A Survey of Key Management for Secure Group Communication

SANDRO RAFAELI AND DAVID HUTCHISON

Computing Department, Lancaster University

Group communication can benefit from IP multicast to achieve scalable exchange of messages. However, there is a challenge of effectively controlling access to the transmitted data. IP multicast by itself does not provide any mechanisms for preventing nongroup members to have access to the group communication. Although encryption can be used to protect messages exchanged among group members, distributing the cryptographic keys becomes an issue. Researchers have proposed several different approaches to group key management. These approaches can be divided into three main classes: centralized group key management protocols, decentralized architectures and distributed key management protocols. The three classes are described here and an insight given to their features and goals. The area of group key management is then surveyed and proposed solutions are classified according to those characteristics.

Categories and Subject Descriptors: C.2.2 [**Computer Systems Organization**]: Network Protocols; K.6.5 [**Computing Milieux**]: Security and Protection

General Terms: Design, Management, Security

Additional Key Words and Phrases: Multicast Security, Group Key Distribution

1. INTRODUCTION

Group communication applications can use IP multicast [Deering 1989] to transmit data to all n group members using minimum resources. Efficiency is achieved because data packets need to be transmitted once and they traverse any link between two nodes only once, hence saving bandwidth. This contrasts with unicast-based group communication where the sender has to transmit n copies of the same packet.

However scalable, IP multicast does not provide mechanisms to limit the access to the data being transmitted to authorised group members only [Ballardie and Crowcroft 1995]. Any multicast-enabled host can send IGMP [Fenner 1997] messages to its neighbour router and request to join a multicast group. There is no authentication or access control enforced in this operation [Hardjono and Tsudik 2000]. The security challenge for multicast is in providing an effective method for controlling access to the group and its information that is as efficient as the underlying multicast.

A primary method of limiting access to information is through encryption and

The work presented here was done within the context of ShopAware—a research project funded by the European Union in the Framework V IST Programme.

Authors' address: D. Hutchison, Computing Department, Faculty of Applied Sciences, Engineering Building, Lancaster University, Lancaster LA1 4YR, United Kingdom; S. Rafaeli, Rua Atanasio Belmonte, 175/828, Porto Alegre, Brazil, CEP 90520-550; email: sandro@hydra.trix.net.

selective distribution of the keys used to encrypt group information. An encryption algorithm takes input data (e.g., a group message) and performs some transformations on it using a *cryptographic key*. This process generates a ciphered text. There is no easy way to recover the original message from the ciphered text other than by knowing the right key [Schneier 1996]. Applying such a technique, one can run secure multicast sessions. The messages are protected by encryption using the chosen key, which in the context of group communication is called the *group key*. Only those who know the group key are able to recover the original message.

Furthermore, the group may require that membership changes cause the group key to be refreshed. Changing the group key prevents a new member from decoding messages exchanged before it joined the group. If a new key is distributed to the group when a new member joins, the new member cannot decipher previous messages even if it has recorded earlier messages encrypted with the old key. Additionally, changing the group key prevents a leaving or expelled group member from accessing the group communication (if it keeps receiving the messages). If the key is changed as soon as a member leaves, that member will not be able to decipher group messages encrypted with the new key.

However, distributing the group key to valid members is a complex problem. Although rekeying a group before the join of a new member is trivial (send the new group key to the old group members encrypted with the old group key), rekeying the group after a member leaves is far more complicated. The old key cannot be used to distribute a new one, because the leaving member knows the old key. Therefore, a group key distributor must provide another scalable mechanism to rekey the group.

A simple scheme for rekeying a group with n members has the key distribution centre (KDC) assigning a secret key to each member of the group. In order to distribute the group key, the KDC encrypts it with each member's secret key. This operation generates a message $O(n)$ long which is then transmitted to the whole group via multicast. On receiving the message, a member can recover the group key from the appropriate segment of the message using its own secret key.

In fact, this is not so simple or scalable. Generating one copy of the group key for each member means that the KDC has to encrypt the group key n times. The size of the broadcast message has to be considered as well. For example, a message including all n copies of the encrypted group key, assuming n equal to one million and using a cryptographic algorithm with a key 56 bits long, the message would have size 10253 KB. A session where the membership changes very frequently becomes difficult to administrate. Even though the process is simple, the cost of using the simple scheme in large groups is very high.

The literature presents us with several different approaches to group key management. We can divide them into three main classes:

—*Centralized group key management protocols*. A single entity is employed for controlling the whole group, hence a group key management protocol seeks to minimize storage requirements, computational power on both client and server sides, and bandwidth utilization;

—*Decentralized architectures*. The management of a large group is divided among subgroup managers, trying to minimize the problem of concentrating the work in a single place;

—*Distributed key management protocols*. There is no explicit KDC, and the members themselves do the key generation. All members can perform access control and the generation of the key can be either contributory, meaning that all members contribute some information to generate the group key, or done by one of the members.

This survey will unfold as follows: In Section 2, we give an overview of the common goals of those classes cited above. The main contribution of this paper is described in Section 3. In Section 4, we

present and analyse the group key management protocols. The decentralised architectures are described in Section 5. Finally, the distributed key management schemes are analysed in Section 6.

2. KEY MANAGEMENT ROLE

Key management plays an important role enforcing access control on the group key (and consequently on the group communication). It supports the establishment and maintenance of key relationships between valid parties according to a security policy being enforced on the group [McDaniel et al. 1999]. It encompasses techniques and procedures that can carry out:

—*Providing member identification and authentication*. Authentication is important in order to prevent an intruder from impersonating a legitimate group member. In addition, it is important to prevent attackers from impersonating key managers. Thus, authentication mechanisms must be used to allow an entity to verify whether another entity is really what it claims to be.

—*Access control*. After a party has been identified, its join operation should be validated. Access control is performed in order to validate group members before giving them access to group communication[1] (the group key, in particular).

—*Generation, distribution and installation of key material*. It is necessary to change the key at regular intervals to safeguard its secrecy [Schneier 1996]. Additional care must be taken when choosing a new key to guarantee key independence. Each key must be completely independent from any previous used and future keys, otherwise compromised keys may reveal other keys.

The key secrecy can be extended to membership changes. When a group requires backward and forward secrecy [Kim et al. 2000], the key must be changed for every membership change. Backward secrecy is used to prevent a new member from decoding messages exchanged before it joined the group. If a new key is distributed for the group when a new member joins, it is not able to decipher previous messages even if it has recorded earlier messages encrypted with the old key. Forward secrecy is used to prevent a leaving or expelled group member to continue accessing the group's communication (if it keeps receiving the messages). If the key is changed as soon as a member leaves, that member will not be able to decipher group messages encrypted with the new key.

As multicast is being used for group transmission, it is generally assumed that multicast should also be used to rekey the group.[2] It is not reasonable to consider transmitting data using a scalable multicast communication and rekeying the members under a non-scalable peer-to-peer communication. If the group has thousands of members, sending them a new key one by one would not be efficient. Although rekeying a group before the join of a new member is trivial,[3] rekeying the group after a member leaves it is far more complicated. The old key cannot be used to distribute a new one, because the leaving member knows the old key. A group key distributor must therefore provide other mechanisms to rekey the group using multicast messages while maintaining the highest level of security possible.

3. CONTRIBUTION OF THIS ARTICLE

This article presents a survey of group key management. We present the group key management solutions proposed so far in the literature, distributing them into the classes previously introduced in Section 1. We analyze them comparatively within their respective class. This survey effectively extends and updates Moyer's survey [Moyer et al. 1999].

[1]This is because hiding information is not the only matter. If anyone can obtain the key, it does not make sense to encrypt the content at all.

[2]The term *rekey* determines the action of distributing a new key in order to replace a previous one.

[3]Send the new key to the group encrypted with the old one.

4. CENTRALIZED GROUP KEY MANAGEMENT PROTOCOLS

In a centralized system, there is only one entity controlling the whole group. The central controller does not have to rely on any auxiliary entity to perform access control and key distribution. However, with only one managing entity, the central server is a single point of failure. The entire group will be affected if there is a problem with the controller. The group privacy is dependent on the successful functioning of the single group controller; when the controller is not working, the group becomes vulnerable because the keys, which are the base for the group privacy, are not being generated/regenerated and distributed. Furthermore, the group may become too large to be managed by a single party, thus raising the issue of scalability.

