2

Centralized Multi-user Key Management

One of the most important challenges for securing group-oriented communications is the issue of key management. As we outlined in the introductory chapter, managing keys in a group-oriented scenario is harder than traditional key management services.

In this chapter, we explore the challenges associated with centralized key management for group-oriented applications. We will begin with an overview of the fundamental limits governing centralized multicast key distribution, and then provide a survey of several approaches that exist in the literature. We then develop a new framework for multicast key management that reduces the communication overhead associated with key management, and show how to best tune this key management scheme to reduce communication overhead.

2.1 Basic Multicast Information Theory

We now provide a summary of information theory results relevant to multicasting. Much of this summary is based upon results that were presented in [3, 4].

First, let $\mathcal{U} = \{u_1, u_2, \cdots, u_n\}$ denote the user set consisting of n users u_j. Associated with each user u_j is a private key K_j that is drawn uniformly from a key space $\mathcal{K}$. Of the n users, only a subset of them will be privileged. We denote the set of private keys associated with privileged members by K_P, and the set of private keys associated with non-privileged users by K_F. For example, if there are $n = 4$ users, and users u_1, u_3 are privileged, then

$K_P = \{K_1, K_3\}$, and $K_F = \{K_2, K_4\}$. There is a secret S that is drawn from a space $\mathcal{S}$ that the group center wishes to transmit to members of the multicast group $\mathcal{U}$. The broadcast message α is a function of the secret S, as well as the private user information of the privileged users, $\alpha = f(S, K_P)$.

It is useful to derive bounds on the size of the broadcast message given the following security constraints:

- The user's secrets K_P and the secret S uniquely determine the broadcast message
$$H(\alpha|S, K_P) = 0. \tag{2.1}$$
- Knowing *only* a user's private key K_j does not decrease the uncertainty of the secret S. That is
$$H(S|K_j) = H(S). \tag{2.2}$$
In particular, this implies that $H(S|K_P) = H(S)$.
- No uncertainty in the secret remains if both a user's private key K_j and the broadcast message are available.
$$H(S|K_j, \alpha) = 0. \tag{2.3}$$
- The broadcast message does not decrease the uncertainty in a user's private key:
$$H(K_j|\alpha) = H(K_j). \tag{2.4}$$
- The broadcast message alone does not decrease the uncertainty in the secret:
$$H(S|\alpha) = H(S). \tag{2.5}$$

The first results that we present are from Just et al. [3]. In the proofs, we have followed the basic derivations provided in [3].

Lemma 1. *The entropy of the broadcast message α is equal to mutual information between the message and the joint random variable (K_P, S):*
$$H(\alpha) = I(\alpha; K_P, S). \tag{2.6}$$

Proof. We start by applying the chain rule to the mutual information:

$$\begin{aligned} I(\alpha; K_P, S) &= I(\alpha; K_{j_1}) + I(\alpha; K_{j_2}|K_{j_1}) + \cdots \\ &\quad + I(\alpha; K_{j_m}|K_{j_1}, K_{j_2}, \cdots, K_{j_{m-1}}) + I(\alpha; S|K_P). \end{aligned}$$

Expanding the mutual information terms yields the telescoping sum:

$$\begin{aligned} I(\alpha; K_P, S) &= H(\alpha) - H(\alpha|K_{j_1}) + H(\alpha|K_{j_1}) - H(\alpha|K_{j_1}, K_{j_2}) + \cdots \\ &\quad + H(\alpha|K_P) - H(\alpha|K_P, S), \end{aligned}$$

which yields

$$I(\alpha; K_P, S) = H(\alpha) - H(\alpha|K_P, S). \tag{2.7}$$

However, $H(\alpha|K_P, S) = 0$, so that

$$I(\alpha; K_P, S) = H(\alpha). \tag{2.8}$$

□

Lemma 2. *Let $D \subset P$ be a subset of privileged members such that $|D| \leq m-1$, and let K_D be the set of private keys associated with users in D. Let K_i be a private key of a user $u_i \in P - D$. Then for a secret S and a broadcast message α, we have*

$$H(K_i) - H(K_i|\alpha, K_D) \geq H(S). \tag{2.9}$$

Proof. The term $H(K_i, S|\alpha, K_D)$ may be expanded in two different ways:

$$\begin{aligned} H(K_i, S|\alpha, K_D) &= H(K_i|\alpha, K_D) + H(S|\alpha, K_D, K_i) && (2.10)\\ &= H(S|\alpha, K_D) + H(K_i|\alpha, K_D, S). && (2.11) \end{aligned}$$

Since $H(S|\alpha, K_j) = 0$ for any user u_j in the privileged set P, we have that $H(S|\alpha, K_D, K_i) = H(S|\alpha, K_D) = 0$, and thus

$$H(K_i|\alpha, K_D) = H(K_i|\alpha, K_D, S). \tag{2.12}$$

Observe that since $I(K_i; S|\alpha) = I(S; K_i|\alpha)$ we have

$$\begin{aligned} H(K_i|\alpha) - H(K_i|\alpha, S) &= H(S|\alpha) - H(S|\alpha, K_i)\\ H(K_i) - H(K_i|\alpha, S) &= H(S). && (2.13) \end{aligned}$$

Since $H(K_i|\alpha, S) \geq H(K_i|\alpha, S, K_D)$, we may apply (2.12) to get $H(K_i|\alpha, S) \geq H(K_i|\alpha, K_D)$. Substituting this result into (2.13) gives $H(K_i) - H(K_i|\alpha, K_D) \geq H(S)$. □

A consequence of this lemma is the fact that each private key K_i will have entropy greater than the entropy of secret, i.e. $H(K_i) \geq H(S)$. We may now put these results together to get a lower bound on the size of the broadcast message given the conditions stated.

Theorem 1. *Suppose that the keys K_j are distributed independently of each other, i.e. $H(K_j, K_i) = H(K_j) + H(K_i)$, and the conditions (2.1)-(2.5) hold, then the following bound on the size of the broadcast message holds:*

$$H(\alpha) \geq mH(S) \tag{2.14}$$

Proof. By Lemma 1 we have

$$\begin{aligned} H(\alpha) &= I(\alpha; K_P, S) && (2.15)\\ &= I(\alpha; K_P) + I(\alpha; S|K_P) && (2.16)\\ &= I(K_P; \alpha) + I(S; \alpha|K_P) && (2.17)\\ &= H(K_P) - H(K_P|\alpha) + H(S) - H(S|\alpha, K_P). && (2.18) \end{aligned}$$

Using the fact that $H(S|\alpha, K_P) = 0$ and that

$$\begin{aligned} H(K_P) &= H(K_{j_1}, K_{j_2}, \cdots, K_{j_m}) && (2.19)\\ &= H(K_{j_1}) + \cdots + H(K_{j_m}), && (2.20) \end{aligned}$$

which follows from the independence of the private keys, we have

$$H(\alpha) = H(K_{j_1}) + \cdots + H(K_{j_m}) - H(K_P|\alpha) + H(S). \quad (2.21)$$

Similarly, expanding $H(K_P|\alpha)$ using the chain rule gives

$$\begin{aligned} H(K_P|\alpha) \quad = \quad & H(K_{j_1}|\alpha) + H(K_{j_2}|\alpha, K_{j_1}) + \cdots \\ & + H(K_{j_m}|\alpha, K_{j_1}, \cdots, K_{j_{m-1}}). \end{aligned} \quad (2.22)$$

Upon substitution we get

$$H(\alpha) = H(K_{j_1}) - H(K_{j_1}|\alpha) + \sum_{i=2}^{m} \Big(H(K_{j_i}) - H(K_{j_i}|\alpha, K_{j_1}, \cdots, K_{j_{i-1}}) \Big) + H(S). \quad (2.23)$$

By observing that $H(K_{j_1}|\alpha) = H(K_{j_1})$, and applying Lemma 2 we get the desired result $H(\alpha) \geq mH(S)$. □

In summary, we have presented two main results from [3] that govern the theoretical underpinnings of multicast key management. The first result that was shown states that the entropy of a user's private information must be greater than the entropy of the secret that is to be distributed to the group. This translates into the security terminology by implying that the bit length of the user's private key should be as large as the bit length of the group secret. It was also shown, under the assumption of independent keys, that the size of the broadcast message must be at least as large the size of the group times the size of the secret that is to be conveyed. This latter result gives a lower bound on the communication requirements for rekeying. In particular, it implies that the best that can be done is a message whose size is linear in the amount of group members unless the key independence assumption is relaxed. As we shall see in the later discussions, the implication of this result is that we must do away with the independence assumption in order to reduce the message size. Currently, the most popular family of multicast key management schemes are those that employ a tree key hierarchy, in which the key information that each user has is not independent of each other.

