Scalable Group Key Management for Secure Multicast: A Taxonomy and New Directions

Sencun Zhu[1] Sushil Jajodia[2]
[1] Department of Computer Science and Engineering & School of Information Sciences and Technology
The Pennsylvania State University, University Park, PA 16801
[2] Center for Secure Information Systems, George Mason University, Fairfax, VA 22030
szhu@cse.psu.edu, jajodia@gmu.edu

Abstract

Multicast is an efficient technique to distribute data to a large group of users. To prevent disclosure of distributed data to unauthorized users, multicast-based applications require confidential group communication, which is achieved by encrypting data with a group-wide shared key. The group key must be updated and redistributed to all authorized users in a secure and reliable fashion whenever a user joins or leaves the group. Thus, group key management becomes a very challenging issue for large and dynamic groups with unreliable channels. We give an overview of the approaches that have been recently proposed to address the group key management issue, show the research trends, and finally discuss several new research directions.

1 Introduction

Many multicast-based applications (e.g., pay-per-view, online auction, and teleconferencing) require a secure communication model to prevent disclosure of distributed data to unauthorized users. One solution for achieving this goal is to let all members in a group share a key that is used for encrypting data. To provide backward and forward confidentiality [23] (i.e., a new member should not be allowed to decrypt the earlier communication and a revoked user should not be able to decrypt the future communication) this shared group key must be updated for every membership change and redistributed to all authorized members in a secure, reliable, and timely fashion. This process is referred to as *group rekeying*.

A group rekeying operation usually involves two phases. The first phase deals with the key encoding problem. To prevent passive eavesdropping attacks, a new group key must be encrypted by some key encryption keys (KEKs) before its distribution. The goal of a key encoding algorithm is to minimize the number of encrypted keys that have to be distributed. The second phase deals with the key distribution problem, i.e., distributing the encryptions output from a key encoding algorithm to group members reliably even in the presence of packet losses. The scalability of group rekeying is determined by the efficiency of both key encoding and key distribution mechanisms.

A simple approach for group rekeying is one based on unicast; that is, the key server sends the group key to each member individually and securely. Despite its simplicity, this approach is not scalable because its communication cost increases linearly with the group size. Specifically, for a group of size N, the key server needs to encrypt and send N keys without considering packet losses. For large groups with very frequent membership changes, scalable group rekeying becomes an especially challenging issue.

In recent years, many approaches for scalable group rekeying have been proposed. Among them, Logical Key Hierarchy (LKH) [22, 23], One-way Function Trees (OFT) [2, 4], and Subset-Difference [16, 10],

work on the key encoding phase; Proactive-FEC [25] and Weighted Key Assignment and Batched Key Retransmission (WKA-BKR) [21]), work on the key distribution phase. Due to different research focuses and diversity of performance metrics such as bandwidth overhead (the bandwidth used for sending or receiving new keys), rekeying latency (the time a user waits for receiving its keys after the key server distributes new keys), and storage requirement (the memory the key server or a user uses to store relevant keys), each approach has its own merits, and some of them are complementary to each other. Therefore, rather than comparing one scheme to another in detail, we are more interested in showing the previous research trends and envisioning some future research directions. We studied these approaches chronologically and made the following observations:

- The research interests have been moving from stateful protocols to stateless protocols. In a stateful protocol a user normally has to receive all keys of interest in all previous rekeying operations to be able to extract the current group key, whereas in a stateless protocol, a legitimate user can readily extract the new group key from the received keying materials despite the number of previous group rekeying operations it has missed. Stateless protocols such as Subset Difference [16, 10] are hence more attractive than stateful protocols such as logical key hierarchy (LKH) [22, 23] when no feedback channel exists (e.g., unidirectional communication) or when users go off-line frequently or experience high losses.

- Reliable key distribution has also become a research hot spot. Recently, researchers have proposed customized reliable multicast protocols for group key distribution (e.g., Proactive-FEC [25] and WKA-BKR [21]). Other schemes (e.g., ELK [17]) integrate the key encoding phase with the key distribution phase.

- Self-healing key distribution, which can also be thought of as belonging to the key distribution phase, has drawn much attention recently. A self-healing key distribution scheme [19, 27] provides the property that a user is able to recover the lost previous group keys on its own without asking the key server for retransmission, thus preventing the key server from being overwhelmed by feedback implosion.

- Several optimization schemes [1, 20, 26] have been proposed to further reduce the rekeying communication overhead by exploiting the characteristics of group members such as topology, membership durations, and loss rates.

- The study of group key management is no longer limited on IP multicast. Recently, several group key management schemes have been proposed for wireless networks such as mobile ad hoc networks [13, 29] and sensor networks [28].

The remainder of this paper is organized as follows. We first introduce several existing work in more detail in Section 2, then discuss new research directions in Section 3.

2 A Taxonomy of Group Rekeying Protocols

This section details several group rekeying schemes based on our earlier observations. The schemes we will discuss do not cover the literature entirely. We focus on *scalable* and *centralized* group rekeying, which has become the mainstream of research on secure multicast recently. Given so many work in this direction, we believe that scalable group rekeying deserves an independent study. We do not discuss contributory group rekeying, in which every member is required to contribute a piece of information to generate a new group key, and other tightly related issues such as multicast source authentication.

2.1 Stateful Protocols

A stateful protocol is one in which normally a member has to receive all keys of interest in all of the previous rekeying operations to be able to decrypt the new group key, unless it asks the key server for key retransmission. Most of the logical-key-tree-based group rekeying protocols in the literature (e.g., LKH [22, 23], OFT [2], and ELK [17]) are stateful protocols. We illustrate below the LKH and OFT schemes and show why they are stateful.

Logical Key Hierarchy(LKH) The basis for the LKH approach for scalable group rekeying is a key tree data structure maintained by the key server. The root of the key tree is the group key used for encrypting data in group communications and it is shared by all users. Every leaf node of the key tree is a secret key shared only between an individual user and the key server, and the middle level keys are KEKs used to facilitate the distribution of the root key. Of all these keys, each user owns only those keys that lie on the path from its individual leaf node to the root of the key tree. As a result, when a user joins or leaves the group, all of the keys on its path have to be changed and re-distributed to maintain backward and forward data confidentiality. Note that all the keys in the key tree are randomly generated and there are no functional relationships among them.