The group key management protocol used in a centralized system seeks to minimise the requirements of both group members and KDC in order to augment the scalability of the group management. The efficiency of the protocol can be measured by:

—*Storage requirements*. The number of key encryption keys (KEKs) that group members and the KDC need to keep;

—*Size of messages*. Characterized by the number of bytes is a rekey message for adding and removing members. The protocol can combine unicast and multicast messages to achieve the best results. Note that the usage of unicast channels implies establishing a secure channel, hence increasing the final cost of the protocol;

—*Backwards and forward secrecy*. As described in Section 2, the capability of a protocol to provide secrecy despite changes to the group membership;

—*Collusion*. Evicted members must not be able to work together and share their individual piece of information to regain access to the group key.

Throughout this Section, k_i represents a shared key. Encryption and decryption of x with key k are respectively $\{x\}_k$ and $\{x\}_k^{-1}$. Additionally, $\{x\}_{k_i,k_j}$ means that x was encrypted with both k_i and k_j. Finally, n is the number of members.

4.1. Group Key Management Protocol

The Group Key Management Protocol (GKMP) [Harney and Muckenhirn 1997a, 1997b] enables the creation and maintenance of a group key. In this approach, the KDC helped by the first member to join the group creates a Group Key Packet (GKP) that contains a group traffic encryption key (GTEK) and a group key encryption key (GKEK). When a new member wants to join the group, the KDC sends it a copy of the GKP. When a rekey is needed, the GC generates a new GKP and encrypts it with the current GKEK ($\{GTEK\}_{GKEK}$). As all members know the GKEK, there is no solution for keeping the forward secrecy when a member leaves the group except to recreate an entirely new group without that member.

4.2. Logical Key Hierarchy

Other contributions (Wong et al. [2000] and Wallner et al. [1999]) propose the use of a Logical Key Hierarchy (LKH). In this approach, a KDC maintains a tree of keys. The nodes of the tree hold key encryption keys. The leaves of the tree correspond to group members and each leaf holds a KEK associated with that one member. Each member receives and maintains a copy of the KEK associated with its leaf and the KEKs corresponding to each node in the path from its parent leaf to the root. The key held by the root of the tree is the group key. For a balanced tree, each member stores at most $(\log_2\ n)+1$ keys, where $(\log_2\ n)$ is the height of the tree. For example, see Figure 1, member u_1 knows k_1, k_{12}, k_{14} and k.

A joining member is associated with a leaf and the leaf is included in the tree. All KEKs in the nodes from the new leaf's parent in the path to the root are compromised and should be changed (backward secrecy). A rekey message is generated containing each of the new KEKs

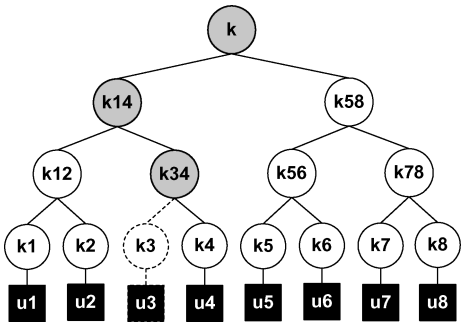

Fig. 1. KEKs affected when a member joins the tree.

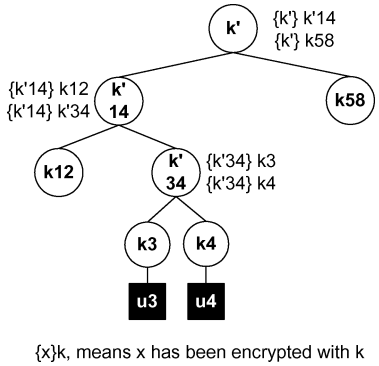

Fig. 2. Necessary encryptions when a member joins the tree in the basic LKH.

encrypted with its respective node's children KEK. The size of the message produced will be at most $2 \cdot (\log_2\ n)$ keys long. Figure 1 shows an example of the KEKs being affected. The new member u_3 receives a secret key k_3 and its leaf is attached to the node k_{34}. The KEKs held by nodes k_{34}, k_{14} and k, which are the nodes in the path from k_3 to k, are compromised. New KEKs (k'_{34}, k'_{14} and k') are generated as shown in Figure 2. Finally, the KEKs are encrypted with each of its respective node's children KEK ($\{k'_{34}\}_{k_3,k_4}$; $\{k'_{14}\}_{k_{12},k'_{34}}$; and $\{k\}_{k'_{14},k_{58}}$ (see Figure 2). The size of a rekeying message for a balanced tree has at most $2 \cdot (\log_2\ n)$ keys.

Removing a member follows a similar process. When a member leaves (or is evicted from) the group, its parent node's KEK and all KEKs held by nodes in the path to the root are compromised and should be updated (forward secrecy). A

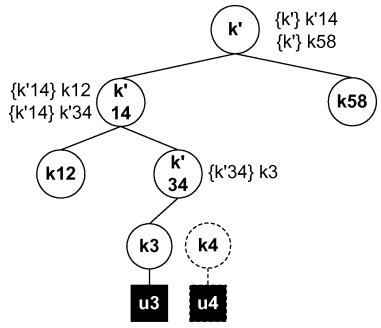

Fig. 3. Necessary encryptions when a member is removed from the basic LKH.

rekey message is generated containing each of the new KEKs encrypted with its respective node's children KEK. The exception is the parent node of the leaving member's leaf. The KEK held by this node is encrypted only with the KEK held by the remaining member's leaf. As the key held by the leaving member was not used to encrypt any new KEK, and all its known KEKs were changed, it is no longer able to access the group messages.

Figure 3 presents what happens when a member leaves. Member u_4 is leaving the group and it knows KEKs k_{34}, k_{14} and k. KEKs k'_{34}, k'_{14} and k' are updated and encrypted with each of its respective children's KEKs. An exception is made for the k'_{34}. This KEK is encrypted only with k_3, which is the key of the remaining member of n_{34}.

The algorithm proposed by Waldvogel et al. [1999] is different for joining operations. Instead of generating fresh keys and sending them to members already in the group, all keys affected by the membership change are passed through a one-way function.[4] Every member that already knew the old key can calculate the new one. Hence, the new keys do not need to be sent and every member can calculate them locally. This algorithm is known as LKH+.

[4]This scheme has been first proposed by Radia Perlman during an IETF meeting.

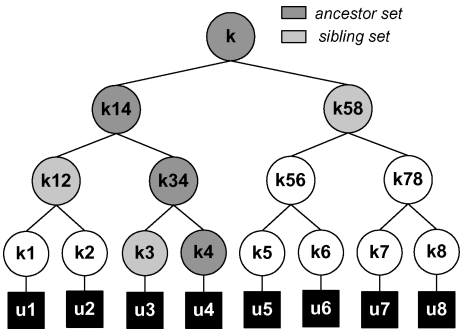

Fig. 4. Ancestor and sibling sets of member u_4.

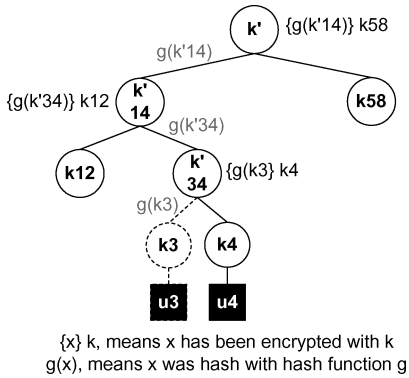

Fig. 5. Necessary encryptions when u_3 joins the tree in the improved LKH.

4.3. One-way Function Tree

An improvement in the hierarchical binary tree approach is a one-way function tree (OFT) and was proposed by McGrew and Sherman [1998]. Their scheme reduces the size of the rekeying message from $2 \cdot (\log_2\ n)$ to only $(\log_2\ n)$. Here a node's KEK is generated rather than just attributed. The KEKs held by a node's children are *blinded* using a one-way function and then mixed together using a mixing function. The result of this mixing function is the KEK held by the node. This is represented by the following formula:

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

Where *left(i)* and *right(i)* denote respectively the left and right children of node i. The function g is one-way, and f is a mixing function:

Ancestors of a node are those nodes in the path from its parent node to the root. In this article, the set of ancestor of a node is called *ancestor set* and the set of siblings of the nodes in *ancestor set* are called *sibling set* (see Figure 4). Each member receives the key (associated to its leaf node), its sibling's blinded key and the blinded keys corresponding to each node in its *sibling set*.