2.2 Overview of Multicast Key Management

The distribution of identical data to multiple parties using the conventional point-to-point communication paradigm makes inefficient usage of resources. The redundancy in the copies of the data can be exploited in multicast communication by forming a group consisting of users who receive similar data, and sending a single message to all group users [1]. Access control to multicast communications is typically provided by encrypting the data using a key that is shared by all legitimate group members. The shared key, known as the session key (SK), will change with time, depending on the dynamics of group membership as well as the desired level of data protection. Since the key must change, the challenge is in key management– the issues related to the administration and distribution of keying material to multicast group members.

In order to update the session key, a party responsible for distributing the keys, called the group center (GC), must securely communicate with the users to distribute new key material. The GC shares keys, known as key encrypting keys (KEKs), that are used solely for the purpose of updating the session key and other KEKs with group members.

As an example of key management, we present a basic example of a multicast key distribution scheme. Suppose that the multicast group consists of n users and that the group center shares a key encrypting key with each user. Upon a member departure, the previous session key is compromised and a new session key must be given to the remaining group members. The GC encrypts the new session key with each user's key encrypting key and sends the result to that user. Thus, there are $n-1$ encryptions that must be performed, and $n-1$ messages that must be sent on the network. The storage requirement for each user is 2 keys while the GC must store $n+1$ keys. This approach to key distribution has linear communication, computation and GC storage complexity. As n becomes large these complexity parameters make this scheme undesirable, and more scalable key management schemes should be used.

In general, during the design of a multicast application, there are several issues that should be kept in consideration when choosing a key distribution scheme. We now provide an overview of some of these issues.

- **Dynamic nature of group membership:** It is important to efficiently handle members joining and leaving as this necessitates changes in the session key and possibly any intermediate keying information.

- **Ability to prevent member collusion:** No subset of the members should be able to collude and acquire future session keys or other member's key encrypting keys.

- **Scalability of the key distribution scheme:** In many applications the size of the group may be very large and possibly on the order

of several million users. The required communication, storage, and computational resources should not become a hindrance to providing the service as the group size increases.

One approach to group key management is provided by the group key management protocol (GKMP) [5]. In this scheme, the GC uses a SK, called a group traffic encrypting key (GTEK) in the GKMP literature, and a group key encrypting key (GKEK). The GC updates the SK by using the GKEK. This allows all group members to be updated using a single encrypted message. A major disadvantage of GKMP, however, is that it is not able to handle member departures, or the compromise of a single member. The compromise of the GKEK means that all future communication is compromised since an adversary can calculate future session keys.

Fiat and Naor [6] present a broadcast key distribution scheme that allows for a single source to transmit a SK to a dynamic subset of privileged users such that no coalition of at most k non-privileged users can acquire the SK. The communication overhead of their scheme is not dependent on the amount of non-privileged members, but instead on the security parameter k and a parameter describing the probability that a coalition of at most k non-privileged users can acquire the SK.

In Section 2.1, we summarized the theoretical work of [3, 4] for the distribution of secret information via broadcast messages. These results provide an insight into the communication resources needed to achieve the above goals. In particular, it was shown in Theorem 1 that for a key size of B bits, the message needed to update a group of n users must be at least nB bits to provide *perfect security* in the key distribution. One key result of [3] is that in order to achieve a smaller broadcast size, it is necessary to do away with the constraint that the private information held by each user is mutually independent. Therefore, to reduce the usage of communication resources, the users must share secret information.

One strategy for having users share secret information is to arrange the keys according to a tree structure. The tree based approach to group rekeying was originally presented by Wallner et al. [7], and independently by Wong et al. [8]. In such schemes an a-ary tree of depth $\log_a n$ is used to break the multicast group into hierarchical subgroups. Each member is assigned to a unique leaf of the tree. KEKs are associated with all of the tree nodes, including the root and leaf nodes. A member has knowledge of all KEKs from his leaf to the root node. Thus, some KEKs are shared by multiple users. Adding members to the group amounts to adding more depth to the tree, or adding new branches to the tree [8, 9]. Upon member departure the session key and all the internal node KEKs assigned to that member become compromised and must be renewed. Due to the tree structure, the communication overhead is $\mathcal{O}(\log n)$, while the storage for the center is $\mathcal{O}(n)$ and for the receiver is $\mathcal{O}(\log n)$.

Various modifications to the tree scheme have been proposed. In [10], a modification to the scheme of Wallner et al. is presented. By using pseudo-

random generators, their scheme reduces the usage of communication resources by a factor of two. Similarly, Balenson et al. [9] were able to reduce the communication requirements by a factor of two using one-way function trees. The security of the Canetti et al. scheme can be rigorously proven, while the security of the approach using one-way function trees is based upon non-standard cryptographic assumptions and has therefor not been rigorously shown. In [11] Canetti et al. examine the tradeoffs between storage and communication requirements, and a modification to the tree-based schemes of [7, 8] is presented that achieves sublinear server-side storage. Further, in [12], it was shown that the optimal key distribution for a group leads to Huffman trees and the average number of keys assigned to a member is related to the entropy of the statistics of the member deletion event.

2.3 Requirements for Centralized Group Key Management

A conditional access system for group communications must be able to cope with the demands of the application. These demands must not only address the security and access requirements of the service provider, but also address the convenience and satisfaction of the client. Below we have listed several functionalities that are desirable in a conditional access system for dynamic group communication scenarios:

1. *The solution should be able to refresh the keys used to protect content.*

   Due to the bulk quantities of data being multicast, it is feasible that session keys may become compromised. Therefore, it is important that there is a means available to refresh the session key and intermediate keying material in order to maintain a desirable level of content protection.

2. *The solution should provide the ability for members to join and depart the service at will, as well as allow the content distributor to easily revoke a member's ability to access content.*

   Unlike unicast communication, the departure of a group member does not imply the termination of the communication link. In addition, upon departing the service, users must be de-registered and prevented from obtaining future multicasts. Similarly, when new members join the service, it is desirable to prevent them from accessing past content. Additionally, situations might arise where the content provider desires to prevent a user from accessing future content.

3. *The solution should be resistant to member collusion.*

   No subset of the members should be able to collude and acquire keying information of non-colluding members.

4. *The solution should provide a means for an end-user to recover from missed rekeying messages.*

   In many application environments, the connection between a client and the server may be severed. For example, in cellular applications, a client might move temporarily through a region of severe fading. Adverse communication conditions and common accidents, such as a system crash, might mean that the client misses several rekeying messages needed to update his key database. Users might also desire to switch from terminal to terminal, with the possibility of not being able to receive communication while moving across terminals. It is important to have a means that allows the client to resume access to the service.

5. *The solution should allow the user to temporarily transfer access rights to another party.*

   In many business scenarios, a client will subscribe to a service where content, such as multimedia or stock quotes, is streamed. Users may wish to transfer their access rights to the data stream to their friends without canceling or transferring their subscription.

6. *The solution should address the issue of resource scalability for scenarios consisting of large privileged groups.*

   In many applications, the size of the group may be very large and possibly on the order of several million users. The required communication, storage, and computational resources should not become a hindrance to providing the service as the group size increases.

Some of these functionalities have been discussed in other tree-based key management schemes. However, many of these objectives are not considered. For the remainder of this chapter we shall present an architecture for the management of keys in a conditional access multicast system that is capable of achieving each of these requirements. The system that we describe makes use of a tree-structured key hierarchy and basic primitive operations to provide a solution that satisfies the above requirements.

Additionally, whereas most of the multicast key management schemes in the literature do not consider the issue of flagging to the user which rekeying messages are intended for them, we provide this important functionality in our message structure and factor this additional overhead into our considerations. We will focus on the usage of communication resources and calculate the amount of communication needed to perform a member join and a member departure operation for different tree degrees and different amount of users. We determine the optimal tree degree for scenarios where member join is most important, member departure is most important, and where both operations are equally important. Then, in order to better study the optimization of the key management scheme, we present a stochastic

occupancy model that allows one to study the mean behavior of a key tree under different degrees of occupancy. Additionally, we compare the amount of communication overhead needed in our scheme with the amount of communication overhead that a conventional tree-based rekeying scheme, such as [8], would need to flag users which component of a rekeying message is intended for them.