We show an example key tree in Fig. 1. In this figure, $K_{1\text{-}9}$ is the group key shared by all users, K_1, $K_2, \ldots, K_9$ are individual keys, and $K_{123}, K_{456}, K_{789}$ are KEKs known only by users who are in the subtrees rooted at these keys. We next illustrate the member join and leave procedures that are called by the key server separately when receiving a member join or leave request.

- **Join Procedure** Suppose in Fig. 1 the root key was $K_{1\text{-}8}$ and K_{789} was K_{78} before user U_9 joined the group, and they are replaced with keys $K_{1\text{-}9}$ and K_{789} respectively when U_9 joins. All the users need $K_{1\text{-}9}$, but only U_7, U_8 and U_9 need K_{789}. To distribute these new keys to the members of interest securely, the key server encrypts $K_{1\text{-}9}$ with $K_{1\text{-}8}$, K_{789} with K_{78}, and $K_{1\text{-}9}$ and K_{789} with K_9. Let $Enc(m, k)$ denote encrypting message m with key k, and $x|y$ denote the concatenation of messages x and y. The message multicast by the key server is:

$$KeyServer \longrightarrow All : Enc\{K_{1\text{-}9}, K_{1\text{-}8}\}, Enc\{K_{789}, K_{78}\}, Enc\{K_{1\text{-}9}|K_{789}, K_9\}.$$

  Each user can extract the keys it needs independently. For example, user U_1 decrypts the first item in the message to obtain the new group key $K_{1\text{-}9}$; besides $K_{1\text{-}9}$, user U_7 also decrypts the second item to obtain key K_{789}. Here we can see that some users are only interested on a fraction of the rekeying payload. This is referred to as *sparseness property*.

- **Departure Procedure** When user U_4 departs from the group, the keys K_{456} and $K_{1\text{-}9}$ need to be changed. Assume that these keys are replaced with keys K'_{456} and $K'_{1\text{-}9}$ respectively. In the join procedure, an updated key can be encrypted by its old key for distribution. In the departure procedure, however, an updated key is encrypted by its child keys that are unknown to the revoked users, because these users also know the old key. Therefore, the key server encrypts $K'_{1\text{-}9}$ with K_{123}, K'_{456} and K_{789} separately, encrypts K'_{456} with K_5 and K_6 separately, and then multicasts these five encrypted keys to the group.

$$\begin{aligned} KeyServer \longrightarrow All : \quad & Enc\{K'_{1\text{-}9}, K_{123}\}, Enc\{K'_{1\text{-}9}, K'_{456}\}, Enc\{K'_{1\text{-}9}, K_{789}\}, \\ & Enc\{K'_{456}, K_5\}, Enc\{K'_{456}, K_6\}. \end{aligned}$$

  Also due to the sparseness property, after it receives the broadcast message, a member extracts the encryptions that are of interest to it to obtain the relevant updated keys.

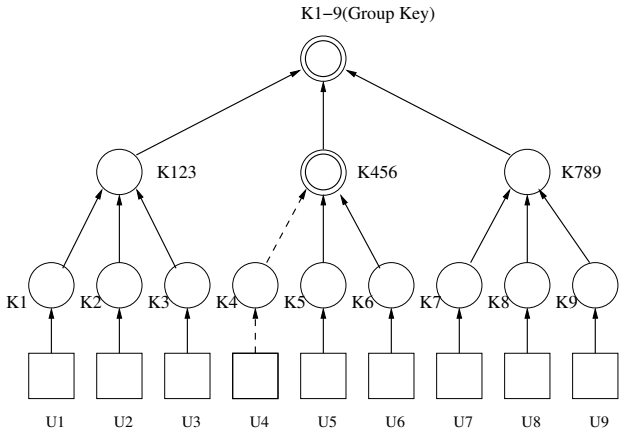

Figure 1: An example of a logical key tree. The root key is the group key and a leaf key K_i is an individual key shared between the key server and a user U_i.

Consider user U_7 in Fig. 1. During the group rekeying for adding U_9 into the group, U_7 must receive K_{789}; otherwise, it will not be able to decrypt the new group key $K'_{1\text{-}9}$ (encrypted by K_{789}) during the group rekeying for revoking user U_4. Therefore, LKH is a stateful rekeying protocol.

LKH is very efficient and hence scalable for group rekeying when compared to a unicast-based naive approach. Let N be the group size and d be the degree of the key tree (the optimal degree d is shown to be 4 [22]). The communication cost in LKH is $O(log_d N)$, whereas the unicast-based approach requires a communication cost of $O(N)$. In LKH+ [9], when a new user joins, every group member applies a one-way hash function to the affected keys instead of the key server redistributing them, thus the communication cost can be significantly reduced.

One-way Function Trees(OFT) In OFT [2] the key server maintains a binary key tree. Unlike in LKH, in OFT an interior key in the key tree is derived from its child keys. Each node v has a node secret x_v and a node key k_v. The node secrets of interior nodes and the node keys are computed as follows: $x_v = \langle f(x_L) \oplus f(x_R) \rangle$ and $k_v = g(x_v)$, where L and R are, respectively, the left and right children of v, f and g are special one-way functions, and $\oplus$ is bitwise exclusive-or. The node secret of the root is used as the group key. Often, $f(x_v)$ is called the blinded key of x_v because knowing $f(x_v)$ does not enable the recovery of x_v due to one-wayness of function f. Figure 2 shows the structure of an example OFT key tree.

In OFT, the key server and all members individually compute the group key. A member only knows its own node secret and the blinded keys of the sibling nodes of the nodes on its path to the root node, and these keys allow it to compute the group key through a bottom-up approach. For example, in Figure 2, member U_4 only knows x_7 and $f(x_6), f(x_2)$. It can derive x_3 from x_7 and $f(x_6)$, then derive the group key x_1 from x_3 and $f(x_2)$.

When there is a member join or leave, the node secrets of the nodes on the path from the member to the root node must be updated. Consequently, the blinded keys of these node secrets need also be updated. To securely distribute the updated blinded key $f(x'_v)$ of x'_v to and only to the members who need it (i.e., the members who are in the subtree rooted at v's sibling node s), the key server encrypts $f(x'_v)$ with k_s, the node key of s. Note that the key server does not need to distribute x'_v to the members that are in the subtree rooted at v because these members can derive x'_v from the blinded keys of v's child nodes. As a result, the bandwidth overhead of OFT in a member join/leave is about $log_2 N$ keys, which is half of that for revoking a member in LKH because there an updated key is encrypted by its each of its child keys separately.