For a balanced tree, each member stores $log_2 n + 1$ keys. For example, in Figure 4, member u_4 knows key k_4 and blinded keys k_3^B (its sibling's blinded key) and k_{12}^B and k_{58}^B (blinded keys in u_4's *sibling set*). Applying this information to the formula 1, member u_4 is able to generate all keys in its *ancestor set* (k_{34}, k_{14} and k).

The message size reduction is achieved because in the standard scheme, when a node's key changes, the new key must be encrypted with its two children's keys, and in the OFT scheme, the blinded key changed in a node has to be encrypted only with the key of its sibling node. Figure 5 shows an example of this scheme. Member u_3 joins the group at node n_{34}. It requires keys k_{34}, k_{14} and k to be changed. The only values that must be transmitted are the *blinded* KEKs k_3^B, $k_{34}^{\prime B}$ and $k_{14}^{\prime B}$. Each is respectively encrypted with k_4, k_{12} and k_{58}. The new KEKs can be calculated by every group member: $k'_{34} = f(g(k_3), g(k_4))$, $k'_{14} = f(g(k_{12}), g(k'_{34}))$ and $k' = f(g(k'_{14}), g(k_{58}))$.

4.4. One-way Function Chain Tree

Canetti et al. [1999a] proposed a slightly different approach that achieves the same communication overhead. Their scheme uses a pseudo-random-generator [Goldreich et al. 1986] to generate the new KEKs rather than a one-way function and it is applied only on users removal. This scheme is known as the one-way function chain tree. The pseudo-random-generator, $G(x)$, doubles the size of its input (x), and there are two functions, $L(x)$ and $R(x)$, that represent the left and right halves of the output of $G(x)$ (i.e., $G(x) = L(x)R(x)$,

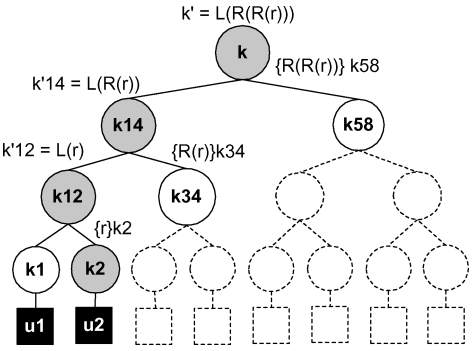

Fig. 6. New key r is attributed to leaf K2.

where $|L(x)| = |R(x)| = |x|$). When a user u leaves the group, the algorithm to rekey the tree goes as follows:

(1) a new value r_v is associated to every node v from u to the root of the tree using $r_{p(u)} = r$ for the first node and $r_{p(v)} = R(r_v)$ for all other v (where $p(v)$ denotes the parent of v).
(2) the new keys are generated as $k'_v = L(r_v)$.
(3) each $r_{p(v)}$ is encrypted with key $k_{s(v)}$ (where $s(v)$ denotes the sibling of v) and sent off.

From r_v, one can compute all keys k'_v, $k'_{p(v)}$, $k'_{p(p(v))}$ up to the root node key.

Taking into account the example of Figure 1, if u_1 leaves the group (Figure 6), nodes n_{12}, n_{14} and n_0 will be associate respectively with r, $R(r)$ and $R(R(r))$ and these values will be encrypted for n_2, n_{34} and n_{58}, with their respective KEKs (k_2, k_{34} and k_{58}). Finally, the new KEKs k_{12}, k_{14} and k will be $L(r)$, $L(R(r))$ and $L(R(R(r)))$.

Rafaeli et al. [2001] presented a variation of this scheme called the efficient hierarchical binary tree (EHBT) protocol.

4.5. Hierarchical *a*-ary Tree with Clustering

Canetti et al. [1999b] proposed the clustering of members around single leaves of a *a-ary* tree (Li et al. [2001] showed how to compute the cluster size for Canetti's model). Dividing the group with n member into clusters of size m and assigning a cluster to a unique leaf node, then we have n/m clusters. Thus, the depth of the tree is $(\log_a (n/m))$ (see Figure 7).

All members in a cluster share the same cluster KEK. Every member of the cluster is also assigned a unique key k_i, which is shared only with the KDC. The KDC uses a random seed r as an index for a pseudo-random function f_r to generate the key k_i for member i ($k_i = f_r(i)$). Hence, for each cluster, only the seed and the cluster KEK are stored. Members of the same cluster also share the set of keys in the path from the leaf node to the root, which means that every member in a cluster holds also $(\log_a (n/m)) + 1$ KEKs.

When a member leaves, the cluster holding it receives a new cluster KEK. The new KEK is encrypted with the individual KEKs of all remaining members. Hence, when a member is removed from a cluster, the KDC performs $m - 1$ encryptions to update the common cluster KEK to all remaining $m-1$ members. In addition, the KDC updates all keys in the path from the cluster leaf to the root, and every new KEK is encrypted with its respective node's children KEK. That is, when a single member is deleted, the update message has $m - 1 + a(\log_a (n/m))$ keys.

4.6. Centralized Flat Table

Waldvogel et al. [1999] extended their own solution proposing to change the hierarchical tree for a flat table (FT) with the effect of decreasing the number of keys held by the KDC. The table has one entry for the Traffic Encryption Key (TEK) and $2w$ more entries for KEKs, where w is the number of bits in the member id. There are two keys available for each bit in the member id, one associated with each possible value of the bit (Table I shows an example with $w = 4$). A member knows only the key associated with the state of its bit. In total, each member holds $w + 1$ keys. For example, a member with id 0101 knows $KEK_{0.0}$, $KEK_{1.1}$, $KEK_{2.0}$ and $KEK_{3.0}$ (see Table I).

When a member leaves the group, all keys known by it are changed and the KDC sends out a rekey message containing

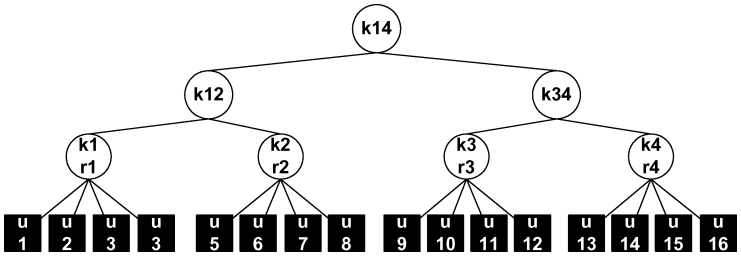

Fig. 7. Example of tree with degree 2 and cluster 4.

Table I. Flat ID Assignment

	TEK	
ID Bit #0	KEK 0.0	KEK 0.1
ID Bit #1	KEK 1.0	KEK 1.1
ID Bit #2	KEK 2.0	KEK 2.1
ID Bit #3	KEK 3.0	KEK 3.1
	Bit 0	Bit 1

two parts. The first part has the new TEK encrypted with each unchanged KEK (any member with an id with at least one single bit of difference from the leaving member's id can recover the TEK). In the second part, each of the new KEKs is encrypted with its old KEK and with the new TEK (see Table II). This way, every remaining member can update its old KEKs without gaining further knowledge about the KEKs other members had.

A group from IBM proposed a similar scheme [Chang et al. 1999] to the one from the ETH group. Although the IBM proposal has the same properties as the flat table, they presented an optimisation for the number of messages needed for rekeying the group based on *Boolean function minimisation techniques* (BFM) [Wegener 1987]. In this case, rather than always sending a message containing the new TEK encrypted by each unchanged KEK, they apply the techniques and find the smallest set of KEKs needed to rekey all remaining members.

This scheme is susceptible to collusion attacks. A set of evicted members, which have IDs with complementary bits, may combine their sets of keys to recover a valid set of keys, hence are able to have unauthorised access to group communication.

4.7. Efficient Large-Group Key

Perrig et al. [2001] proposed the Efficient Large-group Key (ELK) protocol. The ELK protocol uses a hierarchical tree and is very similar to the OFT in the sense that a parent node key is generated from its children keys. ELK uses pseudo-random functions (PRFs) to build and manipulate the keys in the hierarchical tree. A PRF uses a key K on input M of length m to generate output of length n represented by the following notation: $PRF_k^{m\to n}(\mathrm{M})$.