Looking forward, in Section 2.4 we introduce a method for distributing keys using polynomial interpolation and parametric one-way functions. This basic scheme is used as a building block for a protocol primitive described later in the chapter. Therefore, we present a study of its security and communication features. In Section 2.5 we present some protocol primitives and use these to construct more complex key management operations capable of maintaining the key hierarchy in scenarios with dynamic membership. The size of the messages needed for updating the keys is computed in Section 2.6 and are used to determine the optimal degree of the key distribution tree. Additionally, we examine the computational requirements of the tree-based polynomial interpolation scheme proposed in this chapter.

2.4 Basic Polynomial Interpolation Scheme

The heart of the new multicast key management scheme that we will describe involves the use of a polynomial interpolation algorithm that is capable of reducing the communication overhead needed for key management compared with multicast key management schemes that use messages that are concatenations of individual rekeying messages.

In this section we describe the basic scheme for distributing keys that will be used in the scalable key management protocol of Section 2.5. The basic key distribution scheme that we describe is a modification of the polynomial interpolation scheme of [4]. We have introduced the use of one-way functions and a broadcast seed to protect private user KEKs from compromise and allow private user KEKs to be reused.

We shall use keyed (parametric) one-way functions in our work to provide computational security. A one-way function h is a function from $\mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{Z}$ such that given $z = h(x, y)$ and y it is computationally difficult to determine x [13]. Keyed one-way functions, or parametric one-way functions (POWF) are families of one-way functions that are parameterized by the parameter y. Symmetric block ciphers can be used to construct POWFs. Let $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, and consider a symmetric cipher $E_x(y) : \mathcal{Y} \rightarrow \mathcal{Y}$ where the subscript denotes the key used in the encryption of the plaintext y. Thus $\mathcal{X}$ is the key space of the cipher E, while $\mathcal{Y}$ is the space of plaintexts and ciphertexts. Define a hash function $f : \mathcal{Y} \rightarrow \mathcal{Z}$. Then the function $h(x, y) = f(E_x(y))$ is a POWF parameterized by y since any reasonable cryptosystem can withstand a known-plaintext attack, that is knowledge of $E_x(y)$ and y does not make it easy to determine the key x. Note that it is

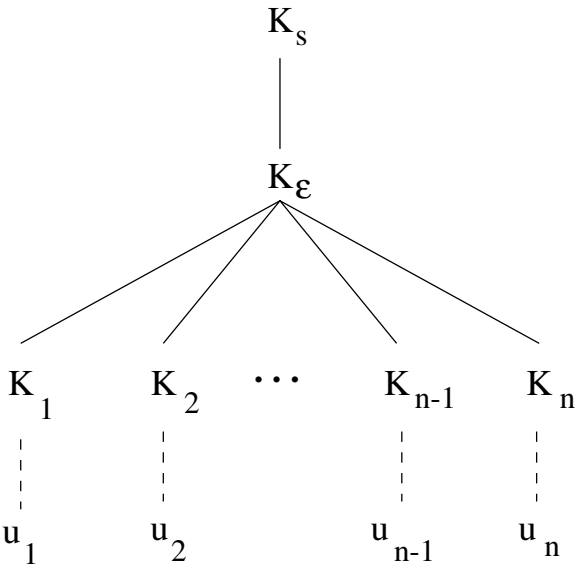


FIGURE 2.1. The basic key distribution scheme used in the polynomial interpolation method.

not necessary that the hash function f have any cryptographic properties as the required cryptographic strength is provided by E. Throughout this chapter we shall assume the existence of parametric one-way functions that map sequences of $2B$ bits into sequences of B bits.

Consider the basic key distribution scheme depicted in Figure 2.1. Each user u_i has a personal B-bit KEK K_i that is known only by the group center and user u_i. Additionally, all of the users share a B-bit root KEK $K_\epsilon(t)$ and a session key $K_s(t)$ that will vary with time t.

Suppose that user u_n decides to depart, then we must renew the keys $K_\epsilon(t-1)$ and $K_s(t-1)$ since they were shared by u_n and the other users. The first step is to send the new $K_\epsilon(t)$ to the remaining users. In the polynomial scheme, each user u_i has the distinct pair $(z_i, K_i) \in Z_p \times Z_p$, where Z_p denotes the integers modulo the prime p. The z_j are public knowledge, and are not considered as part of the secret information that the user must store. Instead, the z_j is any quantity that is used to identify the user, for example a processor id. The GC has made available f, a POWF taking $2B$ bits to B bits. The GC first broadcasts the seed $\mu(t)$ to everyone. Next, the GC associates the following quantity with each user u_j

$$w_j = K_\epsilon(t) + f(K_j, \mu(t)) \pmod{p}. \tag{2.24}$$

The GC generates a degree $n-2$ polynomial $p(z)$ that interpolates the points (z_j, w_j), i.e. $p(z_j) = w_j$. The GC represents $p(z)$ as

$$p(z) = \sum_{i=0}^{n-2} c_i z^i \pmod{p} \tag{2.25}$$

and transmits the message $\alpha_\epsilon(t) = (c_0, c_1, \cdots, c_{n-2})$ to update $K_\epsilon(t)$. This completes the action needed by the GC to update the root KEK, and the session key is then updated using $K_\epsilon(t)$ by transmitting $\alpha_s(t) = E_{K_\epsilon(t)}(K_s(t))$.

A member u_j can calculate $p(z_j) = w_j$ and $f(K_j, \mu(t))$, and hence can recover $K_\epsilon(t)$.

2.4.1 Resistance to Attack

There are two sources of adversaries for a key management scheme. The first type of adversary is an *external* adversary. This type of adversary is not a member of the service, but receives the encrypted content as well as the rekeying messages. In order for the external adversary to cheat the service, he must mount a successful attack against the rekeying messages in order to acquire the session key, which is needed to decrypt the content. The second type of adversary is an *internal* adversary, who is a member that uses the rekeying messages and his knowledge of his keys to attempt to acquire another user's keys. If an internal adversary can successfully acquire another user's keys, he may cancel his membership to the service, and use the compromised keys belonging to another user to enjoy the service without having to pay.

In the polynomial scheme, an external adversary receives α_ϵ as well as $\alpha_s(t)$. In order for the adversary to acquire the SK, he must mount a successful attack against the cipher used in forming the message $\alpha_s(t)$. Careful selection of a strong cipher algorithm that has received serious study, such as Rijndael [14], will make a successful attack of the SK rekeying message unlikely. Even should a successful attack of the SK rekeying message take place, a future update of the SK would require a subsequent successful attack of the SK rekeying message, which is equally unlikely. Hence, a successful attack against the SK rekeying message would only be a short-lived victory for a pirate.

A second method for acquiring the session key is to attack the message α_ϵ. Given the message $\alpha_\epsilon(t)$, and knowledge of a z_j, it is possible that an adversary may calculate w_j. However, the adversary must either determine $K_\epsilon(t)$ or a user's $f(K_j, \mu(t))$ given $w_j = K_\epsilon(t) + f(K_j, \mu(t)) \pmod{p}$. The modulo operation makes w_j independent of either $K_\epsilon(t)$ or $f(K_j, \mu(t))$. Should an external adversary successfully attack $K_\epsilon(t)$, then he may acquire the session key. However, upon the next update of the session key, he must make another successful attack upon the root KEK.

The only method for an external adversary to be able to repeatedly acquire the SK is to mount a successful attack on a user's personal key K_j. This requires successful determination of $f(K_j, \mu(t))$ given w_j, which requires searching a space of order p possibilities, and then successfully attacking the one-way function to acquire K_j. The strength of the one-way function should be as strong as the strength of the encryption used to protect the SK rekeying message.