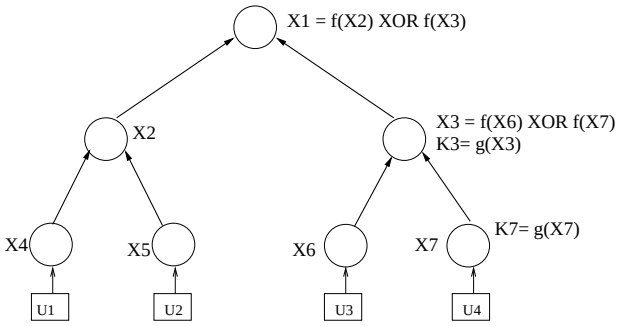

Figure 2: An OFT key tree and the functional relationships among its node secrets and node keys.

Batched Rekeying For a large group with very dynamic memberships, LKH or OFT may not scale well [18] because it performs a group rekeying for every membership change. To reduce the frequency of group rekeying operations, researchers have proposed to use batched rekeying [18, 25] instead of individual rekeying. Batched rekeying can be done in a periodic fashion so that the rekeying frequency is decoupled from the membership dynamics of a group and hence the processing overhead at the key server can be reduced. In addition, using batched rekeying can reduce the overall bandwidth consumption significantly. This is because every compromised key in the key tree, due to member joins or leaves, only needs to be updated once despite the number of joins and leaves. For example, in Fig. 1 when users U_4 and U_6 both depart from the group during the same rekeying period, $K_{1\text{-}9}$ and K_{456} only need to be changed once.

2.2 Stateless Protocols

A stateless rekeying protocol is one that allows a legitimate group member to obtain the group key from the received rekeying materials, despite the number of previous rekeying operations it has missed. The statelessness property is very desirable when no feedback channel exists (e.g., encrypted DVD distribution or stealthy radio receivers) or when group members go off-line frequently or experience high packet losses. A simplest stateless protocol, which we refer to as *flat-tree rekeying*, is one in which the key server encrypts the group key with the individual key of each member and then multicasts all the encryptions. Every member uses its individual key to decrypt one of the encryptions to obtain the group key. However, this scheme does not scale well with the group size. More scalable stateless protocols include subset-difference rekeying (SDR) [16] and MARKS [3], which we introduce below in more detail.

Subset-Difference Rekeying (SDR) In SDR, the key server maintains a logical binary key tree and maps every member to a leaf node of the key tree. Let V_i and V_j be two vertices in the key tree and V_i an ancestor of V_j. Let S_{ij} be a subset, then S_{ij} can be thought as the set of users in the sub-tree rooted at node V_i minus the set of users in the sub-tree rooted at node V_j (see Fig. 3 for an example). Each subset is associated with a unique key that is only known to the members belonging to this subset. During a rekey operation, the key server partitions the current members of the group into a *minimal* number of such subsets (see Fig. 4 for an example). It then encrypts the new group key with the unique key of each subset separately. Hence, the number of encrypted keys to be distributed to the users is the same as the number of subsets the method generates. A group member only needs to receive exactly one encrypted key in every rekeying operation, which is the new group key encrypted with the key of a subset to which it belongs. The SDR scheme falls back to the flat-tree rekeying scheme when every subset contains one member.

In SDR the average number of subsets is $1.25r$ when there are totally r revoked users in the system. The

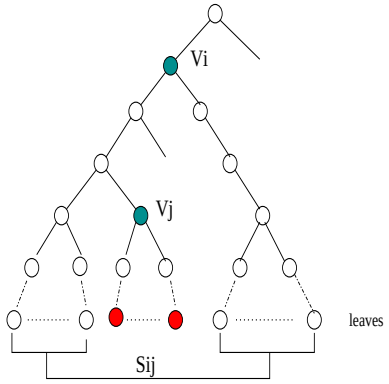

Figure 3: The Subset Difference Rekeying Method. The solid leaf nodes denote the revoked users. Subset S_{ij} contains the current members.

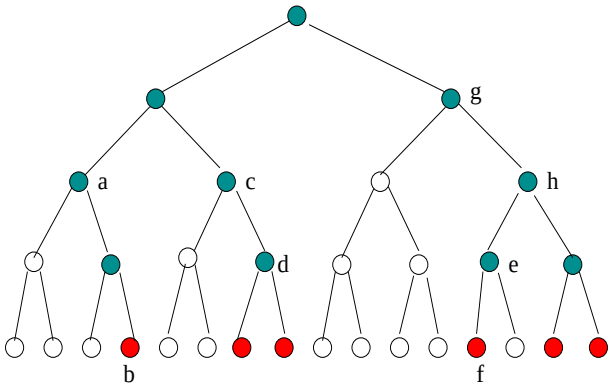

Figure 4: An example of SDR. A solid leaf node denotes a revoked user and all the solid nodes form a minimum spanning tree that connects all the revoked users. The subsets $\{S_{ab}, S_{cd}, S_{ef}, S_{gh}\}$ cover all the remaining users.

communication complexity (i.e., the number of subsets) is independent of the group size N, which makes this scheme very scalable with the group size. However, because r always grows, the performance of SDR degrades with time. Note that in LKH the rekeying cost is determined by the group size N and the number of users being revoked since the *previous* rekeying operation; thus, LKH does not incur this performance degradation. This comparison suggests that SDR is suitable for applications in which the number of revoked users r is relatively small, particularly when $r \ll N$, whereas LKH seems to be preferable for applications that have a large number of revoked users. On the other hand, in SDR each user stores $0.5log^2N$ keys, whereas in LKH each user stores $logN$ keys. [10] presents a variant of SDR, which allows a tradeoff between user storage and communication cost.

MARKS MARKS is mainly designed for receiver-initiated Internet multicast applications where the membership duration of a member is known to the key server when the member joins the group (e.g., a user may subscribe to a multimedia distribution program for a certain time period). In MARKS, the key server maintains a binary hash tree that is constructed through a top-down approach using a random key as the root key. Except the root key, every other keys in the key server is derived from its parent key based on a one-way hash function. Figure 5 shows an example. $S(0,0)$ is a random key. From $S(0,0)$ the key server derives $S(1,0)$ and $S(1,1)$ as follows:

$$S(1,0) = f(S(0,0)), S(1,1) = g(S(0,0)).$$

Here f and g are different one-way functions. Recursively, the key server derives all the other keys.