Using the PRF on a key, it is possible to derive four different keys to be used in different contexts: $k_i^{\alpha} = PRF_{k_i}^{n\to n}(1)$, used to generate values n_1 and n_2 (see below), $k_i^{\beta} = PRF_{k_i}^{n\to n}(2)$, used to encrypt key update messages, $k_i^{\gamma} = PRF_{k_i}^{n\to n}(3)$, used to generate hints, and $k_i^{\delta} = PRF_{k_i}^{n\to n}(4)$, used to update key nodes.

ELK employs a timely rekey, which means that the key tree is completely updated in each time interval. The group key is updated using the derivation $k'_G = PRF_{k_G^{\gamma}}^{n\to n}(0)$ (group key) and all other k'_i are derived by $k'_i = PRF_{k_i^{\gamma}}^{n\to n}(k_G)$. By deriving all keys, ELK does not require any multicast messages during a join operation (apart from unicast messages that are needed when members move around the tree due to insertion of new member nodes).

When members are deleted, new keys have to be generated for those nodes in the path from the removed node to the

Table II. Message to Exclude Member 0101

TEK		
(KEK 0.0_{new}) TEK_{new}	(TEK_{new}) KEK 0.1	ID Bit #0
(TEK_{new}) KEK 1.0	(KEK 1.1_{new}) TEK_{new}	ID Bit #1
(KEK 2.0_{new}) TEK_{new}	(TEK_{new}) KEK 2.1	ID Bit #2
(TEK_{new}) KEK 3.0	(KEK 3.1_{new}) TEK_{new}	ID Bit #3
Bit 0	Bit 1	

Table III. Comparison Table of Group Key Management Protocols

Scheme/ Feature	Secrecy		Secure Against Coll.	Message			Storage	
				join				
	back	fore		multicast	unicast	leave	KDC	member
Simple	Y	Y	Y	nK	K	nK	nK	K
GKMP	Y	N	Y	$2K$	$2K$	—	$2K$	$2K$
LKH	Y	Y	Y	$(2d-1)K$	$(d+1)K$	$I+2dK$	$(2n-1)K$	$(d+1)K$
OFT	Y	Y	Y	$(d+1)K$	$(d+1)K$	$I+(d+1)K$	$(2n-1)K$	$(d+1)K$
OFCT	Y	Y	Y	dI	$(d+1)K$	$I+(d+1)K$	$(2n-1)K$	$(d+1)K$
Clusters	Y	Y	Y	$m-1+$ $a\log_a(\frac{n}{m})$	$\log_a(\frac{n}{m})$ 2	$m-1$ $a\log_a(\frac{n}{m})$	$\frac{n}{m}\frac{a}{a-1}+$ $\frac{n}{m}$	$\log_a(\frac{n}{m})+$ 2
FT	Y	Y	N	$2IK$	$(I+1)K$	$2IK$	$(2I+1)K$	$(I+1)K$
ELK	Y	Y	Y	0	$(d+1)K$	$I+d(n_1+n_2)$	$(2n-1)K$	$(d+1)K$

root. This is accomplished by deriving the new key k_i' as $k_i' = PRF_{C_{LR}}^{n\to n}(k_i)$, with $C_{LR} = PRF_{k_{il}^{\alpha}}^{n\to n_1}(k_i) \mid PRF_{k_{ir}^{\alpha}}^{n\to n_2}(k_i)$, where k_{il} and k_{ir} are respectively left and right child key of node i. The server then multicasts $\{PRF_{k_{il}^{\alpha}}^{n\to n_1}(k_i)\}_{k_{ir}^{\beta}}$ and $\{PRF_{k_{ir}^{\alpha}}^{n\to n_2}(k_i)\}_{k_{il}^{\beta}}$.

ELK also introduces the idea of *hints*. A hint is a small piece of information, which is smaller than a key update message, that can be used to recover possible lost rekey message updates. It is provided to improve the reliability of the rekey operation and it is conveyed in data messages. Every key k_i is generated from n_1 bits from the left side child and n_2 bits from the right-side child. Moreover, using a *key verification* value, which is derived from the new key $V_{K'} = PRF_{K'}(0)$, the right child can brute-force the left child's n_1 bits (trying all combinations) and recover K_i'. The right child can do the same with the right child's n_2 bits. Normally, $n_1 < n_2$; therefore, the right child would need more computations to recover the right n_2 bits; hence, it also needs the $n_2 - n_1$ least significant bits of the right child contribution. In conclusion, a hint conveys the key verification value ($V_{K'}$) and the $n_2 - n_1$ least significant bits of the right child contribution.

4.8. Summary

In this section, we summarize and compare the properties of those protocols presented in Section 4 in a quantitative way. We focus our criteria on those characteristics presented at the beginning of Section 4. We have divided the comparison into two tables. Table III identifies those protocols that provide backward and forward secrecy; also, we show the size of messages for join and leave operations and storage requirements from both KDC and members. Table IV identifies computations required from KDC and members during join and leave operations.[5]

The notation used in Tables III and IV is described in Table III:

[5]Values written in bold are the best values for a certain column.

Table IV. Comparison Table of Group Key Management Protocols

Scheme/ Feature	Join processing		Leave processing	
	KDC	member	KDC	member
Simple	nE	D	nE	D
GKMP	$2E$	$2D$	—	—
LKH	$dH + 3dE$	$(d+1)D$	$2dE$	dD
OFT	$(d+1)H + dX + 3dE$	$(d+1)D + d(H+X)$	$d(H+X+E)$	$D + d(H+X)$
OFCT	$dH + (d+1)E$	$(d+1)D$	$d(PRG+E)$	$D + dPRG$
Cluster	$mPRG + (m + a\log_a(\frac{n}{m}))E$	$(\log_a(\frac{n}{m}) + 2)D$	$(m-1)PRG+$ $(m + a\log_a(\frac{n}{m}) - 1)E$	$(\log_a(\frac{n}{m}) + 2)D$
FT	$2IE$	ID	$(2I+1)E$	$(I+1)D$
ELK	$2(2n-1)E + 2E + (d+1)E$	$(d+1)D$	$8dE$	$dD + 5dE$

Notation for Tables III and IV.

n	number of member in the group
I	number of bits in member id
a	degree of the tree
d	height of the tree (for a balanced tree $d = \log_a n$)
m	cluster size
H	hash function
X	xor operation
E	encryption operation
D	decryption operation
K	size of a key in bits
PuK	size of a public key
PrK	size of a private key

Note that for a join operation, n includes the joining members, and for a leaving operation, n excludes the leaving member.

Tables III and IV show that every protocol achieves different results when applying different techniques. Some protocols achieve exceptionally better results than others do. However, in order to reach those performances, they demand some strong requirements and may create unbearable risks for the group security.

The protocol GKMP achieves exceptional results for storage, communication and processing on both KDC and members. However, those results are achieved by providing no method for rekeying the group after a member has left, hence seriously compromising the forward secrecy.

ELK needs no multicast message for join operations, because it employs a timed rekey, which means that the tree is completely refreshed at intervals, in spite of any membership changes. Due to the refreshing time intervals, ELK imposes some delay on the joining member before it receives the group key.

ELK has a slightly smaller multicast message for leave operations than the other protocols, because it enables the key length to match the security requirements of the group.

So far, the best solutions for a group key management protocol appear to be those using a hierarchical tree of KEKs. They achieve good overall results without compromising any aspects of security.

5. DECENTRALIZED ARCHITECTURES

In the decentralized subgroup approach, the large group is split into small subgroups. Different controllers are used to manage each subgroup, minimizing the problem of concentrating the work on a single place. In this approach, more entities are allowed to fail before the whole group is affected.

We use the following attributes to evaluate the efficiency of decentralized frameworks:

—*Key independence*. As described in Section 2, disclosure of a key must not compromise past keys;

—*Decentralized controller*. A centralizing manager should not manage the subgroup managers. The central manager raises the same issues as the centralized systems seen in Section 4, namely

if the centralizing manager is unavailable, the whole group is compromised;

—*Local rekey*. Membership changes in a subgroup should be treated locally, which means that rekey of a subgroup should not affect the whole group. This is also known as the *1-affects-n* problem [Mittra 1997];

—*Keys vs. data*. The data path should be independent of the key management path, which means that rekeying the subgroup should not impose any interference or delays to the data communication;

—*Rekey per membership*. Related to backward and forward secrecy;

—*Type of communication*. Groups with a single data source are said to use *1-to-n* communication, and groups with several or all members being able to transmit are characterized by using *n-to-n* communication.