We now discuss the susceptibility of the original polynomial scheme of [4] to *internal* attacks. In the discussion that follows, we refer the reader to Section 3.1 of [4]. For simplicity, we shall assume that the same key K is being distributed to all of the users. Observe that since the z_j-coordinates are public knowledge, an internal adversary may calculate w_j by evaluating the interpolating polynomial at z_j. With knowledge of w_j, the adversary may use his knowledge of K to determine user u_j's private information. Thus, the polynomial scheme of [4] does not protect the private information of each user, and hence cannot be used more than once. If both the z_j coordinate and the personal key K_j are kept secret, then an adversary's task is to search Z_p for any of the n user's z_j coordinate. This is more difficult for an adversary to attack, but also requires both the server and the clients to store twice as much secret information.

As we shall describe in Section 2.6.3, we chose to pursue a different approach to ensuring the sanctity of each user's private information in order to reduce the communication overhead in our protocol. An inside adversary u_i who desires to calculate another user's key information K_j can calculate $p(z_j) = w_j$, and therefore can calculate $f(K_j, \mu(t)) = w_j - K_\epsilon(t) \pmod p$. However, it is difficult for him/her to calculate K_j given $\mu(t)$ and $f(K_j, \mu(t))$ since f is a parametric one-way function. Additionally, should two or more users collude, their shared information does not provide any advantage in acquiring another user's K_j.

2.4.2 Anonymity Reduces Communication Overhead

The above scheme is used in constructing a protocol primitive in the following section. In the protocol primitive, there is a parent key K_ϵ and a handful of sibling keys K_j that are used to update the parent key. Unlike the example described above, application of the protocol primitive might not use all of the sibling keys to update the parent key. This scenario might occur when the GC knows that a sibling key has become compromised or invalidated.

Suppose that there are a possible sibling keys and that m of those sibling keys are used to update the parent key. In a conventional key distribution scheme, such as [8], the update to the parent key is performed by a rekeying message of the form

$$\alpha = \{E_{K_{j_1}}(K_\epsilon) \| E_{K_{j_2}}(K_\epsilon) \| \cdots \| E_{K_{j_m}}(K_\epsilon)\} \tag{2.26}$$

where j_k denotes the sequence representing the m sibling keys used in updating parent key, and $\|$ denotes message concatenation. In addition to the rekeying message, it is necessary to transmit the amount m of children keys, and the user ID message $\{j_1, j_2, \cdots, j_m\}$, which specifies which portion of the rekeying message a user needs in order to determine the new session key.

The transmission of the user ID message in the conventional scheme reveals which sibling keys are still valid. However, it requires that $\lceil \log_2 a \rceil$ bits to represent m and $m\lceil \log_2 a \rceil$ bits to represent $\{j_1, j_2, \cdots, j_m\}$. The total communication overhead of the conventional scheme is thus $(m + 1)\lceil \log_2 a \rceil$ bits.

The polynomial interpolation scheme creates a composite message that does not require any user ID message, but instead requires the broadcast of the seed $\mu(t)$. The polynomial scheme defines the rekeying message as the output of a function $PolyInt$ which returns the coefficients of the interpolating polynomial, thus

$$\alpha = PolyInt(K, \{z_{j_1}, z_{j_2}, \cdots, z_{j_m}\}, \{K_{j_1}, K_{j_2}, \cdots, K_{j_m}\}, \mu(t)). \quad (2.27)$$

The input to $PolyInt$ is the key K that is to be distributed, the set of valid non-secret ID parameters $\{z_{j_1}, z_{j_2}, \cdots, z_{j_m}\}$, the broadcast seed $\mu(t)$, and the set of valid sibling keys $\{K_{j_1}, K_{j_2}, \cdots, K_{j_m}\}$. Given a valid sibling key and the seed $\mu(t)$, the new parent key can be determined. On the other hand, an invalid sibling key is unable to determine the new parent key.

If the prime p used in the polynomial scheme has the same bit length as the output of one of the encryptions E_K, then the message size of the polynomial scheme will be the same as the rekeying message of the conventional scheme. If B_μ is the bit length of the broadcast seed, then a measure of comparison between the conventional scheme and the polynomial scheme is the difference $(m + 1)\lceil \log_2 a \rceil - B_\mu$. For a single sibling update of the parent node, this difference might favor the conventional approach. The advantage of the polynomial scheme becomes more pronounced when used in a multi-level tree as in Section 2.5.

2.5 Extending to a Scalable Protocol

In the previous section we described the basic scheme for distributing keys during member departures. The basic polynomial interpolation scheme had linear communication requirements during member departures. We now describe a scalable protocol that provides renewal of security levels, handles membership changes, provides a mechanism for reinserting valid members, and allows for the transferal of access rights.

In order to achieve improved scalability, we use a tree-based key hierarchy as depicted in Figure 2.2. In general, the tree can be an a-degree tree.

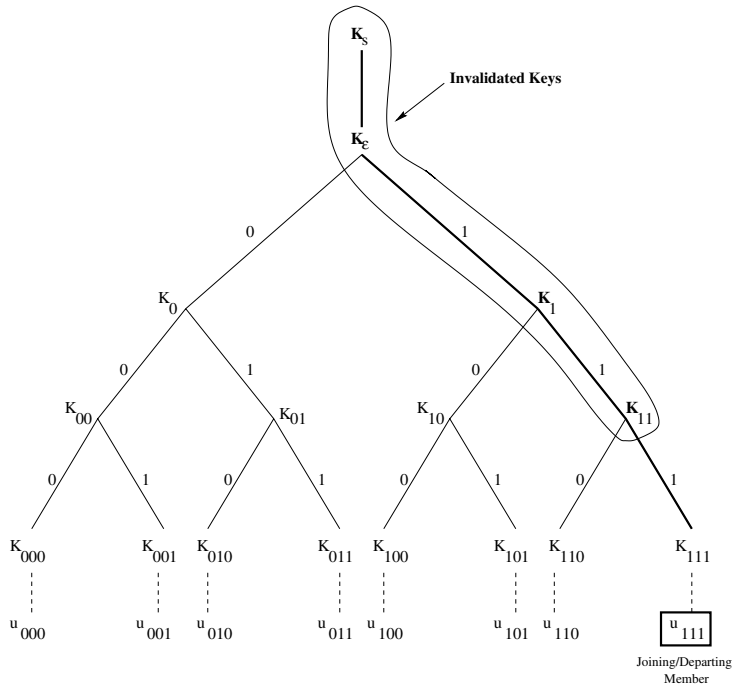


FIGURE 2.2. Tree-based key distribution.

Attached to the tree above the root node is the session key K_s. Each node of the tree is assigned a KEK which is indexed by the path leading to itself. Additionally, each node has a non-secret ID variable z_σ which is used as a non-secret parameter for the $PolyInt$ function. The symbol ϵ is used to denote the root node. Each user is assigned to a leaf of the tree and is given the KEKs of the nodes from the leaf to the root node. Additionally, all users share the session key K_s. For example, user u_{111} is given the keys K_{111}, K_{11}, K_1, K_ϵ, and K_s.

In the protocol that follows, the GC transmits messages to the users via a broadcast channel. It is assumed that each user has an *upstream* channel with minimal bandwidth that is available to convey messages to the GC, such as informing the GC of the intent to depart the service.

The messages that the GC broadcasts to the users must have a standardized structure that is known to all receivers. There are two basic message formats as depicted in Figure 2.3. The first contains three components while the second has five components. The function $B()$ is used to denote the bit length of its operand, thus $B(\sigma)$ is the amount of bits needed to represent σ. The variable Operation ID flags the user which protocol primitive is about to be performed. Only five primitive operations are used, and we may therefore represent Operation ID using a 3 bit string. Table 2.1 maps the primitive operations with their corresponding ID bit string.

TABLE 2.1. Mapping between primitive operations and their corresponding ID bit string.

bit ID	primitive
000	Primitive-1
001	Primitive-2
010	Primitive-3
011	Primitive-4
100	Primitive-5

In the discussion that follows, we assume that the tree has degree a, and that there are L levels to the tree. The amount of multicast group members n is limited by the amount of leaf nodes on the tree. Thus $n \leq a^L$.

2.5.1 Basic Protocol Primitives

We have identified five basic operations needed in building a system that allows for the update and renewal of the key hierarchy. We now describe each case.

1. **Primitive-1(Update SK):** This basic operation uses the current root KEK K_ϵ to update the session key via the rekeying message

$$\alpha = E_{K_\epsilon(t)}[K_s(t)] \tag{2.28}$$

   The message format is depicted in Figure 2.3(a). We assume that the maximum size that α can be is 256 bits, and we therefore need 8 bits to represent $B(\alpha)$. This choice of bit length for α would allow for the use of encryption algorithms with a key size of up to 256 bits.