MARKS divides a multicast application into multiple sessions, and each session uses a unique group key that is a leaf key in the binary hash tree. For example, in Figure 5, $K0$ is the session key for the time interval $[T0, T1]$, $K1$ is for $[T1, T2]$, and so on. Since the membership duration of a member is assumed to be known to the key server when the member joins, the key server sends to the member the subset of leaf keys that are used for the duration. Indeed, the key server can minimize the size of the keying message by sending some interior keys instead of the leaf keys if the leaf keys can be derived from them. For example, in Figure 5, if a member has membership duration from $T0$ to $T4$, it only needs to receive $S(1,0)$ and $K4$. From $S(1,0)$ it can derive $K0, K1, K2$ and $K3$ based on f and g.

MARKS does not incur rekeying overhead for member leaves because it completely de-couples senders from the joining and leaving activities of all receivers. Therefore, it scales well with the group size. However, the requirement that membership durations be known in advance limits MARKS to some specific multicast

applications.

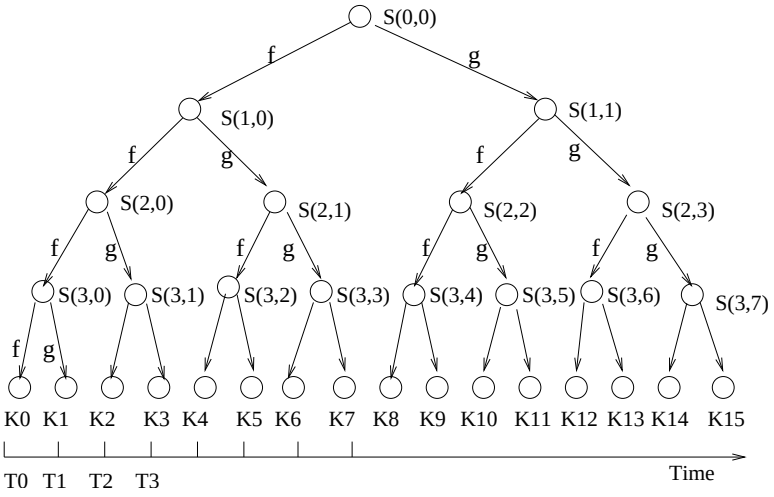

Figure 5: An example of binary hash tree used in MARKS

2.3 Reliable Key Distribution

During a group rekeying operation based on a rekeying algorithm such as LKH, the key server first updates the compromised keys, including the group key and a fraction of KEKs; it then encrypts the updated keys with appropriate noncompromised KEKs; finally it multicasts all of the encrypted keys to the group. A reliable key distribution protocol is designed to ensure that most of the group members receive the keys of interest to them reliably in the presence of packet losses in a network.

Although reliable multicast transport protocols such as SRM [8] and RMTP [14] can be used for reliable delivery of keys, these protocols are complex and (in some cases) require additional support from the network infrastructure. Moreover, the reliable key delivery problem has some characteristics that can be exploited to design custom protocols that are more light-weight in nature.

Recently, several schemes have been proposed for providing reliability for key delivery. Sending every packet multiple times (referred to as a *multi-send* scheme) increases the probability that every user receives the keys of interest to it. However, both the proactive FEC approach [25] and the WKA-BKR approach [21] have been shown to incur a much smaller bandwidth overhead than the multi-send approach. A key server can deploy the proactive FEC, WKA-BKR, or their hybrid WFEC-BKR [27] for reliable distribution of the encrypted keys output from a key encoding algorithm such as LKH, OFT, or SDR. Below we introduce in more detail the proactive FEC, WKA-BKR, and another reliable key distribution protocol called ELK [17].

Proactive FEC-based Key Delivery In the proactive FEC-based approach, the key server packs the encrypted keys into packets of s_k keys. These packets are divided into FEC blocks of k packets each; the last block is padded with default packets if not full. The key server then generates $\lceil (\rho - 1)k \rceil$ parity packets for each block based on Reed Solomon Erasure correcting codes [15], where $\rho \geq 1$ is the proactivity factor determined by the loss rates of the group members. The key server broadcasts all the packets.

A user interested in the packets from a certain block can recover all of the original packets in the block as long as it receives any k out of $\lceil k\rho \rceil$ packets from the block. If a user does not receive the packet that contains the encrypted keys of interest to it, and only receives $t (t < k)$ packets from the block that contains this packet, it will need to ask the key server for retransmission of $k - t$ new parity packets.

The key server collects all of the retransmission requests, and then for each block it generates and transmits the *maximum* number of new parity packets required by users. The retransmission phase continues until all of the users have successfully received their keys.

WKA-BKR The WKA-BKR scheme uses the simple packet replication technique which sends every packet multiple times, but it exploits two properties of logical key trees. First, the encrypted keys may have different replication weights, depending on the number of users interested in them and the loss rates of these users. For example, in LKH the keys closer to the root of the key tree are needed by more members than are the keys less closer to the root, and in SDR some subsets cover a larger number of users than other subsets do. If a key is needed by more users and these users have a higher loss rate, it should be given a higher degree of replication so that most of these users will receive the key reliably and timely. Hence, in this scheme the key server first determines the weight w_i for each encrypted key K_i based on the number of users interested in that key and the loss characteristics of these users. It then packs the keys that have the same weight $\lfloor w_i \rfloor$ into a set s_i of packets. When broadcasting the packets, the key server sends packets in s_i $\lfloor w_i \rfloor$ times. This process is called weighted key assignment (WKA).

The second exploited property is the sparseness of rekeying payload, i.e., a user typically only needs to receive a small fraction of keying messages to decode the group key. The sparseness property suggests that when a user requests for the retransmission of its lost keys, there is no need for the key server to retransmit the entire packet that contains the requested keys sent in the previous round. Instead, the key server repackages the requested keys into new packets before retransmitting them. This process is called batched key retransmission (BKR). The WKA-BKR scheme has been shown to have a lower bandwidth overhead than the multi-send and the proactive-FEC schemes in many network settings.

