update messages. In what follows we present some protocols that use this concept for re-keying:

Local Key Hierarchy (LKH): Independently, Wong et al. [49,50] and Wallner et al. [48] proposed the Logical Key Hierarchy (LKH) protocol. In LKH, the key server maintains a tree of keys. The nodes of the tree correspond to KEKs and the leaves of the tree correspond to secret keys shared with the members of the group. Each member holds a copy of its leaf secret key and all the KEKs corresponding to the nodes in the path from its leaf to the root. The key corresponding to the root of the tree is the TEK. For a balanced binary tree, each member stores at most $1 + log_2(n)$ keys, where n is the number of group members.

This *key hierarchy* allows to reduce the number of re-key messages to $O(\log n)$ instead of $O(n)$ in GKMP.

Example: 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 a hierarchy of keys as shown in figure 4. 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 re-keying messages. According to figure 4 : u_1 owns{ k_1, k_{12}, k_{1234}, TEK}, u_2 owns{ k_2, k_{12}, k_{1234}, TEK}, u_3 owns{ k_3, k_{34}, k_{1234}, TEK}, u_4 owns{ k_4, k_{34}, k_{1234}, TEK}, u_5 owns{ k_5, k_{56}, TEK} and u_6 owns{ k_6, k_{56}, TEK}.

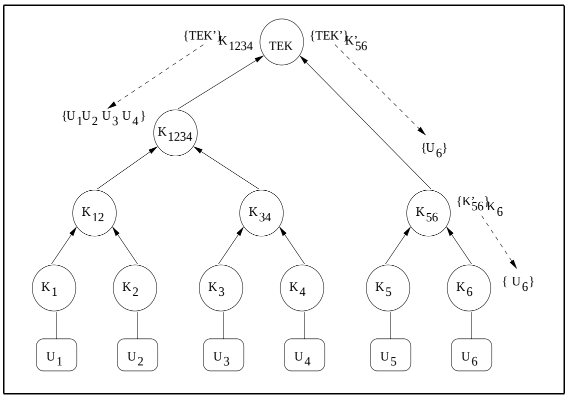


Fig. 4. key Hierarchy

Let us assume that u_5 leaves the group, KS updates k_{56} into k'_{56}, sends k'_{56} to u_6 encrypted with k_6. 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 if GKMP were used.

Wong et al. [49,50] proposed the extension of the binary key tree to a k-ary key tree. Using a greater degree reduces the number of keys maintained by the members because of a smaller tree depth. Performance analysis shows that optimal results are reached with trees having a degree less or equal to 4. The authors propose also the Keystone architecture [51], where the key server is aided by secondary controllers called registrars, whose role is limited to registration and authentication of new members.

One-way Function Trees (OFT): McGrew and Sherman [2,30] proposed an improvement over LKH called One-way Function Trees (OFT). OFT allows to reduce the number of re-key messages from $2\log_2(n)$ to only $\log_2(n)$. With OFT, a KEK is calculated by members rather than attributed by the key server. Indeed, each KEK k_i is computed using its child KEKs using the formula:

$$k_i = f(g(k_{left(i)}), g(k_{right(i)})) \tag{1}$$

Open Science Index, Computer and Information Engineering Vol:2, No:10, 2008 waset.org/Publication/11804

where $left(i)$ and $right(i)$ denote respectively the left and right children of node i, and g is a one-way function. The result of applying g to a key k: $g(k)$, is called the *blinded key* version of k.

In this protocol, each member maintains its leaf secret key and its *blinded sibling* key and the set of blinded sibling KEKs of its *ancestors*. Figure 5 illustrates the ancestors and their corresponding sibling keys of member $u2$.

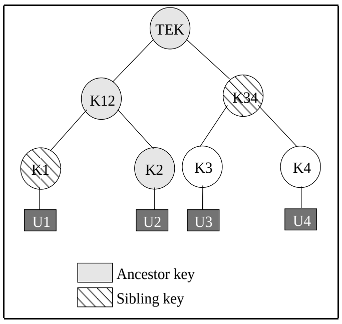


Fig. 5. Ancestor and Sibling keys of member U_2.

Using formula 1, each member can calculate all the required ancestor KEKs (KEKs on the nodes in the path from the leaf secret to the root) recursively. In the original scheme (LKH), when a new KEK is generated, it is encrypted with its two child KEKs. However, in OFT when a blinded key is changed in a node it has to be encrypted only with the key of its sibling node. Hence, the required number of re-key messages is reduced by half.

Example: Let us consider the hierarchy of keys in figure 6.

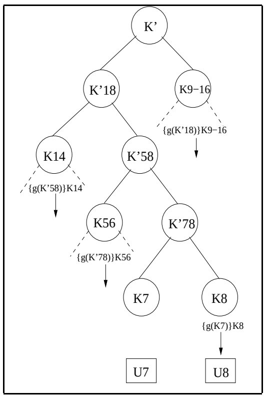


Fig. 6. Example of a *OFT* scenario. U_7 joins the session.

When user U_7 joins the group, the keys K_{78}, K_{58}, K_{18} and the group key K, should be modified into K'_{78}, K'_{58}, K'_{18} and K', respectively. In order to redistribute the new group key and the modified *KEK*s, the only values that should be sent, are the *blinded* keys: $g(K_7)$, $g(K'_{78})$, $g(K'_{58})$, and $g(K'_{18})$, respectively encrypted with: K_8, K_{56}, K_{14}, and K_{9-16}. The new *TEK* and *KEK*s can be now calculated as follows: $K'_{78} = f(g(K_7), g(K_8))$, $K'_{58} = f(g(K_{56}), g(K'_{78}))$, $K'_{18} = f(g(K_{14}), g(K'_{58}))$, and $K' = f(g(K'_{18}), g(K_{9-16}))$. In this example, user U_8 maintains K_8, $g(K_{56})$, $g(K_{14})$, and $g(K_{9-16})$. When it receives $\{g(K_7)\}_{K_8}$, it calculates recursively all the keys on its path to the root of the hierarchy, using the above formulas. These calculations culminate into the new group key K'.

Canetti et al. [10] proposed a similar approach that has the same communication overhead. The proposed scheme called: one-way function chain tree, uses a pseudo-random-generator to generate the new KEKs rather than a one-way function. Perrig et al. proposed yet another similar protocol called: Efficient Large group Key distribution (ELK) [37], that uses Pseudo Random Functions to generate the new KEKs.

Centralized Flat table Key Management (CFKM): Waldvogel et al. [47] proposed the Centralized Flat Table Key Management protocol (CFKM). In this approach, the key hierarchy is replaced by a flat table in order to reduce the number of keys maintained by the *Key Server*. The table contains a single entry for the TEK and $2w$ entries for the KEKs, where w is the number of bits in a member identifier (the authors propose to use IP addresses as member identifiers). Two keys are associated to the two possible values of each bit in a member id. Figure 7 illustrates the structure of the table for $w = 4$.