2. **Primitive-2(Transmit Seed):** The broadcast seed is used in the polynomial scheme to provide protection of secret information. Additionally, it plays a role in reducing the communication overhead associated with flagging the users which part of the message is intended for them. The broadcast of the seed $\mu(t)$ does not require encryption to protect it. The message format for the transmission of the broadcast seed is depicted in Figure 2.3(a). Here $\alpha = \mu(t)$, and $B(\alpha)$ is the amount of bits needed to represent $\mu(t)$. Again, we assume that the maximum size of α is 256 bits, and that 8 bits are used to represent $B(\alpha)$.

3. **Primitive-3(Self Update):** It is often necessary for a node, indexed by the a-ary symbol σ, to have its associated key updated using the key at the previous time instant. Thus we will go from $K_\sigma(t-1)$ to $K_\sigma(t)$ by the following message

$$\alpha = E_{K_\sigma(t-1)}[K_\sigma(t)]. \tag{2.29}$$

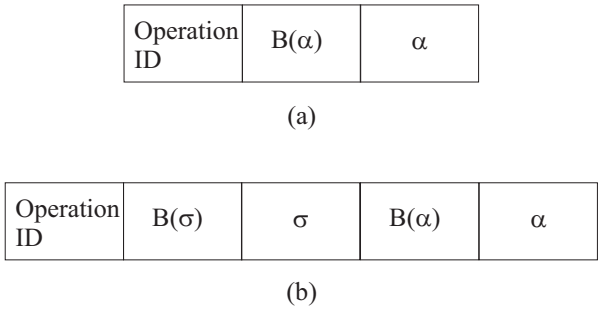


FIGURE 2.3. The two message structures used in the protocol primitives.

In this case, we need to flag the receivers which node is being updated. This requires the transmission of the a-ary representation of the node, as well as the amount of bits needed to represent the node. This is depicted in Figure 2.3(b) by the $B(\sigma)$ and σ components of the message. The rest of the message contains the bit length of the message α and the actual rekeying message α. Since the maximum depth of the tree that needs to be represented is $L-1$ and the tree is an a degree tree, the maximum amount of bits needed to represent σ is $\lceil \log_2 a \rceil (L-1)+1$, where the addition of 1 bit was included to account for the need to represent the empty string ϵ as a possible choice for ϵ. In order to represent $B(\sigma)$, we use $\lceil \log_2 (\lceil \log_2 a \rceil (L-1)+1) \rceil$ bits. The maximum bit length for α is 256 bits, and 8 bits are used to represent $B(\alpha)$.

4. **Primitive-4(Update Parent):** It is also necessary for the children nodes to update the key of their parent nodes. If σ is the symbol representing the parent node to be updated, then the message

$$\alpha = PolyInt(K_\sigma(t), \{z_{Child(\sigma)}(t)\}, \{K_{Child(\sigma)}(t)\}, \mu(t)) \qquad (2.30)$$

is used. Here we have defined the function $Child(\sigma)$ to denote the set of valid children nodes of σ. For example, if we have a binary tree and $\sigma = 00$, and both children nodes are valid, then $Child(\sigma) = \{000, 001\}$. Thus, the message α uses the keys of valid children nodes to update $K_\sigma(t)$. Observe that this message requires that $\mu(t)$ has already been broadcast using Primitive-2, or that the choice of $\mu(t)$ is implicitly known. The message form is depicted in Figure 2.3(b), where again we transfer the bit length of σ and the actual symbol σ to the recipients, followed by the bit length of α and the rekeying message α. We use the same bit allocation for σ and $B(\sigma)$ as in Primitive-3. However, the maximum length for α is aB_{KEK}, and we therefore need $\lceil \log_2 aB_{KEK} \rceil$ bits to represent $B(\alpha)$.

5. **Primitive-5(Reaffirming Parent):** In some operations, it is useful to have a sibling node reaffirm the value of a parent node's key. We

define a function $Par(\sigma)$ to denote the symbol corresponding to the parent of the node indexed by σ. To reaffirm the value of a parent node's key, we transmit the message

$$\alpha = E_{K_\sigma(t)}[K_{Par(\sigma)}(t)]. \tag{2.31}$$

The message form is depicted in Figure 2.3(b), and follows the same structure as used in Primitive-3.

2.5.2 Advanced Protocol Operations

We now describe more advanced protocol operations that can be constructed using the primitive operations described above. In particular, we focus on the operations of an addition to the membership, a deletion of a user from the membership, the reinsertion of a member into the system, and the transferal of access rights from one user to a new user.

Before we proceed, we present a few comments about how the primitive operations can be used to perform periodic renewal of keying material. Primitive-1 provides a method for performing periodic refreshing of the session key. Refreshing the session key is important in secure communication. As a session key is used, more information is released to an adversary, which increases the chance that a SK will be compromised. Periodic renewal of the session key is required in order to maintain a desired level of content protection, and can localize the effects of a session key compromise to a short period of data. Since the amount of data encrypted using KEKs is usually much smaller than the amount of data encrypted by a session key, it is not necessary to refresh KEKs as often. However, the periodic renewal of a KEK can be performed using Primitive-3.

Member Join

In many applications, such as pay-per-view broadcasts and video conferences, the group membership will be dynamic. It is important to be able to add new members to any group in a manner that does not allow new members to have access to previous data. In a pay-per-view system, this amounts to ensuring that members can only watch what they pay for, while in a corporate video conference there might be sensitive material that is not appropriate for new members to know.

Suppose that a new user contacts the service desiring to become a group member. The new client sends the GC a message detailing the client's credentials, such as identity information, billing information, and public key parameters that the GC may use to communicate with the new client. Mutual authentication between the new client and the GC should be performed. A public key infrastructure, such as X.509 certificates [15], may be used for this purpose. Upon verification of the new user's information, the GC assigns the client to an empty leaf of the key tree. For simplicity of

presentation, we assume that the tree has empty slots. If the tree is already full, then the user may either be turned away, or an additional layer must be added to the tree using a separate operation. The GC then issues the new client his keys via a communication separate from the communications sent to the current group members, as well as informing the new user the time at which those keys will become valid.

Meanwhile, the GC updates the current members of the multicast group. Suppose that the GC plans on inserting the new member into the leaf node indexed by the symbol ω. Then the SK as well as the KEKs on the path from the parent node of ω to the root node ϵ must be renewed. The following algorithm describes how this procedure can be accomplished using the protocol primitives. We use the notation $Par^j(\omega)$ to denote the parent function applied j times to ω. Thus $Par^2(\omega)$ is the *grandparent* of ω.

for $j = 1 : L$ **do**
  $\sigma = Par^j(\omega)$;
  Update $K_\sigma(t-1) \rightarrow K_\sigma(t)$ using Primitive-3 ;
end
Update SK using Primitive-1 ;

Member Departure

Members will also wish to depart the service, and must be prevented from accessing future communication. Assume that user u_ω contacts the GC wishing to depart the service. Upon authenticating the user's identity, the procedure that the GC enacts to remove member u_ω and update the keys of the remaining members is

Generate random $\mu(t)$;
Broadcast $\mu(t)$ using Primitive-2 ;
for $j = 1 : L$ **do**
  $\sigma = Par^j(\omega)$;
  Determine valid children of σ: $Child(\sigma)$;
  Update $K_\sigma(t-1) \rightarrow K_\sigma(t)$ using Primitive-4 ;
end
Update SK using Primitive-1 ;

Member Reinsertion

It might often occur that a valid member, denoted by index ω, misses the rekeying messages needed to update the key hierarchy. The client must notify the GC that he missed rekeying messages using an upstream (client

to server) channel. Upon verification of the user's identity, the GC performs the member reinsertion operation, which sends the new user the specific keys he needs to be able to resume the service.

If the service provider has a downstream (server to client) channel available to communicate with the user, then service provider may use this channel to send the needed keys by encrypting them with the user's personal key K_ω. In many scenarios, however, after the initial contact with the service provider, the client has a low-bandwidth channel for upstream communication, and only the broadcast channel available for downstream communication. In these cases, although only a single user needs the rekeying messages, the rekeying messages must be multicast. Since this user has a valid private key K_ω, the GC can start with this key to provide $K_{Par(\omega)}(t)$ to the user. We can then proceed up the tree, using the sibling key to convey the current status of the parent key. The procedure for this operation is as follows:

for $j = 1 : L$ **do**
  $\sigma = Par^j(\omega)$;
  Convey parent key $K_\sigma(t)$ to siblings using Primitive-5 ;
end
Convey current SK using Primitive-1 ;

An added bonus of using the sibling key to convey the current status of the parent key is that other users may observe these rekeying messages to reaffirm the validity of some of their keys.