5.1. Scalable Multicast Key Distribution

RFC1949 [Ballardie 1996] proposes a scheme to use the trees built by the Core Based Tree (CBT) multicast routing protocol to deliver keys to a multicast group. Any router in the path of a joining member from its location to the primary core can authenticate the member since the router is authenticated with the primary core. This scheme requires some modifications to the IGMP[6] [Fenner 1997] and assumes that CBT is deployed. Furthermore, there is no solution for forward secrecy other than to recreate an entirely new group without the leaving members.

5.2. Iolus

Mittra proposes Iolus [Mittra 1997], a framework with a hierarchy of agents that splits the large group into small subgroups. A Group Security Agent (GSA) manages each subgroup. The GSAs are also grouped in a top-level group that is managed by a Group Security Controller (see Figure 8).

[6]The description of the modifications required can be found in the Appendix B of RFC1449.

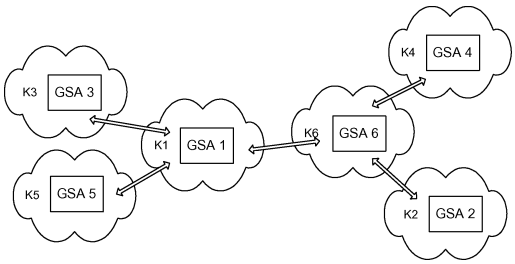

Fig. 8. Iolus hierarchy.

Iolus uses independent keys for each subgroup and the absence of a general group key means membership changes in a subgroup are treated locally. It means that changes that affect a subgroup are not reflected in other subgroups. In addition, the absence of a central controller contributes to the fault-tolerance of the system. If a subgroup controller (namely GSA) fails, only its subgroup is affected.

Although Iolus is scalable, it has the drawback of affecting the data path. This occurs in the sense that there is a need for translating the data that goes from one subgroup, and thereby one key, to another. This becomes even more problematic when it is taken into account that the GSA has to manage the subgroup and perform the translations needed. The GSA may thus become a bottleneck.

5.3. Dual-Encryption Protocol

In order to solve the problem of trusting third parties, Dondeti et al. [1999b] proposed a dual-encryption protocol (DEP). In their work, they suggest a hierarchical subgrouping of the members where a subgroup manager (SGM) controls each subgroup. There are three kinds of KEKs and one Data Encryption Key (DEK). KEK_{i1} is shared between a SGM_i and its subgroup members. KEK_{i2} is shared between the Group Controller (GC) and the members of subgroup i, excluding SGM_i. Finally, GC shares KEK_{i3} with SGM_i. In order to distribute the DEK to the members, the GC generates and transmits a package containing the DEK encrypted with KEK_{i2} and encrypted again with KEK_{i3} (SGMs' KEK). On receiving the package, SGM_i

decrypts its part of the message using KEK_{i3} and recovers the DEK encrypted with its subgroup KEK (KEK_{i2}), which is not known by the SGM_i. SGM_i encrypts this encrypted DEK using KEK_{i1} shared with its subgroup members and sends it out to subgroup i. Each member of subgroup i decrypts the message using KEK_{i1} and then, decrypting the message using KEK_{i2} (shared with GC), recovers DEK. The DEK cannot be recovered for 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 subgroup i's membership changes, the SGM_i changes KEK_{i1} and sends it to its members. Future DEK changes cannot be accessed for members of subgroup i that did not received the new KEK_{i1}. However, evicted members that did not receive KEK_{i1} can still access the group communication until the DEK is changed (note that it has not been changed with KEK_{i1}), and this compromises forward secrecy.

Weiler [2001] proposed SEMSOMM, a system with similar properties to those of DEP, namely the dual-encryption technique. However, SEMSOMM uses the dual-encryption technique to encrypt the group communication rather than the group key. This achieves forward secrecy from the moment that a subgroup key is changed (KEK_{i1} from DEP). However, because the intermediate nodes have to translate messages sent by the sender, SEMSOMM presents the same limitation of Iolus affecting the data path.

5.4. MARKS

In MARKS, Briscoe [1999] suggests slicing the time-length to be protected (such as the transmission time of a TV program) 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. The internal nodes of the tree are also called seeds. A blinding function, such as MD5 [Rivest 1992], is used on the seed

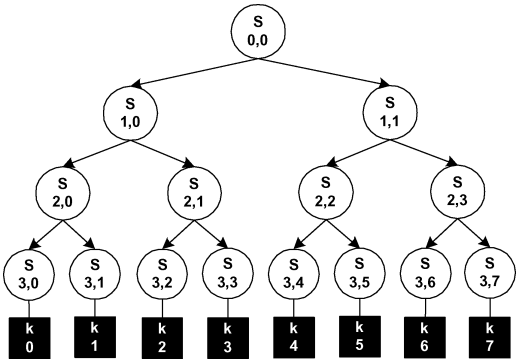

Fig. 9. Binary hash tree.

to create the tree in the following way:

(1) first, the depth D of the tree is chosen. The depth, D, defines the total number (N) of keys ($N = 2^D$).
(2) then, the root seed, $S_{0,0}$, is randomly chosen. In $S_{i,j}$, i represents the depth of the seed in the tree, and j is the number of that key in level i.
(3) after that, two intermediate seeds (*left* and *right*) are generated. The *left* node is generated by *shifting* $S_{0,0}$ one bit to the left and applying the blinding function on it ($S1, 0 = b(ls(S_{0,0}))$). The *right* node is generated by *shifting* $S_{0,0}$ one bit to the right and applying the blinding function on it ($S1, 1 = b(rs(S_{0,0}))$).
(4) the same algorithm is applied to the following levels until the expected depth is reached.

Users willing to access the group communication receive the seeds need to generate the keys required. For example, Figure 9 shows a binary hash tree of depth 3. if a user wants to participate in the group from time 3 to 7, it would be necessary to have only two seeds: $S_{3,3}$, as K3, and $S_{1,1}$, to generate K_4 till K_7.

This system cannot be used in situations when a membership change requires change of the group key, since the keys are changed as a function of the time.

5.5. Cipher Sequences

Molva presented a framework for multicast security [Molva and Pannetrat 1999]

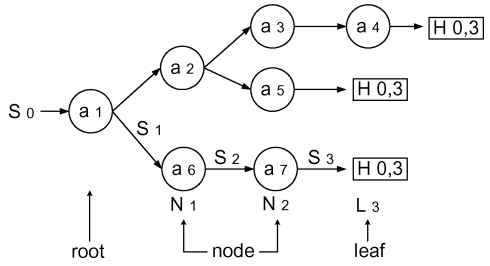

Fig. 10. An RPS tree.

that is based on Cipher Sequences (CS). A function $f(S, a)$ is called Cipher Group (CG) if it has the following characteristics: there is a sequence of n elements, such as $a_{1\leq i\leq n}$; and there is a sequence of $n+1$ elements, such as $S_{0\leq i\leq n}$, where $S_i = f(S_{i-1}, a_i)$, for $i > 0$ and S_0 is the initial value; and for every couple (i, j), where $i < j$, there exists a function $h_{i,j}$ such as $S_i = h_{i,j}(j)$. The multicast group is placed in a tree, where the *root* of the tree is the multicast source, the *leaves* of the tree are group members and the internal *nodes* of the tree are intermediate elements in the multicast communication.

Now let S_0 be the information to be multicast and let every node N_i be assigned a value $a_i > 1$. Every node N_i can perform function f, so that when 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 in the tree (note that the children can be other nodes or leaves). The leaves were 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 recovers the original data.

Figure 10 shows an example of Molva's scheme that may be described as:

—the root calculates $S_1 = f(S_0, a_1)$ and sends S_1 to N_1;
—node N_1 calculates $S_2 = f(S_1, a_6)$ and sends S_2 to N_2;
—node N_2 calculates $S_3 = f(S_2, a_7)$ and sends S_3 to leaf L_3;
—leaf L_3 calculates $S_0 = h_{0,3}(S_3)$ and recovers the original data (S_0).

A leaf may be composed of several group members and all members in the same leaf share the same function $h_{0,n}$. When a membership change occurs in a leaf, the node N_n receives a new value a'_n and all members in the leaf receive a new function $h'_{0,n}$. Naturally, if the membership change occurred because of member removal, the removed member will not receive the new $h'_{0,n}$, thus will not be able to recover S_0.