TEK	
KEK00	KEK01
KEK10	KEK11
KEK20	KEK21
KEK30	KEK31

Fig. 7. CFKM table with $w = 4$

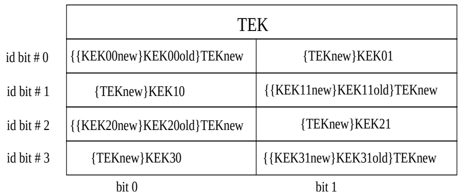
Each member holds the KEKs associated to the values of its identifier bits. Thus, each member holds $w + 1$ keys (w KEKs in addition to TEK). For instance, a member with the identifier 0101 maintains KEK00, KEK11, KEK20, KEK31 and the TEK. When a member leaves the group, all the keys held by this departing member should be modified to assure forward secrecy. Therefore the key server sends a re-key message containing two parts: a first part contains the TEK encrypted with each not compromised KEK from the flat table, and hence all the remaining members would be able to decrypt the new TEK. The second part contains the new KEKs encrypted with both the old KEK and the new TEK. This way, the leaving member cannot recover the new TEK and the remaining members can update their old KEKs without having access to the KEKs of other members. Figure 8 illustrates the re-key message sent by the key server after the leave of the member with the identifier 0101.

A.4 Comparison

In table I we compare the above protocols against the following relevant criteria:

1. *1-affects-n*
2. *Storage at the key server*: the number of keys that should be maintained by a key server.

Open Science Index, Computer and Information Engineering Vol:2, No:10, 2008 waset.org/Publication/11804



	TEK	
id bit # 0	{{KEK00new}KEK00old}TEKnew	{TEKnew}KEK01
id bit # 1	{TEKnew}KEK10	{{KEK11new}KEK11old}TEKnew
id bit # 2	{{KEK20new}KEK20old}TEKnew	{TEKnew}KEK21
id bit # 3	{TEKnew}KEK30	{{KEK31new}KEK31old}TEKnew
	bit 0	bit 1

Fig. 8. CFKM re-key message when member 0101 leaves the group

3. *Storage at a member*: the number of keys that should be maintained by a group member.
4. *Join re-key overhead*: number of re-key messages sent by the key server to redistribute the group TEK after a join.
5. *Leave re-key overhead*: number of re-key messages sent by the key server to redistribute the group TEK after a leave.

The GKMP protocol achieves an excellent result for storage at the members. However, this result is achieved by providing no method for re-keying the group after a member has left, except re-creating the entire group which induces a $O(n)$ re-key messages overhead, with n being the number of the remaining group members. Secure Lock achieves also excellent results for storage and communication overheads on both members and the key server. However, these results are achieved by increasing the computation overhead at the key server due to the Chinese Remainder calculations. So far, the best solutions for centralized group key management appear to be those using a hierarchical tree of KEKs. They achieve good overall results without compromising any aspects of security.

B. Decentralized Architectures

In this approach, a hierarchy of key managers share the labor of distributing the TEK to group members in order to avoid bottlenecks and single point of failure. We distinguish two sub-categories corresponding to the case where the TEK is modified after each membership change (*membership driven*), or systematically after each slot of time (*time driven*) (cf. figure 3).

B.1 Membership-Driven Re-keying

In this sub-category of protocols, the TEK is changed each time a join or a leave occurs in the membership of the group. In what follows we present some protocols relying on this approach.

Scalable Multicast Key Distribution (SMKD): Ballardie proposed in RFC1949 [3] the Scalable Multicast Key Distribution (SMKD); a protocol that exploits the tree built by the Core Based Tree multicast routing protocol (CBT) [4, 5] to deliver keys to multicast group members. In a CBT architecture, the multicast tree is rooted at a main core. Secondary cores can exist eventually. The main core creates an access control list (ACL), a session key GTEK and key encryption key GKEK used to update the session key GTEK. The ACL, the GTEK and the GKEK are transmitted to secondary cores and other nodes when they join the multicast tree after their authentication. Any router or secondary core authenticated with the primary core can authenticate joining members and use the ACL to distribute the keys, but only the main core generates those keys. With SMKD there is no solution for forward secrecy other than to recreate an entirely new group without the leaving members.

Intra-domain Group Key Management Protocol (IGKMP): DeCleene et al. [15, 24] proposed the Intra-domain Group Key Management Protocol IGKMP. This architecture divides the network into administratively scoped areas. There is a Domain Key Distributor (DKD) and many Area Key Distributors (AKD). Each AKD is responsible for one area. Figure 9 illustrates an example of this architecture.

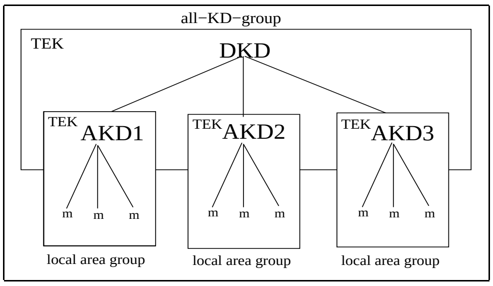


Fig. 9. An example of an Inter-domain Group Key Management (IGKMP) Architecture

The group TEK is generated by the DKD and is propagated to the group members through the AKDs. The DKD and AKDs belong to a multicast group called All-KD-Group. The DKD uses this group to transmit re-key messages to the AKDs who re-key in turn their respective areas. This architecture suffers from a single point of failure which is the DKD; the sole entity responsible for generating the group TEK. Besides, in case of an AKD failure, members belonging to the same area will be not able to access the group communication.

Hydra: Rafeli and Hutchison [39] proposed Hydra protocol, in which the group is organized into smaller subgroups, and a server called the Hydra server (HS_i) controls each sub-group i. If a membership change occurs at subgroup i, the corresponding HS_i generates the group TEK and sends it to the other HS_js involved in that session. In order to have the same group TEK distributed to all HSs a special protocol is used to ensure that only a single valid HS is generating the new TEK whenever required.