Transferal of Rights

Suppose that user u_ω wishes to give his rights to another user who is not currently a member. We will denote this new user by u_{ω_B} to indicate that he will take over the keys on the path from ω to the root node. For the purpose of calculating parent and sibling relationships, ω and ω_B are identical, thus $Par(\omega) = Par(\omega_B)$.

In order to transfer access rights, both users must contact the GC, who performs an authentication procedure to verify that the transferal is legitimate. Then, using a secure channel, the GC gives to user u_{ω_B} its own personal key K_{ω_B}. One method for creating a secure channel is to use public key cryptography. K_{ω_B} replaces K_ω on the key tree. All of the keys that belonged to u_ω must be changed to prevent u_ω from accessing content that he has given up the right to access. The procedure for transferring access rights is as follows:

We observe that the algorithm for transferring rights is nearly identical with the algorithm for removing a member from a group. The difference lies in the fact that user u_{ω_B} is considered a valid user, and hence is a valid child of its parent.

Generate random $\mu(t)$;
Broadcast $\mu(t)$ using Primitive-2 ;
for $j = 1 : L$ **do**
  $\sigma = Par^j(\omega_B)$;
  Determine valid children of σ: $Child(\sigma)$;
  Update $K_\sigma(t-1) \rightarrow K_\sigma(t)$ using Primitive-4 ;
end
Update SK using Primitive-1 ;

The procedure for user u_ω to reclaim his access privileges is similar. This time, only user u_ω is required to contact the GC requesting that he regain his access privileges. The GC performs an authentication procedure to guarantee that the identity of u_ω is truthful, and then replaces K_{ω_B} with K_ω. The KEKs and SK are changed according to the above algorithm, with ω replacing ω_B.

2.6 Architectural Considerations

2.6.1 Optimization of Tree Degree for Communication

The amount of communication that a rekeying protocol requires affects the speed at which the rekeying scheme can handle membership changes. It is therefore important to minimize the size of the communication used by the key management scheme. In particular, since the two most important operations performed by a multicast key management protocol are membership joins and membership departures, we shall focus on optimizing the tree degree for these two operations.

In what follows, we present a worst-case analysis of the communication requirements for member join and member departure operations. It is observed that member join and member departure operations lead to conflicting optimality criteria. Since a real system will have to cope with both member joins and member departures, we jointly consider the departure and join operations, and present optimization results when both member join and departure operations are equally weighted.

We refer the reader to the protocol descriptions as well as the message structure in Figure 2.3. We shall denote the degree of the tree by a, and the number of levels in the tree by L. B_{SK} shall denote the bit length of session key, B_{KEK} shall denote the bit length of the key encrypting keys, and B_μ the bit length of the broadcast seed $\mu(t)$.

Worst-Case Analysis

It is easy to see that, for a given tree, the scenario that produces the most communication for the member join operation occurs when one node on

each level from the root to level $L-1$ must be updated. In this case, all of the KEKs on the path from one user to the root must be refreshed. We now calculate the amount of communication needed to update the tree for this worst-case scenario.

The member join operation consists of two types of operations: updating the KEKs, and updating the SK. In order to update the KEKs, we use Primitive-3 L times. Each step of the loop must send the quintuple (operation ID, bit length of update node $B(\sigma)$, node ID σ, bit length of the update message $B(\alpha)$, update message α). The symbol σ starts near the bottom of the tree, and through application of the Parent function moves toward the root of the tree.

In order to represent the symbol σ during the jth iteration of the loop, we need to convert from base a to base 2 and hence $B(\sigma) = \lceil \log_2 a \rceil (L-j)+1$ bits. In addition, we must send $B(\sigma)$, which requires

$$\lceil \log_2 \left(\lceil \log_2 a \rceil (L-1)+1 \right) \rceil$$

bits. Here the addition of 1 was to allow for the need to represent the empty string ϵ as a possible choice for σ. Similarly, in each stage of the loop the rekeying message α has bit length $B(\alpha) = B_{KEK}$ and since we have fixed the maximum key length to be 256 bits, we require 8 bits to represent $B(\alpha)$. The update to the session key requires sending the ID flag, $B(\alpha)$ and α. Therefore, the amount of bits needed to update the session key is $3+8+B_{SK}$. The total amount of bits needed to update the key tree during a member join is

$$\begin{aligned} C_{MJ} &= \left(\sum_{j=1}^{L} \left[3 + \lceil \log_2 (\lceil \log_2 a \rceil (L-1)+1) \rceil + \lceil \log_2 a \rceil (j-1) + 9 + B_{KEK} \right] \right) \\ &\quad +3+8+B_{SK}. \end{aligned}$$

The amount of communication needed in the member departure case can be similarly calculated. The main difference between member join and member departure is that there are three operations: the broadcasting of $\mu(t)$, updating the KEKs, and updating the SK. The most communication occurs when $a-1$ nodes on level L must be used to update the key on level $L-1$, and a nodes are used to refresh each of the remaining KEKs on the path from the departing member to the root node. After appropriately expanding and gathering terms, the communication for the member departure can be found to be

$$\begin{aligned} C_{MD} &= 22 + B_{SK} + B_{\mu} + (La-1)B_{KEK} + (L) \\ &\quad \left(4 + \lceil \log_2 a B_{KEK} \rceil + \lceil \log_2 \left(\lceil \log_2 a \rceil (L-1)+1 \right) \rceil + \frac{(L-1)}{2} \lceil \log_2 a \rceil \right). \end{aligned}$$

We calculated the worst-case amount of communication required to update an a-degree key tree as a function of the number of users n with the amount of tree levels set to $L = \lceil \log_a n \rceil$. In our calculations, we chose $B_{SK} = B_{KEK} = B_\mu = 64$ bits. We chose to use 64 bits as the key size since such a key length can provide strong levels of security when used with some ciphers, such as RC5 [13]. The amount of communication required for different choices of the degree of the tree a during a member join is depicted in Figure 2.4(a). This figure shows the general trend that less communication is required during member join operations if we use a higher degree tree. On the other hand, Figure 2.4(b) shows the amount of communication needed during the worst case of a member departure operation. In this case, the larger tree degrees are definitely not advantageous. It is also evident that a binary tree is not optimal when considering member departure. In fact, the values of $a = 3$ and $a = 4$ appear to be the best choice, with optimal choice fluctuating depending on n.

Joint Departure-Join Optimization

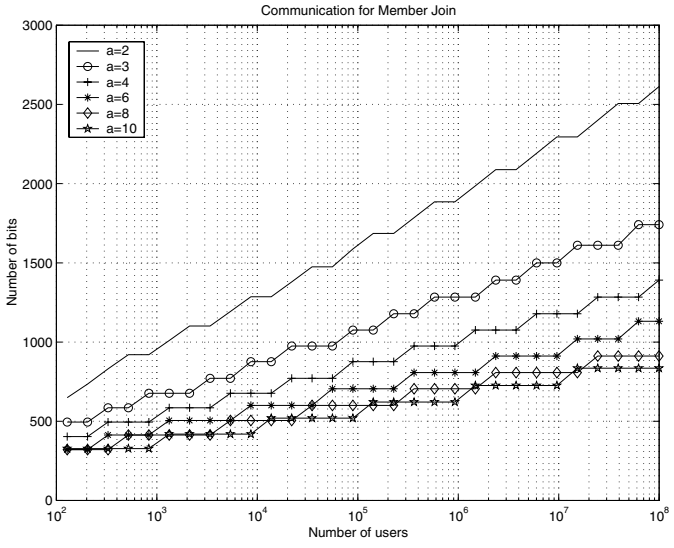
In some application scenarios the key tree might start out relatively empty, and the amount of member join operations would be greater than the amount of member departure operations. In this case, the membership grows towards the tree capacity, and the communication required for the member join operation is more critical than the communication for member departure. On the other hand, some scenarios might start out with a nearly full key tree, and the member departure operation would outweigh the member join operation.