ELK Unlike LKH, OFT, WKA-BKR, or proactive FEC, which provides either key encoding or key distribution, ELK [17] integrates the key encoding phase and the key distribution phase. In ELK, the key server maintains a binary key tree that is similar to the one in OFT [2]. To update a key in the key tree, the key server derives the new key from the old key and contributions from both its child keys, based on pseudo-random functions. The child keys are updated, if necessary, in the same way recursively. A member can update the key in the same way if it knows the old key and the child keys.

During a group rekeying, the key server not only generates new keys but also derives some "hint" information from these new keys based on pseudo-random functions. The new keys are distributed alone without replication, whereas the hints are usually replicated and distributed together with data packets.

When a member receives both the child keys of a new key and knows the old key, it can compute the new key based on pseudo-random functions. However, if it only receives one child key but also the hint of the new key, it can recover the new key by brute-forcing a small key space (e.g., 16 bits). On the other hand, a revoked user must brute-force at least a larger key space (e.g., 44 bits) to be able to recover the same key.

ELK achieves reliability and moderate security through a tradeoff of communication overhead with member computation. Note that ELK is also a stateful rekeying protocol because a user must receive or recover the current group key to be able to decrypt the next group key.

2.4 Self-Healing Key Distribution

The reliable key delivery protocols discussed in Section 2.3 work well for scenarios where a user experiences random packet losses. However, a user might have to request packet retransmissions multiple times until it finally receives the encrypted keys of interest to it. In other words, there is no guarantee that it will receive the group key before the next group rekeying event occurs. This is especially true for receivers that are experiencing intermittent burst packet losses. Another similar scenario arises when a user is off-line (while

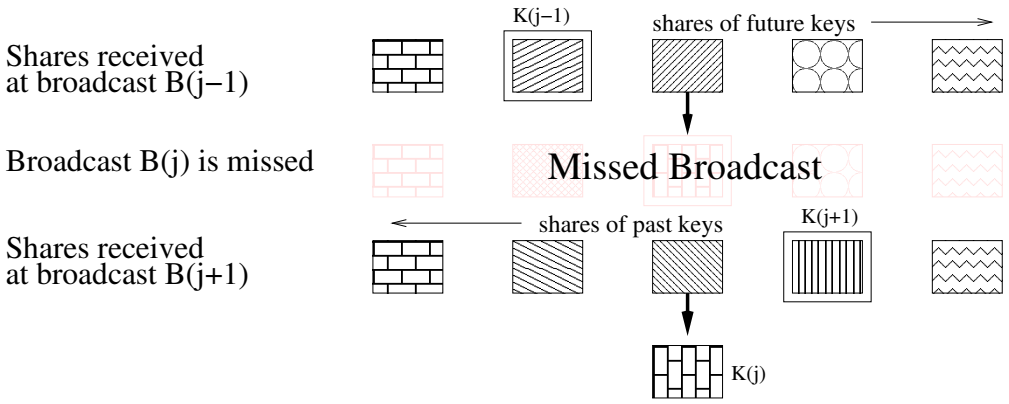

Figure 6: The self-healing scheme. A member can recover the missing group key $K(j)$ broadcast at $B(j)$ by combining the two shares of $K(j)$ it receives from $B(j-1)$ and $B(j+1)$

still a member of the group) at the time of a group rekeying. If the user receives data that was encrypted using a group key that it has not received, it will need to obtain that group key.

A self-healing key delivery protocol allows a user to obtain missing group keys on its own without requesting a retransmission from the key server. This is accomplished by combining information from the current key update broadcast with information received in previous key update broadcasts. In this section, we discuss two such schemes. We say a scheme has *m-recoverability* if the maximum number of previous group keys a legitimate user can recover is m.

2.4.1 Polynomial-based Self-healing

[19] proposes the first self-healing key distribution protocol that uses polynomial-based secret sharing techniques. In this protocol, the key server divides the lifetime of a multicast program into m sessions, each of which uses a unique session(group) key for encrypting the data distributed in the session period. A group key is divided into two shares. When rekeying the group, the key server broadcasts the current group key and shares of all the past and future group keys. To recover a missing key, a member only needs to receive two shares of the key at different rekeying operations. Figure 6 demonstrates this idea.

[19] presents a construction of this protocol with polynomial-based secret sharing techniques. Let F_q be a finite field where q is a prime number that is large enough to accommodate a cryptographic key. All the operations of this construction take place in F_q. The protocol involves three phases, as demonstrate below.

Setup The key server randomly generates $2m$ polynomials of degree-t in $F_q[x]$, $h_1, h_2, ..., h_m, p_1, p_2, ..., p_m$, and m session keys, $K(1), K(2), ..., K(m) \in F_q$. For each $j \in \{1, m\}$, it defines a polynomial in $F_q[x]$, $q_j(x) = K(j) - p_j(x)$. The key server finally delivers $S_i = \{i, h_1(i), h_2(i), ..., h_m(i)\}$ to member i through a secure and reliable channel.

Broadcast In session $j \in \{1, 2, ..., m\}$, the key server broadcasts $B(j)$

$$\begin{aligned} B(j): \quad & \{h_1(x) + p_1(x)\}, ..., \{h_{j-1}(x) + p_{j-1}(x)\}, \\ & \{h_j(x) + K(j)\}, \\ & \{h_{j+1}(x) + q_{j+1}(x)\}, ..., \{h_m(x) + q_m(x)\}. \end{aligned}$$

Session Key and Shares Recovery for Session j A member i receiving $B(j)$ can recover $K(j)$ by evaluating $h_j(x) + K(j)$ at $x = i$ and subtracting $h_j(i)$. Similarly, it can recovers session key shares $p_1(i), p_2(i), ..., p_{j-1}(i), q_{j+1}(i), ..., q_m(i)$. These shares allow member i to recover $K(j)$ (if it misses $B(j)$) based on a $q_j(i)$ it received earlier and a $p_j(i)$ it will receive later.

The above scheme allows self-healing key distribution, but it does not have the revocation capability because a revoked user can continue recovering the future session keys from the future broadcast messages. [19] addresses this issue by combining the above self-healing scheme with a personal key distribution scheme that has revocation capability. Note that the security of this protocol is determined by t, the degree of the polynomials, because if more than t members collude, with their shares they will be able to recover the polynomials based on Lagrange interpolation, thus breaking the system. Therefore, the key server has to set t be larger than the maximal number of revoked nodes in the m sessions.