Baal Chaddoud et al. [11] proposed a similar protocol called Baal which defines three entities:

1. *The group controller (GC)*: maintains a participant list (PL) and creates and distributes the group key (TEK) to group members via local controllers.
2. *Local controllers (CL)*: the GC delegates a LC to each subnet (generally a local network) to manage the keys within its subnet. When a LC receives a new TEK it distributes it to the members connected to its subnet. Besides, a LC can play the role of the GC by generating and distributing new TEKs after membership

TABLE I

Comparison of centralized group key management

Protocol	1-affects-n	Re-key Overhead			Storage Overhead	
		Join		Leave	Key Server	Member
		Multicast	Unicast			
GKMP	Yes	2	2	$2n$	$n+2$	3
LKH	Yes	$\log_2(n)-1$	$\log_2(n)+1$	$2\log_2(n)$	$2n-1$	$\log_2(n)+1$
OFT	Yes	$\log_2(n)+1$	$\log_2(n)+1$	$\log_2(n)+1$	$2n-1$	$\log_2(n)+1$
CFKM	Yes	$2I$	$I+1$	$2I$	$2I+1$	$I+1$
Secure Lock	No	0	2	0	2n	2

n: number of group members, I: number of bits in a member id.

changes following some coordination rules.

3. *Group member*: a member in the PL.

When a membership change occurs at a subnet, the corresponding LC can generate a new TEK and distribute it to its subnet and to the other members via their LCs. To assure that a single LC generates a new TEK at a time, the GC assigns a priority to each LC and when many LCs distribute simultaneously a new TEK, the LCs are instructed to commit to the TEK issued by the LC having the highest priority.

R. Oppliger and A. Albanese [35] proposed a similar decentralized solution called *Distributed Registration and Key distribution (DiRK)*. In their architecture, the authors distinguish between *active* and *passive* group members. *Active* members have the ability to register other new members, and to generate and distribute *TEK*s whenever required. *DiRK* relies on a *PKI* to authenticate *active* members, and the origin of received *TEK*s.

B.2 Time-Driven Re-keying

In this sub-category of protocols, the TEK is changed after each specific period of time. Thereby, the departing members are not excluded immediately from having access to the secure content. Similarly, new members are appointed to wait for the beginning of a new interval of time before having access to the content. In what follows we present some protocols relying on this concept.

Kronos: Setia et al. [42] proposed the Kronos protocol. This protocol is driven by periodic re-keying rather than membership changes, which means that a new group TEK is generated after each period of time rather than after each membership change. Similarly to IGKMP, in Kronos a domain is divided into areas managed by different AKDs. However, in Kronos the DKD does not multicast the group TEK each time to the AKDs. Instead of that, each AKD generates independently the same group TEK whenever required and re-keys the members belonging to its area. To implement this scheme, the AKDs' clocks should be synchronized, and the AKDs have to agree on a re-key period. Second, the DKD transmits secret factors K and R_0 to AKDs using secure channels. To generate the group TEK R_{i+1}, AKDs calculate after each period of time: $R_{i+1} = E_K(R_i)$, which is the encryption of the previous TEK (R_i) with the encryption algorithm E using the secret key K.

MARKS: In MARKS, Briscoe [8] suggests slicing the time length to be protected (such as the transmission time of a TV show) into small portions of time and using a different key for encrypting each slice. The encryption keys are leaves in a binary hash tree that is generated from a single seed. A blinding function, such as MD5 [40] is used to create the tree nodes. Figure 10 shows an example of the generated binary tree whose leaves are the keys that correspond to the different slices.

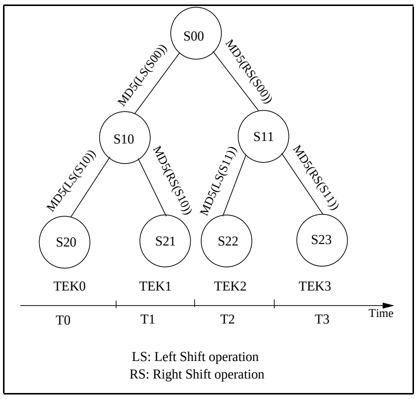


Fig. 10. An example of a MARKS key generation tree

Each intermediate node (including the root) allows to generate two children (left and right children). The left node is generated by shifting its parent one bit to the left and applying the blinding function on it. The right node is generated by shifting its parent one bit to the right and applying the blinding function on it (cf. figure 10). Users willing to access the group communication receive the seeds needed to generate the required keys. The system cannot be used in situations where a membership change requires the change of the group key, since the keys are changed as a function of the time. The distribution of the seeds and the management of receivers' queries are assured by a set of key managers.

Dual Encryption Protocol (DEP): A common drawback of most of decentralized protocols is the involvement of a high number of intermediary parties. In practice it is difficult to assume trustiness for all of these entities. In order to solve the problem of trusting third parties, Dondeti et al. [18–20] proposed the Dual Encryption Protocol (DEP). In their work, they suggest hierarchical sub-grouping of the group members where a sub-group manager (SGM) controls each sub-group. There define three types of KEKs and one Data Encryption Key (DEK): KEK_{i1} is shared between a SGM_i and its sub-group mem-

Open Science Index, Computer and Information Engineering Vol:2, No:10, 2008 waset.org/Publication/11804

Open Science Index, Computer and Information Engineering Vol:2, No:10, 2008 waset.org/Publication/11804

bers. KEK_{i2} is shared between the Key Server (KS) and the group members of sub-group i excluding SGM_i. Finally, KS shares KEK_{i3} with SGM_i. In order to distribute the DEK to the group members, the KS generates and transmits a package containing the DEK encrypted with KEK_{i2} and encrypted again with KEK_{i3}. Upon receiving the package, SGM_i decrypts its part of the message using KEK_{i3} and recovers the DEK encrypted with its sub-group KEK (KEK_{i2}), which is not known by the SGM_i. SGM_i encrypts this encrypted DEK using KEK_{i1} shared with its sub-group members and sends it out to sub-group i. Each member of sub-group i decrypts the message using KEK_{i1} and then, decrypting the message using KEK_{i2} (shared with KS), recovers DEK. Therefore, the DEK cannot be recovered by any entity that does not know both keys. Hence, although there are third parties involved in the management (SGMs), they do not have access to the group key (DEK). When the membership of sub-group i changes, the SGM_i changes KEK_{i1} and sends it to its members. Future DEK changes cannot be accessed for members of sub-group i that did not received the new KEK_{i1}.

B.3 Comparison

In table II, we compare some of the above protocols against the following criteria:

1. *Key independence*
2. *1-affects-n*
3. *Local re-keying*: membership changes in a sub-group should be treated locally.
4. *Data transformation*: data is transformed using some means when messages pass from a sub-group to another.

Kronos does not provide key independence because it generates new keys based on old ones, and if any past key is compromised , all future keys are disclosed. The same happens with MARKS if a seed is compromised.

We note that even though these protocols divide the whole group into sub-groups they still suffer from *1-affects-n* phenomenon because of using the same TEK for all sub-groups.