We therefore would like a communication measure that runs the gamut between the two extremes of just the member join communication, and just the member departure communication. This can be accomplished by considering the convex combination of C_{MJ} and C_{MD}.

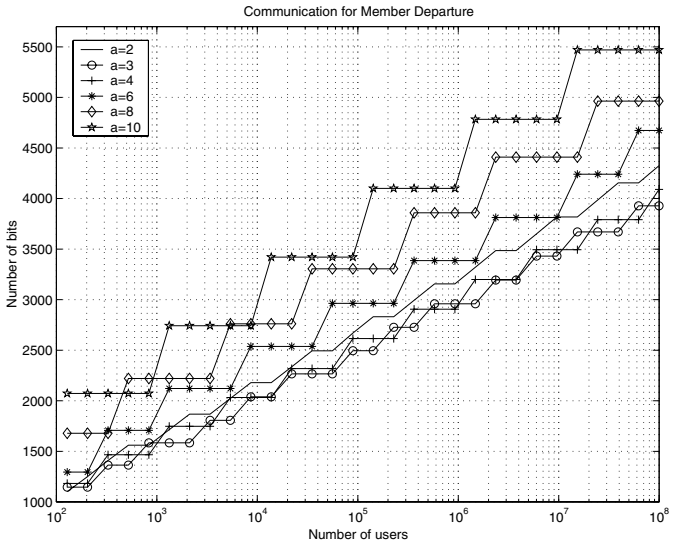
Let λ denote the probability of a member departure operation, and assume that $1 - \lambda$ is the probability of a member join operation, then the combined communication measure C_C given by

$$C_C = \lambda C_{MD} + (1 - \lambda) C_{MJ} \tag{2.32}$$

weights the member departure and member join operations according to their likelihood. For example, when $\lambda = 0$ the emphasis is entirely placed on the member join operation, while $\lambda = 1$ corresponds to when the emphasis entirely placed on the member departure operation. The case of $\lambda = 0.5$ corresponds to equal emphasis on the two operations, which is depicted in Figure 2.5. From this figure, we see that the choice of $a = 4$ stands out as the best choice for $n > 10000$ when equally weighting the member join and member departure operation.



(a)



(b)

FIGURE 2.4. (a) The amount of communication C_{MJ} required during member join operations for different tree degrees a and different amounts of users n. (b) The worst case amount of communication C_{MD} required during member departure operations for different tree degrees a and different amounts of users n.

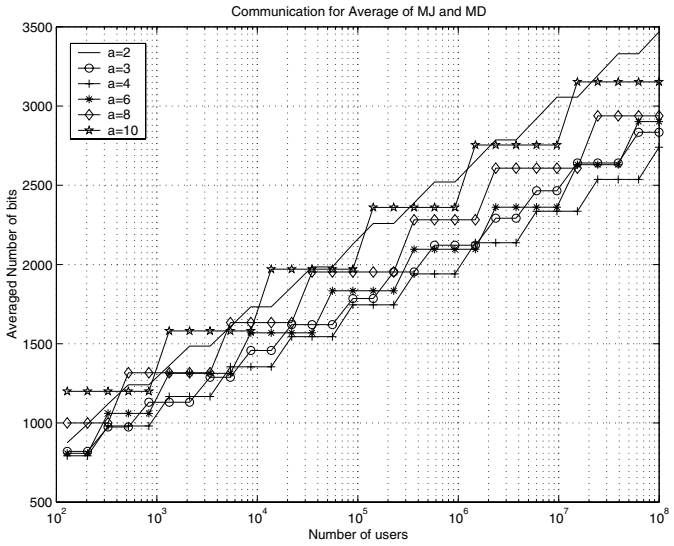


FIGURE 2.5. The average of C_{MD} and C_{MJ} for different tree degrees a and different amounts of users n.

2.6.2 Binomial Occupancy Model

Since it is very difficult to calculate the amount of communication needed during membership changes when a specific amount of users n are placed on the tree, we have devised a stochastic model that allows one to study the behavior of the system when there are varying amounts of occupancy. We assume that the leaf nodes of the a-degree key tree with L levels are occupied according to i.i.d. Bernoulli distributions with a probability of occupancy q_L. This implies that the occupancy n is modeled according to a binomial distribution with mean occupancy $q_L a^L$ and variance $q_L(1-q_L)a^L$. Hence, when q_L is higher, the tree is on average at higher occupancy.

We first calculate the average amount of communication required for member join when the probability of a node being occupied is q_L. Let τ_a denote the a-ary representation of the joining member. We may denote the siblings of τ_a by $\tau_1, \tau_2, \cdots, \tau_{a-1}$. Define the random variable Z_{L-1} as

$$Z_{L-1} = \begin{cases} 1 & \text{if any } \tau_k \text{ is occupied} \\ 0 & \text{if no } \tau_k \text{ are occupied} \end{cases}.$$

Since the τ_k are occupied with a probability of q_L, we have $P(Z_{L-1} = 1) = 1 - (1 - q_L)^{a-1}$, and the expected value of Z_{L-1} is given by $E(Z_{L-1}) = 1 - (1 - q_L)^{a-1}$.

We may perform a similar procedure for the other levels. We denote the j-siblings as those nodes τ such that $Par^j(\tau) = Par^j(\tau_a)$. For level $L - j$,

we may define the random variable Z_{L-j} as

$$Z_{L-j} = \begin{cases} 1 & \text{if any } j\text{-sibling node of } \tau_a \text{ is occupied} \\ 0 & \text{if no } j\text{-sibling nodes of } \tau_a \text{ are occupied} \end{cases}.$$

In this case, $P(Z_{L-j} = 1) = 1 - (1 - q_L)^{a^j - 1}$, and the expected value of Z_{L-j} is given by $E(Z_{L-j}) = 1 - (1 - q_L)^{a^j - 1}$.

The average communication requirements for member join can be derived as

$$\begin{aligned} \overline{C_{MJ}} \quad = \quad & \left(\sum_{j=1}^{L} (1 - (1 - q_L)^{a^j - 1}) \Big[12 + \lceil \log_2(\lceil \log_2 a \rceil (L-1) + 1) \rceil \right. \\ & \left. + \lceil \log_2 a \rceil (L - j) + B_{KEK} \Big] \right) + 11 + B_{SK}. \end{aligned}$$

We now apply the model to calculating the average amount of communication needed during member departure. Again suppose that the departing member is indexed by the a-ary symbol τ_a. Label the siblings of τ_a by $\tau_1, \tau_2, \cdots, \tau_{a-1}$, and define the random variable X_k by

$$X_k = \begin{cases} 1 & \text{if } \tau_k \text{ is occupied} \\ 0 & \text{if } \tau_k \text{ is not occupied} \end{cases}.$$

Let us define $Y_L = \sum_{k=1}^{a-1} X_k$, which is the random variable corresponding to the amount of occupied sibling nodes of τ_a at level L. The probability that i sibling leafs at level L are occupied is given by

$$P(Y_L = i) = \binom{a-1}{i} q_L^i (1 - q_L)^{a-1-i}. \tag{2.33}$$

Y_L is thus a binomial random variable with expected value $E(Y_L) = (a - 1)q_L$. Hence, the average number of nodes to be updated at level L is $(a - 1)q_L$.

At level $L - 1$, we know that the parent node of the departing member will automatically be used in updating the next higher level. Since the probability of a node at level L being occupied is q_L, the probability that a node on level $L - 1$, other than $Par(\tau_a)$, being occupied is

$$q_{L-1} = 1 - (1 - q_L)^a. \tag{2.34}$$

This time, we may denote the siblings of $Par(\tau_a)$ by $\tau_1, \tau_2, \cdots, \tau_{a-1}$. Again, we define the random variable X_k by

$$X_k = \begin{cases} 1 & \text{if } \tau_k \text{ is occupied} \\ 0 & \text{if } \tau_k \text{ is not occupied} \end{cases}.$$

We now define the random variable Y_{L-1} to be the amount of sibling nodes of $Par(\tau_a)$ that are occupied, and we find that $E(Y_{L-1}) = (a-1)q_{L-1}$. Since we must also include $Par(\tau_a)$ in the updating we must add one. Thus, the expected number of nodes on level $L-1$ that must be updated is $1 + (a-1)q_{L-1}$. We may similarly perform this calculation for level j, where $q_j = 1 - (1 - q_{j+1})^a$, and the expected number of nodes on level j to be updated is $1 + (a-1)q_j$.