The protocol is also stateless in addition to self-healing because a member can always derive the current group key from the currently received broadcast message. The broadcast overhead of this protocol is $O(t^2m)$ key sizes. [12] reduces the bandwidth overhead of this protocol to $O(tm)$ by introducing a more efficient personal key distribution protocol that has revocation capability.

2.4.2 Self-healing SDR

[27] proposes a self-healing key distribution scheme for SDR. The key idea is to bind the ability of a user to recover a previous group key to its membership duration. Below we illustrate the basic scheme through an example where $m = 5$.

Consider Fig. 7. Let $T(10)$ be the current rekeying time. The key server first generates a random key $K^5(10)$, based on which it derives a hash key chain including $K^5(10), K^4(10), ..., K^1(10), K^0(10)$, where $K^0(10) = H(K^1(10)) = H^2(K^2(10)) = ... = H^5(K^5(10))$ and H is a one-way hash function. Due to the one-wayness of a hash function, a user knowing $K^j(10)(0 < j < 5)$ can compute independently all the keys between $K^{j-1}(10)$ and $K^0(10)$, but not any of the keys between $K^{j+1}(10)$ and $K^5(10)$. $K^0(10)$ is the group key that all the users should use for data encryption between $T(10)$ and $T(11)$.

The key server then divides the current members of the group into six subgroups according to their membership durations. Applying the SDR algorithm to each subgroup in the key tree, it generates and sends $K^0(10)$ to the newly joined members, $K^1(10)$ to those joining at $T(9)$, $K^2(10)$ to those joining at $T(8)$, $K^3(10)$ to those joining at $T(7)$, $K^4(10)$ to those joining at $T(6)$, and $K^5(10)$ to all those joining at or before $T(5)$. Finally it broadcasts a message $B(10)$

$$B(10) : \{K^0(5)\}_{K^0(4)\oplus K^5(10)}, \{K^0(6)\}_{K^0(5)\oplus K^4(10)}, ..., \{K^0(8)\}_{K^0(7)\oplus K^2(10)}, \{K^0(9)\}_{K^0(8)\oplus K^1(10)}. \quad (1)$$

Consider a current member that joined before $T(5)$. Suppose it has received $K^0(4)$ (or any other keys in the key chain for $T(4)$) and $K^5(10)$ from the key server but missed all the intermediate keys. From $K^5(10)$ it first derives all the other keys in the key chain, i.e. $K^4(10), ..., K^1(10), K^0(10)$. It then decrypts the first item in $B(10)$ to recover $K^0(5)$, using which and $K^4(10)$ to recover $K^0(6)$, and so on to recover all other keys. If the member joined at time $T(7)$, it can only recover $K^0(8)$ and $K^0(9)$. This shows that the self-healing capability of a member is determined by its membership duration and is bounded by the design parameter m. Experimental results show that, in general, this protocol incurs bandwidth overhead of at most $3m$ keys.

2.5 Rekeying Optimization

The logical key tree in LKH is usually required to keep balanced, so that the rekeying cost is fixed to be $log(N)$ for a group of size N. However, [20] shows that it is beneficial to use an unbalanced key tree in

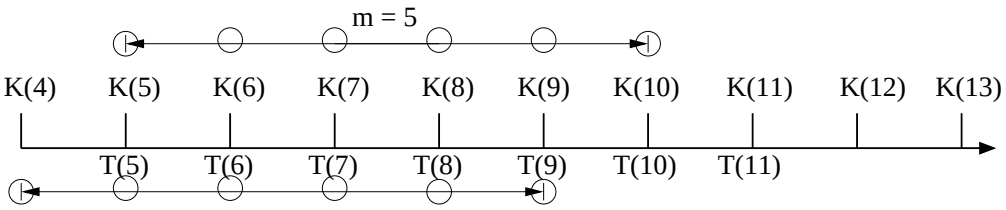

Figure 7: An example illustrating self-healing SDR where $T(i)$ is a group rekeying time point. $T(10)$ is the current rekey time.

scenarios where group members have different membership durations. The idea is to organize the key tree with respect to the compromise probabilities of members, in a spirit similar to data compression algorithms such as Huffman and Shannon-Fano coding. Namely, the key server places a member that is more likely to be revoked closer to the root of the key tree. If the key server knows in advance or can make a good guess of the leaving probability of each member, this probabilistic organization of the LKH tree could lead to a smaller communication overhead than that in a balanced-key-tree case. [1] shows that organizing members in a key tree according to their topological locations could also be beneficial, if the multicast topology is known to the key server.

[26] shows two performance optimizations that are applicable to group key management schemes based on the use of logical key trees. These optimizations involve simple modifications to the algorithms and data structures used by the key server to maintain the logical key tree for a group. The first optimization exploits the temporal patterns of group member joins and leaves. The main idea is to split the logical key tree into two partitions—a short-term partition and a long-term partition. When a member joins the group, the key server initially places it in the short-term partition. If the member is still in the group after a certain time threshold, the key server then moves it from the short-term partition to the long-term partition. The second scheme exploits the loss probabilities of group members. The key server maintains multiple key trees and places members with similar loss rates into the same key tree. Thus, the key server separates keys needed by high loss members from those needed by low loss members when it packs keys into packets. These two optimizations are shown to reduce communication overhead over the one-key-tree scheme for applications with certain member characteristics.

2.6 Group Rekeying in Ad-hoc and Sensor Networks

Most of the previous group rekeying schemes were designed for secure multicast communications in the context of wired networks such as IP Multicast. Recently, as wireless networks such as ad hoc and sensor networks become the research hot spot, researchers have proposed several customized group rekeying schemes for wireless networks.

2.6.1 Group Rekeying for Ad-hoc Networks

An ad hoc network is a collection of wireless mobile nodes, and it can support communications in environments where no wired infrastructure exists. To communicate with parties beyond direct wireless transmission range, nodes in such a network need to cooperate to forward packets for each other; as a result, they act as both hosts and routers. In situations where no infrastructure (e.g., a wired and fixed base station) is available, such a network can be quickly and inexpensively formed to route traffic. Example applications of ad hoc networking include voice and video communications in a battlefield, disaster relief, ubiquitous computing. Some of these applications involve collaborative computing among a large number of nodes and

are thus group-oriented in nature. For deploying such applications in an adversarial environment such as a battlefield or even in many civilian commercial scenarios, it is necessary to provide support for secure group communication. As a result, we have to deal with the group key management issue.