C. Distributed Key-Agreement Protocols

With distributed key-agreement protocols, the group members cooperate to establish a group key. This improves the reliability of the overall system and reduces the bottlenecks in the network in comparison to the centralized approach. In figure 3, we further classify the protocols of this category into three sub-categories depending on the virtual topology created by the members for cooperation.

C.1 Ring-Based Cooperation

In this sub-category, the cooperation of group members forms a virtual ring, as we will see in the following protocols.

INGEMARSON ET AL. PROTOCOL: The protocol proposed by Ingemarson et al. in [27] is one of the earliest propositions to extend Diffie-Hellman key agreement protocol [16] to group communication. In this protocol, members are organized into a virtual ring; in a way that member M_i communicates with member M_{i+1} and member M_n with member M_1. The group members compute the group key within $(n-1)$ rounds. Initially, each member M_i generates a random value N_i, computes g^{N_i} and sends it to the next member M_{i+1}. Then, in each round, each member M_i raises to the power N_i the intermediate value received from member M_{i-1}, and sends the result to the member M_{i+1}. Each member performs n exponentiations and gets the group key $K_n = g^{N_1 N_2 \dots N_n}$ after $(n-1)$ rounds. This protocol is not suitable for dynamic groups because it requires the execution of the entire algorithm after each membership change.

GROUP DIFFIE-HELLMAN (GDH): Steiner et al. [46] proposed this group key exchange protocol as an extension of the Diffie-Hellman protocol [16] to establish group keys. The group agrees on a pair of primes (q and α) and starts calculating in a distributed way the intermediate values. The first member calculates the first value (α^{x_1}, with x_1 a random secret generated by the first member) and sends it to the next member. Each subsequent member receiving the set of intermediary values, raises them using its own secret number generating a new set: a set generated by the i^{th} member will have i intermediate values with $i-1$ exponents and a cardinal value containing all exponents. For example; the fourth member receives the set: $\{\alpha^{x_2x_3}, \alpha^{x_1x_3}, \alpha^{x_1x_2}, \alpha^{x_1x_2x_3}\}$ and generates the set $\{\alpha^{x_2x_3x_4}, \alpha^{x_1x_3x_4}, \alpha^{x_1x_2x_4}, \alpha^{x_1x_2x_3}, \alpha^{x_1x_2x_3x_4}\}$. The cardinal value in this example is $\alpha^{x_1x_2x_3x_4}$. The last member can easily calculate the group key k from the cardinal value: $k = \alpha^{x_1x_2x_3 \cdots x_n} \mod q$. The last member raises all intermediate values to its secret value and multicasts the whole set to all group members. Upon receiving this message, each member j extracts its respective intermediate value and calculates k by exponentiation of the j^{th} value to x_j. The setup time and the length of messages are linear in terms of the number of group members n since all members must contribute to generating the group key.

C.2 Hierarchy-Based Cooperation

OCTOPUS: Becket and Wille [6] proposed the Octopus protocol. This protocol is also based on Diffie-Hellman key exchange protocol [16]. In Octopus, the large group (composed of n members) is split into four sub-groups ($\frac{n}{4}$ members each). Each sub-group agrees internally on an intermediate DH value: $I_{subgroup} = \alpha^{u_1u_2 \dots u_n/4}$, where u_i is the contribution from user i, and then the subgroups exchange their intermediary values. All group members can then calculate the group key. The leader member in each sub-group is responsible for collecting contributions from its sub-group members and calculating the intermediary DH value ($I_{subgroup}$). Let us call the subgroup leaders A, B, C and D. First, A and B, using DH, exchange their intermediary values I_a and I_b, creating $\alpha^{I_a \cdot I_b}$. Also, C and D do the same and create $\alpha^{I_c \cdot I_b}$. Then, A and C exchange $\alpha^{I_a \cdot I_b}$ and $\alpha^{I_c \cdot I_d}$. Leaders, B and D do the same. Now, all of them can calculate $\alpha^{I_a \cdot I_b \cdot I_c \cdot I_d}$. After that, A, B, C

TABLE II

Comparison of some decentralized group key management protocols

Protocol	Key independence	1-affects-n	Local re-key	Data transformation
SMKD	Yes	Yes	No	No
IGKMP	Yes	Yes	No	No
DEP	Yes	Yes	No	No
Kronos	No	-	No	No
Hydra	Yes	Yes	No	No
MARKS	No	-	No	No

and D send to their respective subgroups $\alpha^{\frac{I_a \cdot I_b \cdot I_c \cdot I_d}{u_i}}$, where $i = 1 \ldots (n-4)/4$, and all members of the group are capable of calculating the group key.

Skinny TRee (STR): The *Skinny TRee (STR)* protocol, proposed by Steer et al. [45] and undertaken by Kim et al. [29], is a contributive protocol using a tree structure. Figure 11 illustrates an STR tree with four members.

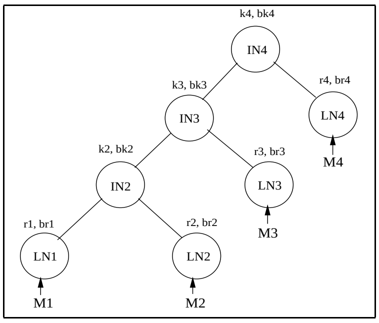


Fig. 11. An example of a STR tree

The leaves are associated to group members. Each leaf is identified by its position LN_i in the tree and holds a secret random r_i (generated by the corresponding member M_i) and its public blinded version $br_i = g^{r_i}$ mod p, where g and p are DH parameters. Each internal node is identified by its position IN_i in the tree and holds a secret random k_i and its blinded public version $bk_i = g^{k_i}$ mod p. Each secret k_i is recursively calculated as follows: $k_i = (bk_{i-1})^{r_i}$ mod p= $(br_i)^{k_{i-1}}$ mod p. The group key is the key associated to the root: $k_n = g^{r_n g^{r_{n-1} \cdots g^{r_2 r_1}}}$. Due to the linear structure of the tree, this solution induces a $O(n)$ key calculations in order to establish the group key associated to the root of the tree. Besides, each member should store and maintain all the public keys associated to all the nodes of the tree. In case of a membership change (join / leave) the tree is re-built consequently and hence all the members update the group key which is the new key k_n associated to the root of the tree.

Diffie Hellman - Logical Key Hierarchy (DH-LKH): Perrig et al. [28,36] proposed a variant of STR using a binary tree (less deeper) in order to reduce the number of key calculations from the order of $O(n)$ to $O(\log(n))$. Indeed the proposed protocol is a distributed version of LKH. The tree is built recursively from bottom to up. Initially, each member M_i generates a random r_i as a secret key associated to its leaf. To build upper level of the tree, two members: one as a leader of a left subtree and another one as a leader of a right subtree, broadcast their respective DH computations and hence allow to all the members to calculate the group key corresponding to the root of the tree. Using the tree in figure 12, intermediate keys are $k_{12} = \alpha^{k_1 k_2}$ mod p, $k_{34} = \alpha^{k_3 k_4}$ mod p and the group key is $k_{14} = \alpha^{k_{12} k_{34}}$ mod p.