5.6. Kronos

Setia et al. [2000] proposed Kronos. It is an approach driven by periodic rekeys rather than membership changes, which means a new group key is generated after a certain period of time, disregarding whether any member has joined, left or been ejected from the group. Although Kronos can be used within a distributed framework such as IGKMP, it works differently because the DKD does not directly generate the group key. Instead, each AKD independently generates the same group key and transmits it to its members at the end of the predetermined period.

For this scheme to work, the AKDs first have to have their clocks synchronized and they have to agree on a rekey period. The authors suggest using the Network Time Protocol (NTP) [Mills 1992] for clock synchronization. Second, the AKDs also have to agree on two secret factors, namely K and R_0, where R_0 is an initial value and K is a master key. The AKDs can receive the secret factors from DKD via a secure channel. To generate the group key, with name R_1, the AKDs apply an encryption algorithm, E, to R_0 using K as the secret key ($R_1 = E_K(R_0)$). The same algorithm applies for the next key generations: $R_{i+1} = E_K(R_i), i >= 0$. Although Kronos does not use a central controller and the subgroup controllers can generate the new keys independently, which makes the system fault-tolerant, it compromises the group security because it generates the new key based on the previous one. If one key is disclosed, then it compromises all following keys.

5.7. Intra-Domain Group Key Management

DeCleene et al. [2001] present an intra-region group key management protocol

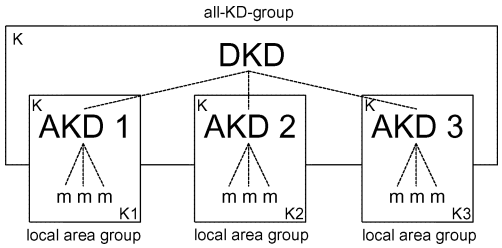

Fig. 11. Intra-Domain Group Key Management Elements.

(IGKMP). IGKMP is divided in administratively scoped areas [Meyer 1998]. There is a Domain Key Distributor (DKD) and many Area Key Distributors (AKD). Each AKD is responsible for one area. The group key is generated by the DKD and is propagated to the members through the AKDs. The key managers (DKD and AKD) are placed in a multicast group, named All-KD-group (see Figure 11). The All-KD-group is used by the DKD to transmit the rekey messages to the AKDs. All areas in the domain use the same group key. Therefore, data packets do not need to be translated when passing from one area to another. In addition, the DKD does not need to keep track of all group members, it only has to keep track of the AKDs. Nevertheless, since Intra-domain protocol employs a central controller (namely, DKD) for controlling the subgroup controllers (namely, AKDs), it presents the undesirable feature of allowing the whole group to be disrupted if the DKD is compromised. Moreover, if an AKD is unavailable no members in that area are able to access the group communication, since they will not be able to access AKDs from other areas.

5.8. Hydra

Rafaeli and Hutchison [2002] proposed Hydra. In Hydra, the large group is divided into smaller subgroups, and a server called the Hydra Server (HS) controls each subgroup. Hydra is a decentralized group key management scheme without a central subgroup controller. If a membership change takes place at HS_i, and a new key must be generated, it can generate the new group key and send this key to the other HS_j involved in that session. The case when one or more HSs become unavailable will not cause a problem for the remaining HSs. In order to have the group key distributed to all HSs a synchronized group key distribution protocol (SGKDP) is employed. The SGKDP protocol ensures that only a single valid HS is generating the new group key at every given time.

5.9. Summary

In this section, we summarize and compare the properties of those architectures presented at the beginning of Section 5. We focus our criteria on those characteriztics presented in Section 5. Table V shows a summary of the properties that should be provided by an architecture for group key management. A value written in bold is the best value for a certain column.

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.

Although DEP provides key independence, it uses a timed rekey, which causes delays to change the group key after a member has left, thus that member can still access the group communication during that period of time. The same occurs with Kronos. In SMKD, the lack of rekey after a membership change is even more serious. SMKD uses the same technique as the GKMP (seen in Section 4), namely using a unique KEK to encrypt the next group key, which has the undesirable effect of not allowing removal of members.

Limiting the rekey operation to the subgroup where the membership change has occurred avoids the need for making all members change to a new key. It minimizes the number of control messages. However, the only solutions proposed so far employ the use of different keys per subgroup. This solution requires direct interference with the data path by means of translating messages transiting from a subgroup to another due to the different

Table V. Comparison Table of Decentralized Frameworks

Scheme/ Feature	key indep.	Decent.		Local rekey	Keys vs. data	Rekey	Fault tolerant	Comm. Type
		Mngmt	KDC					
SMKD	Y	Y	Y	N	Y	N	N	Both
Iolus	Y	Y	Y	Y	N	Y	Y	*1-to-n*
DEP	Y	N	Y	N	Y	N	N	Both
CS	Y	N	N	N	Y	Y	N	*1-to-n*
MARKS	N	Y	—	N	Y	N	Y	Both
Kronos	N	Y	Y	N	Y	N	Y	Both
IGKMP	Y	Y	N	N	Y	Y	N	Both
Hydra	Y	Y	Y	N	Y	Y	Y	Both

keys being employed by those subgroups. Both Iolus and SEMSOMM solve the one-affects-all problem, but require the data to be translated.

Finally, we look at the decentralized controller feature. There are architectures that, athough using subgroup controllers, rely on a central subgroup controller to either control access to the group or generate the group key. The former situation clearly spoils the scalability of the system, because the central controller has to be contacted to verify the membership validity of every member in the group. The latter situation may create a hazard for the group security if the central controller is incapable of generating the required keys.

In both CS and DEP architectures, the central controller has to be contacted during join operations in order to authorise it. However, in the DEP case, the subgroup controllers do not have to be contacted again when members leave. Moreover, in CS, DEP, SEMSOMM, SMKD and IGKMP frameworks, if the central controller is unavailable the whole group is affected.

6. DISTRIBUTED KEY MANAGEMENT

The distributed key management approach is characterized by having no group controller. The group key can be either generated in a contributory fashion, where all members contribute their own share to computation of the group key, or generated by one member. In the latter case, although it is fault-tolerant, it may not be safe to leave any member to generate new keys since key generation requires secure mechanisms, such as random number generators, that may not be available to all members. Moreover, in most contributory protocols (apart from tree-based approaches), processing time and communication requirements increase linearly in term of the number of members. Additionally, contributory protocols require each user to be aware of the group membership list to make sure that the protocols are robust.

We use the following attributes to evaluate the efficiency of distributed key management protocols:

—*Number of rounds*. The protocol should try to minimize the number of iterations among the members to reduce processing and communication requirements;

—*Number of messages*. The overhead introduced by every message exchanged between members produces unbearable delays as the group grows. Therefore, the protocol should require a minimum number of messages;

—*Processing during setup*. Computations needed during setup time. Setting up the group requires most of the computation involved in maintaining the group, because all members need to be contacted;

—*DH key*. Identify whether the protocol uses Diffie–Hellman (DH) [Diffie and Hellman 1976] to generate the keys. The use of DH to generate the group key implies that the group key is generated in a contributory fashion.

6.1. Burmester and Desmedt Protocol

Burmester and Desmedt [1994] 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 key $k = Z_{i-1}^{nr_i} \cdot X_i^{n-i} \cdot X_{i+1}^{n-2} \cdots X_{i-2} mod\ p$.

The BD 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.

6.2. Group Diffie–Hellman Key Exchange

Group Diffie–Hellman key exchange [Steiner et al. 1996] is an extension for the DH key agreement protocol that supports group operations. The DH protocol is used for two parties to agree on a common key. In this protocol, instead of two entities, the group may have n members. The group agrees on a pair of primes (q and α) and starts calculating in a distributive fashion the intermediate values. The first member calculates the first value (α^{x_1}) and passes it to the next member. Each subsequent member receives the set of intermediary values and raises them using its own secret number generating a new set. A set generated by the ith 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 (n) can easily calculate k from the *cardinal* value ($k = \alpha^{x_1 \cdots x_n}\ mod\ q$). Member n raises all intermediate values to its secret value and multicasts the whole set. Each group member extracts its respective intermediate value and calculates k. The setup time is linear (in terms of n) since all members must contribute to generating the group key. Therefore, the size of the message increases as the sequence is reaching the last members and more intermediate values are necessary. With that, the number of exponential operations also increases.