Group key management for ad-hoc networks poses several new challenges with respect to wired networks. First, group key management schemes for ad-hoc networks must be more resource-efficient, because the resources of a node such as power, computation and communication capacity, and storage are relatively constrained. Second, packet loss probability in ad-hoc networks is much higher than in wired networks. This is mainly due to the unreliable transmission links and temporary network partitions caused by node mobility. Thus, group key management schemes for ad-hoc networks must work efficiently under high packet loss rates.

These new challenges seem to rule out the stateful group rekeying protocols such as LKH and OFT that were proposed for wired networks with low packet loss rates, although no formal study on this has been conducted yet. Note that combining reliable key distribution protocols with stateful protocols does not solve the problem in the case of network partitions. Therefore, stateless protocols are preferred. However, stateless protocols such as SDR do not scale with the number of revoked nodes in the system.

Below we briefly introduce two symmetric-key based group rekeying schemes for ad-hoc networks. The first one [13] is an energy-aware group rekeying scheme for ad hoc networks. This scheme exploits the physical locations of the members when organizing the logical key tree (LKH [22]). This exploitation could save $15\% \sim 37\%$ energy in some scenarios, compared to a random LKH that does not use the physical location information. Two assumptions are made in order to use the location information. First, the network topology is assumed to be static. Second, the key server knows the exact location of every node.

The second scheme, called GKMPAN [29], exploits the property of an ad-hoc network that member nodes are both hosts and routers. In IP Multicast, all group members are end hosts, and they have no responsibility for forwarding keying materials to other group members. In contrast, for group communication in an ad hoc network, the members of the group also act as routers. As such, in GKMPAN the key server only has to deliver the new group key securely to the group members that are its immediate neighbors, and these neighbors then forward the new group key securely to their own neighboring members. In this way, a group key is propagated to all the members in a hop-by-hop fashion.

For the above scheme to work, a fundamental requirement is the existence of a secure channel between every pair of neighboring nodes. GKMPAN provides secure channels through probabilistic key predeployment [7]. The main idea is to randomly load every node with a subset of keys from a large key pool before the node joins a network. Two nodes can establish a secure channel directly if they share a common key in their key subsets; otherwise they can request a third node that shares a common key with each of them to establish a secure channel for them. To prevent compromised nodes from colluding to jeopardize the secure channels between other nodes, GKMPAN also includes a distributed key updating scheme for updating any compromised channels. The transmission cost per node in this scheme is close to one key and is independent of group size. Moreover, GKMPAN also provides the stateless property. Therefore, GKMPAN is much more efficient than LKH-like schemes that are used in ad-hoc networks.

2.6.2 Group Rekeying for Sensor Networks

Sensor networks can also be considered as ad-hoc networks, except that they are formed by sensor nodes that are more limited in power, computational and communication capacities, and memory. The current generation of sensor nodes such as Berkeley Mica Motes [24] have only 4MHZ Processors, 19.2bps bandwidth, and 4KB RAM. In a sensor network, the network controller may broadcast missions, commands, or queries to all the nodes; therefore, a secure group communication model is also needed for sensor networks deployed in security critical environments. When compromised sensor nodes are detected, the network

controller needs to update the group key and revoke the compromised nodes.

The group rekeying schemes [13, 29] for ad-hoc networks could be deployed in sensor networks when their assumptions hold. [28] introduces a group rekeying scheme similar to GKMPAN [29], except that a secure channel between two neighboring sensor nodes is established upon other KEKs that are already in place. The transmission cost per node is shown to be one key, which provides the optimal performance for node revocation.

Summary Generally speaking, stateful protocols are more bandwidth efficient than stateless protocols, whereas stateless protocols are more preferable in the presence of high packet loss or when members go off-line frequently. Both stateful and stateless protocols need a key distribution mechanism to distribute a new group key to all members reliably and in a timely fashion, and a self-healing key distribution mechanism to allow a member to recover a certain number of previous group keys on its own. The performance of group rekeying schemes could be further improved by exploiting the characteristics of group members such as their topological distribution, temporal patterns and packet loss rates.

Compared to wired networks, ad-hoc and sensor networks are more constrained in the resources of member nodes and have higher packet loss rates. Group rekeying schemes exploiting the property that member nodes are both hosts and routers in these networks could provide significantly better performance than those adopted directly from the schemes proposed for wired networks.

3 New Research Directions

The group key management problem has been studied for about ten years in the context of IP Mutlicast, and many scalable schemes [2, 16, 22, 23] have been proposed. Each of these schemes has its own merits, subject to network size, membership dynamics and loss characteristics. In the future, we can try to design a "perfect" group rekeying scheme. Ideally, a scalable group rekeying scheme should not only have the communication overhead close to that of a stateful protocol such as LKH or OFT but also have embedded reliability and self-healing mechanisms. If we cannot design such a perfect scheme, the alternative is to augment a stateful protocol with some lightweight plugins. For example, we have seen that a customized reliable multicast scheme such as Proactive FEC or WKA-BKR can be used to increase the reliability of key delivery for LKH, but we do not know if there is a scheme that can add self-healing to LKH, just as the scheme in [27] adding self-healing to the stateless protocol SDR.

Although IP multicast was proposed more than ten years ago, its deployment has been very slow due to both technical and operational concerns. Recently researchers have proposed the concept of overlay multicast (also called EndSystem multicast) [6, 5], which shifts multicast support from routers to end systems. Overlay multicast is a promising technique because it is independent of the underlying physical topology and therefore bypasses the limitations of IP multicast. We may directly deploy the existing group key management schemes proposed for IP multicast in overlay multicast, but these schemes might not provide the optimal performance. Therefore, further research study in this direction is needed.

For ad-hoc and sensor networks, if secure channels between neighboring nodes cannot be established efficiently (e.g., through key pre-deployment [29]), it is unclear, with respect to energy consumption, if a group rekeying scheme exploiting the same property as in GKMAPN [29] will outperform those designed for wired networks. For example, in [11], the secure channels are established upon public-key cryptography, which is very expensive in terms of computation and communication cost. Therefore, research that investigates issues such as which scheme is the best in what networks with what kind of loss probability and mobility models would be very helpful.

A related open issue is to distribute key serving responsibility over several key servers to improve reliability and survivability. This is especially important in the context of mobile ad hoc networks where a

key server may not be accessible due to node mobility. Having multiple replica key servers could address this issue, but it introduces the security weakness of compromising one key server breaking the system. An alternative solution is to use threshold cryptography, in which multiple key servers have to collaborate to perform group key management tasks. So far no efficient schemes have been proposed for this purpose.