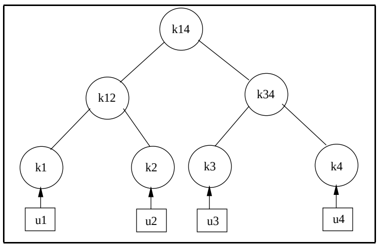


Fig. 12. LKH Tree

Distributed Logical Key Hierarchy (D-LKH): A similar distributed approach based on the logical key hierarchy has been proposed by Rodeh et al. in [41]. In this approach, the Key Server is completely abolished and the logical key hierarchy is generated among the members, therefore there is no entity that knows all the keys at the same time. This protocol uses the notion of subtrees agreeing on a mutual key. This means that two groups of members, namely subtree L and subtree R, agree on a mutual encryption key. Member m_l is assumed to be L's leader and member m_r is R's leader. Subtree L has subtree key k_L and subtree R has subtree key k_R. The protocol used to agree on a mutual key goes as follows:

1. Member m_l chooses a new key k_{LR}, and sends it to member m_r using a secure channel.
2. Member m_l encrypts key k_{LR} with key k_L and multicasts it to members of subtree L; member m_r encrypts key k_{LR} with key k_R and multicasts it to members of subtree R.
3. All members ($L \cup R$) receive the new key.

Similarly, Dondeti et al. [17] and Waldvogel et al. [47] propose distributed versions of OFT [2, 30] (D-OFT) and CFKM [47] (D-CFKM) protocols respectively.

C.3 Broadcast-Based Approach

In this approach, the key agreement relies on broadcasting secret messages and distributed computations that culminate into the group key.

Fiat and Naor protocol: Fiat and Naor [22] proposed a protocol that relies on Diffie-Hellman property

Open Science Index, Computer and Information Engineering Vol:2, No:10, 2008 waset.org/Publication/11804

Open Science Index, Computer and Information Engineering Vol:2, No:10, 2008 waset.org/Publication/11804

that consists in the requirement of each member to broadcast a single message to the other participants in order to agree on a common secret. In the proposed protocol, a reliable center T initializes the system. T choses two primes q_1 and q_2 and broadcasts $p = q_1.q_2$ to all nodes. Then, T generates a random g and keeps it secret. When a new member M_i joins the group, T sends to this new member two values: a random x_i (which is relatively prime with each other x_j previously generated for members M_j), and a key $\alpha_i = g^{x_i} \bmod p$. M_i keeps α_i secret. To agree on a group key K, each member broadcasts its values x_i and hence each of them calculates $K = \alpha_i^{\prod_{j=1,j\neq i}^{n} x_j} \bmod p$ $= g^{x_1 x_2 \dots x_n} \bmod p$. This protocol is not robust against collusions as shown in [22], and has the drawback to require a reliable third party: T.

Burmester and Desmedt protocol: Burmester and Desmedt [9] proposed a very efficient protocol that executes in only three rounds:

1. member m_i generates its random exponent r_i and broadcasts $Z_i = \alpha^{r_i}$;
2. member m_i computes and broadcasts $X_i = (Z_{i+1}/Z_{i-1})^{r_i}$;
3. member m_i can now compute the key $K_n = Z_{i-1}^{nr_i}.X_i^{n-i}\dots.X_{n-2} \bmod p$.

The group key calculated by each member is then $K_n = \alpha^{N_1 N_2 + N_2 N_3 + \dots N_n N_1}$. This protocol requires $n+1$ exponentiations per member and in all but one the exponent is at most $n-1$. The drawback is the requirement of $2n$ broadcast messages.

Conference Key Agreement (CKA): Boyd [7] proposed yet another protocol for conference key agreement (CKA) where all group members contribute to generating the group key. The group key is generated with a combining function: $K = f(N_1, h(N_2), \dots, h(N_n))$, where f is the combining function (a MAC), h is a one-way function, n is the group size, and N_i is the contribution from group member i. The protocol specifies that $n-1$ members broadcast their contributions (N_i) in the clear. The group leader, for example U_1, encrypts its contribution (N_1) with the public key of each member and broadcasts it. All group members who had their public key used to encrypt N_1 can decrypt it and generate the group key.

C.4 Comparison

In table III we compare the presented distributed key management protocols against the following criteria:

1. *Number of rounds*: the number of rounds required before the members commit to a group key.
2. *Number of messages*: the number of messages required to establish the group key.
3. *DH exchange*: whether the protocol is based on Diffie-Hellman key exchange.
4. *Leader required*: whether the protocol requires the existence of a leader or leaders for the operation of the key agreement protocol.

The protocols that do not rely on a group leader have an advantage over those with a group leader because, without a leader, all members are treated equally and if one or more members fail to complete the protocol, it will not affect the whole group. In the protocols with a group leader, a leader failure is fatal for creating the group key and the operation has to be restarted from scratch. We did not consider the *1-affects-n* phenomenon because in distributed protocols all the members are contributors in the creation of the group key and hence all of them should commit to the new key whenever a membership change occurs in the group.

VI. Independent TEK per sub-group

The *common TEK* approach has the drawback to require that all group members commit to a new TEK, whenever a membership change occurs in the group, in order to ensure *perfect backward and forward secrecy.* This is commonly called *1-affects-n* phenomenon. In order to mitigate the *1-affects-n* phenomenon, another approach consists in organizing group members into sub-groups. Each sub-group uses its own independent TEK. Indeed, in this scheme when a membership change occurs in a subgroup, it affects only the members belonging to the same sub-group. The existing protocols that use independent TEK per sub-group fall into two sub-categories: the *membership-driven re-keying* protocols that do re-keying after each membership change, and *time-driven re-keying* protocols that do batch re-keying after each period of time. Figure 3 illustrates this classification.

A. Membership-Driven Re-keying

Iolus: Mittra [31] proposed Iolus, a framework of a hierarchy of multicast sub-groups. Each sub-group is an independent multicast group (with its own multicast address and eventually its own multicast routing protocol). The overall sub-groups form a virtual multicast group. Each sub-group is managed by a Group Security Agent (GSA) which is responsible for key management inside the sub-group. A main controller called the Group Security Controller (GSC) manages the GSAs. Figure 13 illustrates a hierarchy with six sub-groups. Each of them uses its own TEK.