6.3. Octopus Protocol

Becker and Wille [1998] proposed the octopus protocol. This protocol is also based on DH key exchange protocol. In Octopus, the large group (composed by n members) is split into four subgroups ($n/4$ members each). Each subgroup agrees internally on an intermediary DH value ($I_{subgroup} = \alpha^{u_1 \cdots u_{n/4}}$, 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 subgroup is responsible for collecting contributions from all its subgroup 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 $\alpha^{I_c \cdot I_d}$. 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 and D send to their respective subgroups $\alpha^{\frac{I_a \cdot I_b \cdot I_c \cdot I_d}{u_i}}$, where $i = 1 \cdots (n-4)/4$, and all members of the group are capable of calculating the group key.

6.4. Conference Key Agreement

Boyd [1997] 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: $K = f(N_1, h(N_2), \ldots, h(N_n))$, where f is the combining function (a MAC [Schneier 1996]), 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 $n-1$ group 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.

6.5. Distributed Logical Key Hierarchy

A distributed approach based on the logical key hierarchy is suggested by Rodeh et al. [2000]. In this approach, the GC 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∪R) receive the new key.

Figure 12 shows an example of the distributed LKH tree:

(1) Members 1 and 2 agree on subtree key k_{12}
Members 3 and 4 agree on subtree key k_{34}
Members 5 and 6 agree on subtree key k_{56}
Members 7 and 8 agree on subtree key k_{78}
(2) Members 1, 2 and 3, 4 agree on subtree key k_{14}
Members 5, 6 and 7, 8 agree on mutual key k_{58}
(3) Members 1, 2, 3, 4 and 5, 6, 7, 8 agree on mutual key k_{18}

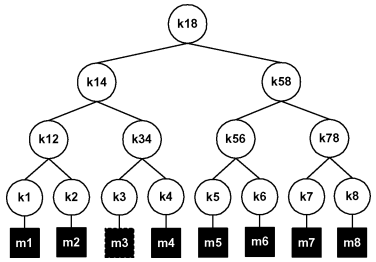

Fig. 12. LKH tree.

In this case, the algorithm takes three rounds and each member stores three keys. This algorithm takes $\log_2 n$ rounds to complete, with each member storing $\log_2 n$ keys.

6.6. Distributed One-way Function Tree

Another approach using logical key hierarchy in a distributed fashion was proposed by Dondeti et al. [1999a]. This protocol uses the one-way function tree proposed by McGrew and Sherman. The difference is that there is no centralized controlling entity. Moreover, every group member is trusted with access control and key generation. A member is responsible for generating its own key and sending the blinded version of this key to its sibling. As in the centralized one-way function tree, every member knows all keys in the path from its node to the root node and all blinded keys from the sibling node to the nodes in the path to the root.

6.7. Diffie–Hellman Logical Key Hierarchy

Perrig [1999] and Kim et al. [2000] also used a logical key hierarchy to minimise the number of key held by group members. The difference here is that group members generate the keys in the upper levels using the Diffie–Hellman algorithm rather than using a one-way function. The key of each node is generated from its two children ($k = \alpha^{k_l k_r} mod\ p$). Using

Table VI. Comparison Table of Distributed Key Management Protocols

Scheme/ Feature	Nro. rounds	No. messages		DH key	Setup	
		Multicast	Unicast		Leader	Others
BD	3	$2n$	0	N	—	$(n+1)Ex$
G-DH	n	n	$n-1$	Y	—	$(i+1)Ex$
Octopus	$2(n-4)/4+2$	0	$3n-4$	Y	$(2(n-4)/4+2)Ex$	$4Ex$
CKA	3	n	$n-1$	N	$H+(n-1)(E+H)+M$	$D+nH+M$
D LKH	3	1	n	N	$\log_2 nEx$	$\log_2 nD$
D OFT	$\log_2 n$	0	$2\log_2 n$	N	—	$\log_2 n(H+X)$
DH-LKH	$\log_2 n$	$\log_2 n$	0	Y	—	$(\log_2 n+1)Ex$
DFT	n	0	$2n-1$	N	(i-1)E	iD

Figure 12 as an example, keys $k_{12}=\alpha^{k_1k_2}$ $mod\ p$, $k_{34}=\alpha^{k_3k_4} mod\ p$, $k_{56}=\alpha^{k_5k_6} mod\ p$ and $k_{78}=\alpha^{k_7k_8} mod\ p$. Following the algorithm, keys $k_{14}=\alpha^{k_{12}k_{34}} mod\ p$ and $k_{58}=\alpha^{k_{56}k_{78}} mod\ p$, and $k_{18}=\alpha^{k_{14}k_{58}} mod\ p$ (or $k_{18}=\alpha^{\alpha^{\alpha^{k_1k_2}\alpha^{k_3k_4}}\alpha^{\alpha^{k_5k_6}\alpha^{k_7k_8}}} mod\ p$).

6.8. Distributed Flat Table

Waldvogel et al. [1999] extends further its solution, proposing to use the flat table in a distributed (DFT) fashion with no Group Controller. In this scheme, no member knows all the keys at any time. Each member knows only the KEKs that it is entitled to. The distributed management has an inconvenience, namely that a joining member is obliged to contact a group of members to get all the keys needed. Furthermore, since many members could be changing the same key at the same time, there could be serious delays in synchronizing the keys.

6.9. Summary

In this section, we summarize and compare the properties of those distributed key management protocols presented in Section 6. We focus our criteria on those characteristics presented at the beginning of Section 6. Table VI shows a summary of the properties that we analyze on a key management protocol. A Value written in bold is the best value for a certain column.

The notation used in Table VI is described in Table VI:

The elements that appear in Table VI mean:

Notation for Table VI.

n	number of members in the group
i	index of member
I	index size
H	hash function execution
X	xor operation
E	encryption
D	decryption
Ex	exponentiation
M	MAC

The DH key column has no values in bold, because the use of DH to generate the group key does not imply any direct benefit. DH key generation is only a technique that can be used for generating the group key in a contributory fashion.

Those protocols that do not rely on a group leader during setup time 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 setup, it will not affect the whole group. In those protocols with a group leader, a leader failure is fatal for creating the group and the operation has to be restarted from scratch.

Both CKA and DH-LKH have a fixed number of rounds, which means that the number of interactions among the members is independent of the number of members in the group. The operations in these two protocols can be done in parallel, minimizing the amount of time needed to create the group key. However, they both rely on a leader to collect contributions

from all other members and then broadcast them to the others, and as we have discussed before, if the leader fails, the whole group is affected and the whole operation has to be restarted.

Protocols like D-OFT and DH-LKH do not have a leader during setup time and all members compute the intermediary values independently. At the final round, all members compute the same group key. Any member failure can be ignored, because it does not block the other members.

7. CONCLUSION

In this article, we presented a survey in the secure group communication area, particularly regarding the secure distribution and refreshment of keying material. We reviewed several proposals, placing them into three main classes: group key management protocols, which try to minimise the requirements of KDC and group members; decentralized architectures, which divide large group in smaller subgroups in order to make the management more scalable; and finally, the distributed key management protocols, which gives all members the same responsibilities. Every class has its particularities, presenting different features, requirements and goals.

Our analysis made it clear there is no unique solution that can achieve all requirements. While centralized key management schemes are easy to implement, they tend to impose an overhead on a single entity. Protocols based on hierarchical subgrouping are relatively harder to implement and raise other issues, such as interfering with the data path or imposing security hazards on the group. Distributed key management, by design, is simply not scalable. Additionally, the best solution for a particular application may not be best for another, hence it is important to understand fully the requirements of the application before selecting a security solution.

A solution for secure group communication should complement a multicast application rather than drive its implementation. Primarily, the usage of security mechanism for secure group communication should be made transparent to the user and it should also work well with other protocols.

REFERENCES

BALLARDIE, A. 1996. *Scalable Multicast Key Distribution*. RFC 1949.

BALLARDIE, A. AND CROWCROFT, J. 1995. Multicast specific security threats and counter-measures. In *Proceedings of the Symposium on Network and Distributed System Security*. (San Diego, Calif., Feb.).

BECKER, C. AND WILLE, U. 1998. Communication complexity of group key distribution. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*. (San Francisco, Calif., Nov.). ACM, New York.

BOYD, C. 1997. On key agreement and conference key agreement. In *Proceedings of the Information Security and Privacy: Australasian Conference*. Lecture Notes in Computer Science, vol. 1270. Springer-Verlag, New York, 294–302.