Another related open issue is node compromise detection in ad-hoc and sensor networks. The key server initiates a group rekeying operation when it knows some node has been compromised. However, node compromise detection is a very challenging issue, especially for sensor networks that are often deployed in unattended environments.

References

[1] S. Banerjee, B. Bhattacharjee. Scalable Secure Group Communication over IP Multicast. International Conference on Network Protocols (ICNP) 2001, Riverside, California, November 2001.

[2] D. Balenson, D. McGrew, and A. Sherman. Key Management for Large Dynamic Groups: One-Way Function Trees and Amortized Initialization. IETF Internet draft (work in progress), August 2000.

[3] B. Briscoe. MARKS: Zero Side Effect Multicast Key Management Using Arbitrarily Revealed Key Sequences. In Proc. of First International Workshop on Networked Group Communication, NGC 1999.

[4] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, B. Pinkas. Multicast Security: A Taxonomy and Some Efficient Constructions. In Proc. of IEEE INFOCOM'99, March 1999.

[5] Y. Chu, S. Rao, S. Seshan, and H. Zhang. Enabling Conferencing Applications on the Internet Using an Overlay Multicast Architecture. In Prof. of ACM SIGCOMM 2001, August 2001.

[6] Y. Chu, S. Rao, and H. Zhang. A Case for EndSystem Multicast. In Proc. of ACM Sigmetrics, June 2000.

[7] L. Eschenauer and V. Gligor. A Key-Management Scheme for Distributed Sensor Networks. In Proc. of ACM CCS 2002.

[8] S. Floyd, V. Jacobson, C. Liu, S. McCanne and L. Zhang. A Reliable Multicast Framework for Lightweight Session and Application Layer Framing. IEEE/ACM Trans. on Networking, December 1997.

[9] H. Harney and E. Harder. Logical Key Hierarchy Protocol Internet Draft, draft-harney-sparta-lkhp-sec-00.txt, March 1999.

[10] D. Halevy, A. Shamir. The LSD Broadcast Encryption Scheme. In Proc. of Advances in Cryptology - CRYPTO 2002

[11] T. Kaya, G. Lin, G. Noubir, A. Yilmaz. Secure Multicast Groups on Ad Hoc Networks. In Proc. of ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN'03),2003.

[12] D. Liu, P. Ning, and K. Sun. Efficient Self-Healing Group Key Distribution with Revocation Capability. In Proc. of the 10th ACM Conference on Computer and Communications Security (CCS '03), pages 231–240, Washington D.C., October, 2003.

[13] L. Lazos and R. Poovendran. Energy-Aware Secure Multicast Communication in Ad-hoc Networks Using Geographic Location Information. In Proc. of IEEE ICASSP'03, Hong Kong, China, April, 2003.

[14] J. Lin and S. Paul. RMTP: A Reliable Multicast Transport Protocol, In Proc. of IEEE INFOCOM '96, March 1996.

[15] A. Mcauley. Reliable Broadband Communications Using a Burst Erasure Correcting Code. In Proc. of ACM SIGCOMM'90, Philadelphia, PA, September 1990.

[16] D. Naor, M. Naor, and J. Lotspiech. Revocation and Tracing Schemes for Stateless Receivers. In Advances in Cryptology - CRYPTO 2001. Springer-Verlag Inc. LNCS 2139, 2001, 41-62.

[17] A. Perrig, D. Song, D. Tygar. ELK, a new protocol for efficient large-group key distribution. In Proc. of the IEEE Symposium on Security and Privacy 2001, Oakland, CA, May 2001.

[18] S. Setia, S. Koussih, S. Jajodia, E. Harder. Kronos: A Scalable Group Re-Keying Approach for Secure Multicast. In Proc. of the IEEE Symposium on Security and Privacy, Oakland, CA, May 2000.
[19] J. Staddon, S. Miner, M. Franklin, D. Balfanz, M. Malkin and D. Dean. Self-Healing Key Distribution with Revocation. In Proc. of the IEEE Symposium on Security and Privacy, Oakland, CA, May 2002.
[20] A. Selcuk, C. McCubbin, D. Sidhu. Probabilistic Optimization of LKH-based Multicast Key Distribution Schemes. Draft-selcuk-probabilistic-lkh-01.txt, Internet Draft, January 2000.
[21] S. Setia, S. Zhu and S. Jajodia. A Comparative Performance Analysis of Reliable Group Rekey Transport Protocols for Secure Multicast. In Performance Evaluation 49(1/4): 21-41 (2002), Special issue Proceedings of Performance 2002, Rome, Italy, September 2002.
[22] C. Wong, M. Gouda, S. Lam. Secure Group Communication Using Key Graphs. In Proc. of SIGCOMM 1998, Vancouver, British Columbia, 68-79.
[23] D. Wallner, E. Harder and R. Agee. Key Management for Multicast: Issues and Architecture. Internet Draft, draft-wallner-key-arch-01.txt, September 1998.
[24] Http://xbow.com.
[25] Y. Yang, X. Li, X. Zhang and S. Lam. Reliable group rekeying: Design and Performance Analysis. In Proc. of ACM SIGCOMM 2001, San Diego, CA, USA, August 2001, 27-38.
[26] S. Zhu, S. Setia, and S. Jajodia. Performance Optimizations for Group Key Management Schemes. In Proc. of the 23rd IEEE ICDCS 2003, Providence, RI, May 2003.
[27] S. Zhu, S. Setia, and S. Jajodia. Adding Reliable and Self-Healing Key Distribution to the Subset Difference Group Rekeying Method for Secure Multicast. In Proc. of 5th International Workshop on Networked Group Communications (NGC 2003), Germany, September 2003.
[28] S. Zhu, S. Setia, and S. Jajodia. LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks. In Proc. of the 10th ACM Conference on Computer and Communications Security (CCS '03), Washington D.C., October, 2003.
[29] S. Zhu, S. Xu, S. Setia, and S. Jajodia. GKMPAN: An Efficient Group Key Management Scheme for Secure Multicast in Ad-hoc Networks. In Proc. of the 1st International Conference on Mobile and Ubiquitous Systems (Mobiquitous'04), Boston, Massachusetts, August 22-25, 2004