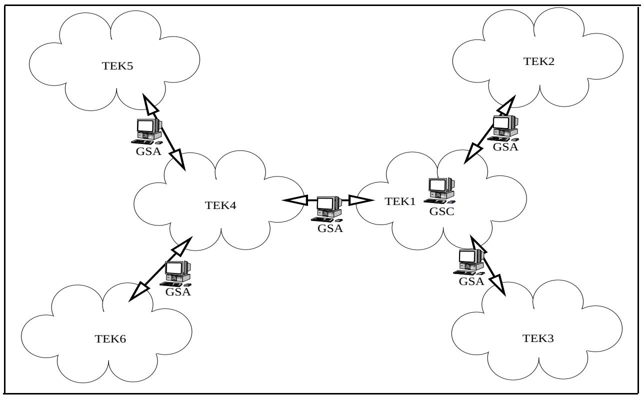


Fig. 13. An example of a Iolus architecture

When a membership change occurs in a sub-group, only that sub-group is involved in a re-key process. This way,

Open Science Index, Computer and Information Engineering Vol:2, No:10, 2008 waset.org/Publication/11804

TABLE III

Comparison of Distributed Key Management Protocols

Scheme	Nb. rounds	Nb. messages		DH exchange	Leader required
		multicast	unicast		
Ingemarson et al.	$n-1$	0	$n(n-1)$	Yes	No
GDH	n	n	$n-1$	Yes	No
Octopus	$2(n-1)/4+2$	0	$3n-4$	Yes	Yes
STR	n	n	0	Yes	No
DH-LKH	$\log_2 n$	$\log_2 n$	0	Yes	No
D-LKH	3	1	n	No	Yes
D-OFT	$\log_2 n$	0	$2\log_2 n$	No	No
D-CFKM	n	0	$2n-1$	No	No
Fiat et al.	2	n	n	Yes	Yes
Burmester et al.	3	$2n$	0	No	No
CKA	3	n	$n-1$	No	Yes

Iolus scales to large groups and mitigates *1-affects-n* phenomenon. However, Iolus has the drawback of affecting the data path. Indeed, there is a need for translating the data that goes from one sub-group, and thereby one key, to another. This induces decryption / re-encryption operations that are not tolerated by most of delay sensitive applications.

Keyed HIerarchical multicast Protocol (KHIP): Shields et al. [44] proposed the Keyed HIerarchical multicast Protocol (KHIP). KHIP is based on a multicast tree built using OCBT [43] routing protocol, and uses a different TEK per each branch of the tree. It uses also an authentication service [23] based on certification to authenticate members and on-tree routers. The multicast tree is organized into sub-branches. Each sub-branch is managed by a trusted router which manages the TEK used within this sub-branch. When a source is ready to send a message to the group, it generates a random key K_T, encrypts the message with K_T, and encrypts K_T with the TEK of the sub-branch to which the source is attached. Then the source puts the encrypted K_T in the header of the packet carrying the message and multicasts the packet. The members located in the same sub-branch know the TEK of the sub-branch and hence can decrypt the K_T and then decrypt the message with K_T. When a border router of the sub-branch (a trusted router at the intersect between two sub-branches) receives the packet, it decrypts the K_T and re-encrypts it using the TEK of the adjacent sub-branch to which the so translated packet will be forwarded. This process is followed until the message reaches all the group members. When a new member joins a sub-group, the router responsible for that sub-branch generates and distributes a new TEK for the sub-branch encrypted with the old one. When a member leaves a sub-branch, the corresponding router distributes a new TEK encrypted with the public key of each remaining member and signed with the router's private key.

Even thought KHIP reduces the decryption / re-encryption operations to a single key per packet, it still suffers from the delay variations of packet delivery due to these operations, and most of applications that require real-time transmission do not tolerate such delays.

Cipher Sequences (CS): Molva and Pannetrat [32] proposed a framework for multicast security that is based on Reversible Cipher Sequences. A function $f(S,a)$ is called Cipher Group (CG) if it has the following characteristics: there is a sequence of n elements $a_i (1 \leq i \leq n)$, and there is a sequence of $n+1$ elements S_i $(0 \leq i \leq n)$, such as $S_i = f(S_{i-1}, a_i)$ for $i > 0$ and S_0 is the initial value; and for every pair (i,j), where $i > j$, there exists a function $h_{i,j}$ such as $S_i = h_{i,j}(S_j)$. The multicast tree is rooted at the source, the group members are the leaves and internal nodes are intermediate elements of the multicast communication. Now, let S_0 be the message to be multicast and let every node N_i be assigned a value $a_i > 1$. When node N_i receives a value S_j from its parent N_j, it computes $S_i = f(S_j, a_i)$ and sends S_i to its children that can be either leaves or other internal nodes. The leaves are assigned the function $h_{0,n}$, which enables them to compute S_0 from S_n, since $S_0 = h_{0,n}(S_n)$, and therefore recover the original data. Figure 14 shows an example of Molva's scheme that may be described as:

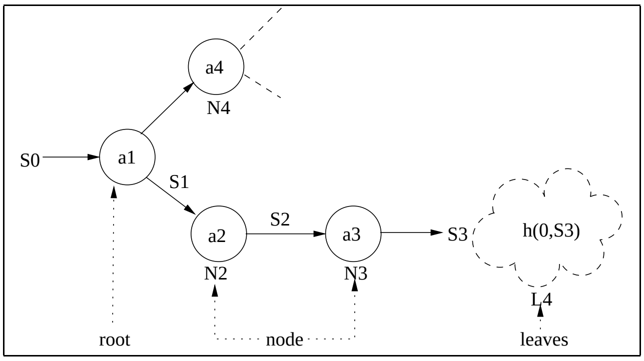


Fig. 14. An example of Molva's scheme

1. the root calculates $S_1 = f(S_0, a_1)$ and sends S_1 to N_2 and N_4;
2. node N_2 calculates $S_2 = f(S_1, a_2)$ and sends S_2 to N_3;
3. node N_3 calculates $S_3 = f(S_2, a_3)$ and sends S_3 to leaves L_4;
4. leaves L_4 calculate $S_0 = h_{0,4}(S_3)$ and recover the original data: S_0.

When a membership change occurs in a leaf, the corresponding node N_n receives a new value a'_n and all members in the leaf receives a new function $h'_{0,n}$. Naturally, if the membership change occurred because of member removal,

Open Science Index, Computer and Information Engineering Vol:2, No:10, 2008 waset.org/Publication/11804

the removed member will not receive the new $h'_{0,n}$, thus will not be able to recover S_0.

SAKM: Challal et al. [12] proposed a new *Scalable and Adaptive Key Management scheme (SAKM)* that addresses the *1-affects-n* and re-keying overheads by taking into consideration the dynamic aspect of the group members. SAKM tackles the scalability issue by organizing the multicast group into clusters, where each cluster uses its own TEK. Figure 15 illustrates the different components of the proposed architecture.