BRISCOE, B. 1999. MARKS: Multicast key management using arbitrarily revealed key sequences. In *Proceedings of the 1st International Workshop on Networked Group Communication*. (Pisa, Italy, Nov.).

BURMESTER, M. AND DESMEDT, Y. 1994. A secure and efficient conference key distribution system (extended abstract). In *Advances in Cryptology—EUROCRYPT 94*, A. D. Santis, Ed., Lecture Notes in Computer Science, vol. 950. Springer-Verlag, New York, pp. 275–286.

CANETTI, R., GARAY, J., ITKIS, G., MICCIANCIO, D., NAOR, M., AND PINKAS, B. 1999a. Multicast Security: A Taxonomy and Some Efficient Constructions. In *Proceedings of the IEEE INFOCOM*. Vol. 2. (New Yok, N.Y., Mar.). 708–716.

CANETTI, R., MALKIN, T., AND NISSIM, K. 1999b. Efficient communication-storage tradeoffs for multicast encryption. In *Advances in Cryptology—EUROCRYPT '99*, J. Stem, Ed. Lectures Notes in Computer Science, vol. 1599. Springer-Verlag, New York, pp. 459–474.

CHANG, I., ENGEL, R., KANDLUR, D., PENDARAKIS, D., AND SAHA, D. 1999. Key management for secure internet multicast using boolean function minimization techniques. In *IEEE INFOCOM*. Vol. 2. (New York, March 1999), 689–698.

DECLEENE, B., DONDETI, L., GRIFFIN, S., HARDJONO, T., KIWIOR, D., KUROSE, J., TOWSLEY, D., VASUDEVAN, S., AND ZHANG, C. 2001. Secure group communications for wireless networks. In *Proceedings of the MILCOM*. (June).

DEERING, S. 1989. *Host Extensions for IP Multicasting*. RFC 1112.

DIFFIE, W. AND HELLMAN, M. E. 1976. New directions in cryptography. *IEEE Trans. Inf. Theory IT-22*, 6 (Nov.), 644–654.

DONDETI, L., MUKHERJEE, S., AND SAMAL, A. 1999a. A distributed group key management scheme for secure many-to-many communication. Tech. Rep. PINTL-TR-207-99, Department of Computer Science, University of Maryland.

DONDETI, L., MUKHERJEE, S., AND SAMAL, A. 1999b. Scalable secure one-to-many group communication using dual encryption. *Comput. Commun. 23*, 17 (Nov.), 1681–1701.

FENNER, W. 1997. *Internet Group Management Protocol*, Version 2. RFC 2236.

GOLDREICH, O., GOLDWASSER, S., AND MICALI, S. 1986. How to construct random functions. *J. ACM 33*, 4 (Oct.), 792–807.

HARDJONO, T. AND TSUDIK, G. 2000. IP multicast security: Issues and directions. *Ann. Telecom.* 324–340.

HARNEY, H. AND MUCKENHIRN, C. 1997a. *Group Key Management Protocol (GKMP) Specification*. RFC 2093.

HARNEY, H. AND MUCKENHIRN, C. 1997b. *Group Key Management Protocol (GKMP) Architecture*. RFC 2094.

KIM, Y., PERRIG, A., AND TSUDIK, G. 2000. Simple and fault-tolerant key agreement for dynamic collaborative groups. In *Proceedings of the 7th ACM Conference in Computer and Communication Security*, (Athens, Greece Nov.). (S. Jajodia and P. Samarati, Eds.), pp. 235–241.

LI, M., POOVENDRAN, R., AND BERENSTEIN, C. 2001. Optimization of key storage for secure. In *Proceedings of the 35th Annual Conference on Information Sciences and Systems (CISS)*. (John Hopkins, Mar.).

MCDANIEL, P., PRAKASH, A., AND HONEYMAN, P. 1999. Antigone: A flexible framework for secure group communication. In *Proceedings of the 8th USENIX Security Symposium*. (Washington, D.C. Aug.). 99–114.

MCGREW, D. A. AND SHERMAN, A. T. 1998. Key establishment in large dynamic groups using one-way function trees. Tech. Rep. No. 0755 (May), TIS Labs at Network Associates, Inc., Glenwood, Md.

MEYER, D. 1998. *Administratively Scoped IP Multicast*. RFC 2365.

MILLS, D. L. 1992. *Network Time Protocol (Version 3) Specification, Implementation and Analysis*. RFC 1305.

MITTRA, S. 1997. Iolus: A framework for scalable secure multicasting. In *Proceedings of the ACM SIGCOMM*. Vol. 27, 4 (New York, Sept.) ACM, New York, pp. 277–288.

MOLVA, R. AND PANNETRAT, A. 1999. Scalable multicast security in dynamic groups. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*. (Singapore, Nov.). ACM, New York, 101–112.

MOYER, M. J., RAO, J. R., AND ROHATGI, P. 1999. A survey of security issues in multcast communications. *IEEE Netw. Mag. 13*, 6 (Nov./Dec.), 12–23.

PERRIG, A. 1999. Efficient collaborative key management protocols for secure autonomous group communication. In *Proceedings of the International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC'99)*. (Hong Kong, China, July). M. Blum and C H Lee, Eds. City University of Hong Kong Press, Hong Kong, China, pp. 192–202.

PERRIG, A., SONG, D., AND TYGAR, J. D. 2001. ELK, A new protocol for efficient large-group key distribution. In *Proceedings of the IEEE Symposium on Security and Privacy*. (Oakland, Calif., May). IEEE Computer Society Press, Los Alamitos, Calif.

RAFAELI, S. AND HUTCHISON, D. 2002. Hydra: A decentralised group key management. In *Proceedings of the 11th IEEE International WETICE: Enterprise Security Workshop*, A. Jacobs, Ed. (Pittsburgh, Pa., June). IEEE Computer Society Press, Los Alamitos, Calif.

RAFAELI, S., MATHY, L., AND HUTCHISON, D. 2001. EHBT: An efficient protocol for group key management. In *Proceedings of the 3rd International Workshop on Networked Group Communications*. (London, U.K., Nov.). Lecture Notes in Computer Science, vol. 2233. Springer-Verlag, New York, pp. 159–171. Springer-Verlag.

RIVEST, R. 1992. *The MD5 Message-Digest Algorithm*. RFC 1321.

RODEH, O., BIRMAN, K., AND DOLEV, D. 2000. Optimized group rekey for group communication systems. In *Network and Distributed System Security*. (San Diego, Calif., Feb.).

SCHNEIER, B. 1996. *Applied Cryptography Second Edition: protocols, algorithms, and source code in C*. Wiley, New York. ISBN 0-471-11709-9.

SETIA, S., KOUSSIH, S., AND JAJODIA, S. 2000. Kronos: A scalable group re-keying approach for secure multicast. In *Proceedings of the IEEE Symposium on Security and Privacy*. (Oakland Calif., May). IEEE Computer Society Press, Los Alamitos, Calif.

STEINER, M., TSUDIK, G., AND WAIDNER, M. 1996. Diffie-Hellman key distribution extended to group communication. In *SIGSAC Proceedings of the 3rd ACM Conference on Computer and Communications Security*. (New Delhi, India, Mar.). ACM, New York, pp. 31–37.

WALDVOGEL, M., CARONNI, G., SUN, D., WEILER, N., AND PLATTNER, B. 1999. The VersaKey framework: Versatile group key management. *IEEE J. Sel. Areas Commun. (Special Issue on Middleware) 17*, 9 (Aug.), 1614–1631.

WALLNER, D., HARDER, E., AND AGEE, R. 1999. *Key Management for Multicast: Issues and Architectures*. RFC 2627.

WEGENER, I. 1987. *The Complexity of Boolean Functions*. Wiley, New York. ISBN: 0-471-91555-6.

WEILER, N. 2001. SEMSOMM—A scalable multiple encryption scheme for one-to-many multicast. In *Proceedings of the 10th IEEE International WETICE Enterprises Security Workshop*, (Cambridge, Mass., June). IEEE Computer Society Press, Los Alamitos, Calif.

WONG, C. K., GOUDA, M. G., AND LAM, S. S. 2000. Secure group communications using key graphs. *IEEE/ACM Trans. Netw. 8*, 1 (Feb.), 16–30.

Received July 2001; accepted June 2003