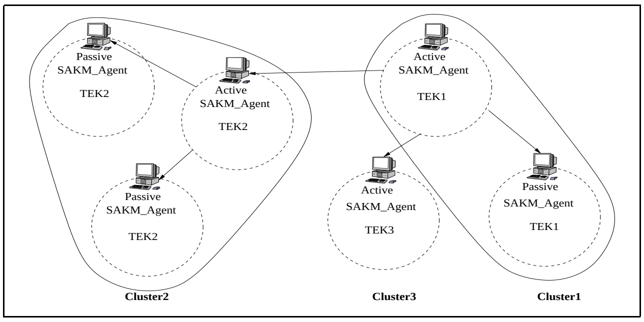


Fig. 15. SAKM Architecture

In contrast to existing solutions that use this concept, with SAKM: the organization of the group into clusters is updated periodically depending on the *dynamism* of the members during the secure session. The partitioning aims to minimize both *key translation* and re-keying overheads depending on the membership change behavior. Indeed, many studies [1] show that the membership behavior of group members in multicast sessions is likely to be not uniform through a large scale group and during the whole session. Some parts of the network may be more dynamic than others during some periods of time and become more stable afterward. In this case, it would be interesting to use a protocol that restricts the re-key to the areas with frequent membership changes. Thereby, SAKM is very efficient in such situations. 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.

B. Time-Driven Re-keying

YANG ET AL. PROTOCOL: Yand et al. [52] proposed a reliable re-keying approach. In the proposed architecture, the multicast group is organized into a set of subgroups, where each subgroup is managed by a Key Server (KS). The role of a KS is to re-key the members of 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 re-key that takes into consideration those membership changes. The overall KSs share a common *KS secret key*. When a KS receives a multicast message encrypted with its local TEK (sent by one of its subgroup members), it decrypts it and re-encrypts it using the *KS secret key*. Then, it multicasts the so re-encrypted message to the other KSs. In turn, these KSs decrypt the message using the *KS secret key* and re-encrypt it using their respective local TEKs. Then, each KS multicasts the so re-encrypted message to its subgroup.

SCALABLE INFRASTRUCTURE FOR MULTICAST KEY MANAGEMENT (SIM-KM): Mukherjee and Atwood [34] proposed a multicast key management infrastructure called SIM-KM: Scalable Infrastructure for Multicast Key Management. SIM-KM bases on subgrouping with message transformation by local controllers. In contrast to solutions based on subgrouping, SIM-KM uses *proxy encryption* [33] to transform data at the border of a subgroup. Proxy functions convert cipher text for one key into cipher text for another key without revealing secret decryption keys or clear text messages. This allows SIM-KM to do subgrouping with data transformation in order to limit the impact of re-keying, even thought intermediaries are not trusted entities.

C. Comparison

In table IV, we compare some of the above protocols against the following criteria:

1. *Key independence*
2. *1-affects-n*
3. *Local re-key*: membership changes in a sub-group should be treated locally.
4. *Data transformation*: data is transformed using some means when messages pass from a sub-group to another.

We notice that Iolus, Cipher Sequences, Yang et al. and SIM-KM protocols affect the data path when the messages pass through a subgroup. KHIP does not transform data itself but transforms the keys with which data is encrypted and hence reduces the delays induced by the transformations at the borders of sub-groups. SIM-KM uses proxy encryption that allows to transform data without having to reveal encryption keys or clear text, and thereby the protocol is more suitable for situations where local controllers may be not trustworthy. The overall protocols succeed to mitigate the *1-affects-n* phenomenon, since they use an independent TEK per sub-group, and hence a membership change in a subgroup affects only members belonging to the same subgroup. However, reducing *1-affects-n* phenomenon is not for free: the multicast messages (or keys for KHIP) should be transformed by the Local sub-group controllers whenever they pass through a new subgroup. These transformations affect the packet delivery delays, and unfortunately this is not suitable for many applications that require real-time transmission.

D. Conclusion

In this paper we have presented a state of the art of *group key management*. We have classified existing solutions into two main categories: the *common TEK approach* and the *TEK per subgroup approach*. Throughout this paper, we refined this classification according to the common concept and techniques used by the proposed solutions. We

TABLE IV

Comparison of some decentralized group key management protocols

Protocol	Key independence	1-affects-n	Local re-key	Data transformation
Iolus	Yes	No	Yes	Yes
KHIP	Yes	No	Yes	No
Cipher Sequences	Yes	No	No	Yes
Yang et al.	Yes	No	Yes	Yes
SIM-KM	No	No	Yes	Yes
SAKM	Yes	No	Yes	Yes

illustrated each identified sub-category with relevant solutions from the literature, and we compared them against pertinent performance criteria. We showed that both proposed approaches suffer from great concerns depending on group dynamism: the *common TEK approach* suffers from the *1-affects-n* phenomenon, where a single group membership change (join or leave) results in a re-keying process that disturbs all group members to update the TEK. Moreover, centralized protocols are not scalable, and distributed ones bring new challenges such as synchronization and conflict resolution. *Time-driven re-keying protocols* attempt to reduce the *1-affects-n* phenomenon by batch re-keying, but then cannot be used with critical applications that require to take into consideration the membership change instantly. On the other hand, the *TEK per subgroup approach* reduces the *1-affects-n* problem. This is advantageous for highly dynamic multicast groups. However, this approach requires transformation of sent messages whenever they pass from a sub-group to another, and this may not be tolerated by applications that are sensitive to packet delivery delay variations. We conclude that there is not a *best* solution, but there good solutions depending on the application level requirements and features.

References

[1] K. Almeroth and M. Ammar. Collecting and modelling the join/leave behaviour of multicast group members in the Mbone. *Symposium on High Performance Distributed Computing*, 1996.

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

[3] A. Ballardie. *Scalable Multicast Key Distribution*, May 1996. RFC 1949.

[4] A. Ballardie. *Core Based Trees (CBT version 2) Multicast Routing protocol specification*, September 1997. RFC 2189.

[5] T. Ballardie, I.P. Francis, and J. Crowcroft. Core Based Trees: an Architecture for Scalable Inter-domain Multicast Routing. *ACM SIGCOMM*, pages 85–95, 1993.

[6] C. Becker and U. Wille. Communication complexity of group key distribution. *5th ACM Conference on Computerand Communications Security*, Nowember 1998.

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

[8] B. Briscoe. MARKS: Multicast key management using arbitrarily revealed key sequences. *1st International Workshop on Networked Group Communication*, November 1999.

[9] M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. *EUROCRYP'94*, LNCS(950):275–286, 1994.

[10] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast Security: A taxonomy and Efficient Constructions. *IEEE INFOCOM*, pages 708–716, March 1999.

[11] G. Chaddoud, I. Chrisment, and A. Shaff. Dynamic Group Communication Security. *6th IEEE Symposium on computers and communication*, 2001.

[12] Y. Challal, H. Bettahar, and A. Bouabdallah. SAKM: A Scalable and Adaptive Key Management Approach for Multicast Communications. *ACM SIGCOMM Computer Communications Review*, 34(2):55–70, April 2004.

[13] G. H. Chiou and W. T. Chen. Secure Broadcast using Secure Lock. *IEEE Transactions on Software Engineering*, 15(8):929–934, August 1989.

[14] H.H. Chu, L. Qiao, and K. Nahrstedt. A Secure Multicast Protocol with Copyright Protection. *ACM SIGCOMM Computer Communications Review*, 32(2):42:60, April 2002.

[15] B. DeCleene, L. Dondeti, S. Griffin, T. Hardjono, D. Kiwior, J. Kurose, D. Towsley, S. Vasudevan, and C. Zhang. Secure group communications for wireless networks. *MILCOM*, June 2001.

[16] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, November 1976.

[17] L. Dondeti, S. Mukherjee, and A. Samal. A distributed group key management scheme for secure many-to-many communication. *Technical Report PINTL-TR-207-99*, 1999.

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

[19] L.R. Dondeti, S. Mukherjee, and A. Samal. Comparison of Hierarchical Key Distribution Schemes. *IEEE Globcom Global Internet Symposium*, 1999.

[20] L.R. Dondeti, S. Mukherjee, and A. Samal. *Survey and Comparison of Secure Group Communication Protocols*, 1999. Technical Report.

[21] T. Dunigan and C. Cao. Group Key Management. *Technical Report ORNL/TM-13470*, 1998.

[22] A. Fiat and M. Naor. Broadcast Encryption. *CRYPTO'93*, LNCS(773):480–491, 1993.

[23] L. Gong and N. Shacham. Trade-offs in Routing Private Multicast Traffic. *GLOBECOM'95*, November 1995.

[24] T. Hardjono, B. Cain, and I. Monga. Intra-domain Group Key Management for Multicast Security. *IETF Internet draft*, September 2000.

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

[26] H. Harney and C. Muckenhirn. *Group Key Management Protocol (GKMP) Specification*, July 1997. RFC 2094.

[27] I. Ingemarson, D. Tang, and C. Wong. A Conference Key Distribution System. *IEEE Transactions on Information Theory*, 28(5):714–720, September 1982.

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

[29] Y. Kim, A. Perrig, and G. Tsudik. Communication-Efficient group Key Agreement. *IFIP SEC*, June 2001.

[30] D.A. McGrew and A.T. Sherman. Key Establishement in Large Dynamic Groups using One-way Function Trees. *Technical Report TR-0755*, May 1998.

[31] S. Mittra. Iolus : A Framework for Scalable Secure Multicasting. *ACM SIGCOMM*, 1997.

[32] R. Molva and A. Pannetrat. Scalable Multicast Security in dynamic groups. *6th ACM Conference on Computer and Communication Security*, November 1999.

[33] R. Mukherjee and J.W. Atwood. Proxy Encryptions for Secure Multicast Key Management. *IEEE Local Computer Networks - LCN'03*, October 2003.

[34] R. Mukherjee and J.W. Atwood. SIM-KM: Scalable Infrastructure for Multicast Key Management. *IEEE Local Computer Networks - LCN'04*, pages 335–342, November 2004.

Open Science Index, Computer and Information Engineering Vol:2, No:10, 2008 waset.org/Publication/11804

[35] R. Oppliger and A. Albanese. Distributed registration and key distribution (DiRK). *Proceedings of the 12th International Conference on Information Security IFIP SEC'96*, 1996.

[36] A. Perrig. Efficient Collaborative key Management protocols for Secure Autonomous Group Communication. *International Workshop on Cryptographic techniques and E-commerce*, 1999.

[37] A. Perrig, D. Song, and J.D. Tygar. ELK, a new protocol for Efficient Large-group Key distribution. *IEEE Security and Priavcy Symposium*, May 2001.

[38] R. Poovendram, S. Ahmed, S. Corson, and J. Baras. A Scalable Extension of Group Key Management Protocol. *2nd Annual ATRIP Conference*, pages 187–191, February 1998.

[39] S. Rafaeli and D. Hutchison. Hydra: a decentralized group key management. *11th IEEE International WETICE: Enterprise Security Workshop*, June 2002.

[40] R. Rivest. *The MD5 Message-Digest Algorithm*, April 1992. RFC 1321.

[41] O. Rodeh, K. Birman, and D. Dolev. Optimized group rekey for group communication systems. *Network and Distributed System Security*, February 2000.

[42] S. Setia, S. Koussih, S. Jajodia, and E. Harder. Kronos: A scalable group re-keying approach for secure multicast. *IEEE Symposium on Security and Privacy*, May 2000.

[43] C. Shields and J.J. Garcia-Luna-Aceves. The Ordered Core Based Tree Protocol. *IEEE INFOCOM'97*, April 1997.

[44] C. Shields and J.J. Garcia-Luna-Aceves. KHIP-A scalable protocol for secure multicast routing. *ACM SIGCOMM Computer Communication Review*, 29(4):53–64, October 1999.

[45] D. Steer, L.L. Strawczynski, W. Diffie, and M. Weiner. A Secure Audio Teleconference System. *CRYPTO'88*, 1988.

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

[47] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, , and B. Plattner. The VersaKey Framework : Versatile Group Key Management. *IEEE Journal on Selected Areas in Communications (Special Issues on Middleware)*, 17(8):1614–1631, August 1999.

[48] D. Wallner, E. Harder, and R. Agee. *Key Management for Multicast : Issues and Architecture*. National Security Agency, June 1999. RFC 2627.

[49] C. K. Wong, M. Gouda, and S. S. Lam. Secure Group Communications Using Key Graphs. *ACM SIGCOMM*, 1998.

[50] C. K. Wong, M. Gouda, and S. S. Lam. Secure Group Communications Using Key Graphs. *IEEE/ACM Transactions on Networking*, 8(1):16–30, February 2000.

[51] C.K. Wong and S.S. Lam. Keystone: A group Key Management Service. *International Conference on Telecommunication*, May 2000.

[52] Y.R. Yang, X.S. Li, X.B. Zhang, and S.S. Lam. Reliable Group Rekeying: A Performance Analysis. *TR-01-21*, June 2001.

Open Science Index, Computer and Information Engineering Vol:2, No:10, 2008 waset.org/Publication/11804