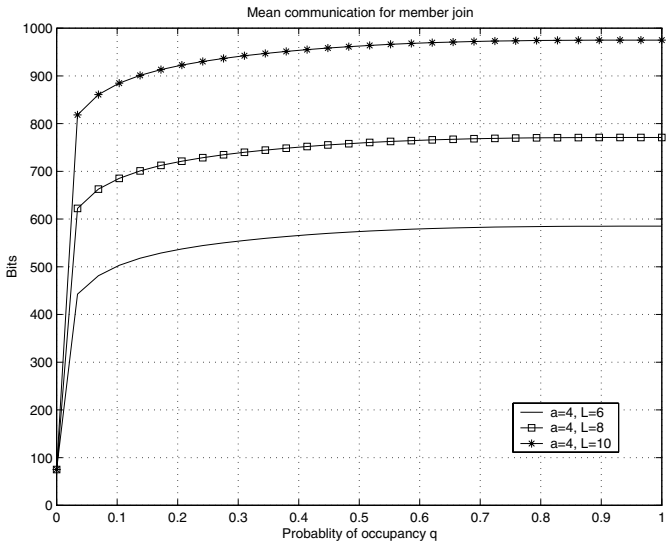
In order to calculate the average amount of communication for the member departure operations, we must consider both the expected amount of communication associated with the overhead and the payload of the message. The average communication for the overhead consists of the amount of communication needed to send the operation id, the node id, and the bit length of the update message. This calculation can be done using the expected value of Z_{L-j}. The average communication for the payload is calculated using the expected number of nodes on level j to be updated. The average amount of communication for n users on an a-degree tree with L levels is therefore given by

$$\begin{aligned}\overline{C_{MD}} \quad = \quad & 22 + B_\mu + B_{SK} + q_L(a-1)B_{KEK} + \left(\sum_{j=1}^{L-1}\left(1 + (a-1)q_j\right)B_{KEK}\right) \\ & + \left(\sum_{j=1}^{L}\left(1-(1-q_L)^{a^j-1}\right)\left(4 + \lceil \log_2(\lceil \log_2 a\rceil(L-1)+1)\rceil \right.\right. \\ & \left.\left. + \lceil \log_2 a\rceil(L-j) + \lceil \log_2 aB_{KEK}\rceil\right)\right).\end{aligned}$$

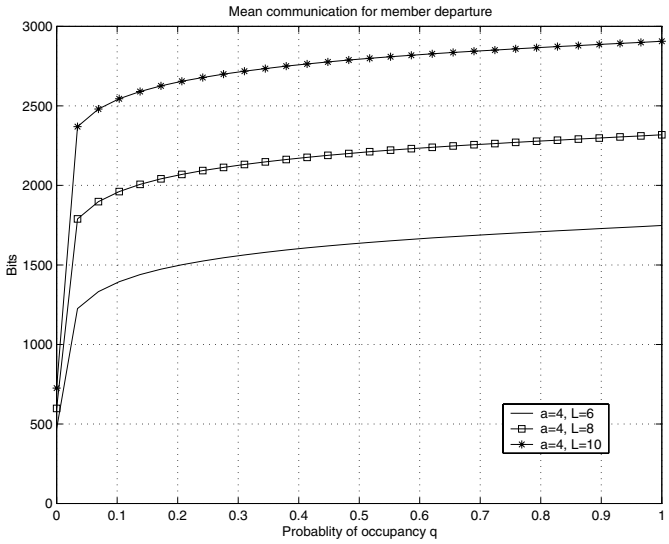
We calculated the mean message size for member join and member departure operations as parameterized by q when the tree degree is $a = 4$ and there are 6, 8 and 10 levels. The key sizes were chosen to be $B_{SK} = B_{KEK} = B_\mu = 64$ bits. In Figure 2.6, we have indicated the mean communication as a function of q. One can see that the expected communication rapidly increases as the probability q becomes slightly greater than 0. In the member join operation, the communication levels off to a flat plateau as the probability of occupancy increases. For the member departure operation, the mean communication also increases rapidly for $q < 0.1$, but then grows less dramatically for higher q. From these two curves, we can infer that a key tree which is roughly half occupied does not have considerably different communication requirements than the worst-case communication requirements, which occur when $q = 1$. This supports our use of the worst-case scenarios for optimizing the tree degree.

2.6.3 Communication Overhead

Earlier we mentioned that one motivation for using the broadcast seed is that it reduces the amount of communication overhead associated with



(a)



(b)

FIGURE 2.6. The expected amount of communication for a degree 4 tree with 6, 8, and 10 levels as a function of the probability q that a leaf node is occupied. (a) Member Join, (b) Member Departure.

notifying to the users which rekeying messages are intended for them during member departures. We now explore this concept in the framework of a tree-based scheme.

Consider an a degree tree with n users. In a general tree-based scheme, when a user departs, all of the keys on the path from the departing member's leaf to the root key must be updated. To update a key associated with a particular node σ, we must determine the keys associated with populated children nodes. These keys are then used to encrypt the update, and the rekeying message is then of the form:

$$\alpha = \{E_{K_{j_1}}(K_\sigma) \| E_{K_{j_2}}(K_\sigma) \| \cdots \| E_{K_{j_m}}(K_\sigma)\}. \tag{2.35}$$

Here we have used the sequence $\{j_k\}$ to denote index the symbols of the valid children nodes. In addition to sending the rekeying message, it is necessary to send the number of valid children nodes m, and the sequence $\{j_1, j_2, \cdots, j_m\}$.

The worst case scenario for communication overhead in updating a tree is when a of the children nodes are used to update each parent node. In this case, the communication overhead required is

$$C_O = (a+1)\lceil \log_2 a \rceil \lceil \log_a n \rceil. \tag{2.36}$$

This equation is obtained by considering both the communication needed to send the amount of valid children nodes, and the symbols for each valid child node.

This amount of communication overhead was calculated for different group sizes n and different tree degrees a. The resulting amount overhead is depicted in Figure 2.7. In this figure we have also drawn a baseline corresponding to $B_\mu = 64$ bits, which is the amount of communication overhead required if one uses the Member Departure protocol of Section 2.5. Examining the case of $a = 4$, which corresponds to the optimal value of the tree-degree as previously determined, shows that for values of $n > 10000$, the Member Departure protocol described in this chapter requires less communication overhead in the worst case scenario. Additionally, observe that if we use a higher degree tree, which is better suited to scenarios where more users are joining than departing, then the efficiency of the Member Departure protocol is even more pronounced.

The use of a broadcast seed can gain further improvement if we choose to use $\mu(t) = K_s(t-1)$. In this case, the broadcast seed does not have to be sent since it is known by the remaining users. Therefore, there is no communication overhead associated with updating during member departure, and we may consider the baseline at $B_\mu = 0$. In this case, the benefits of using a broadcast scheme becomes even more pronounced.

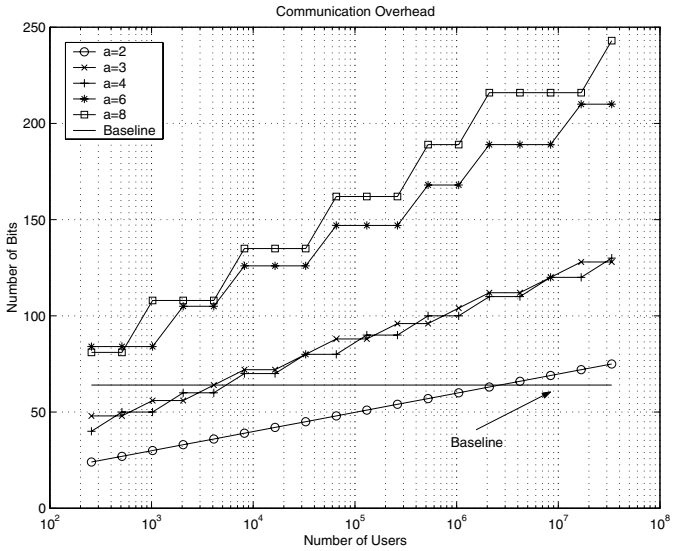


FIGURE 2.7. The worst-case member departure communication overhead required in a conventional tree-based rekeying for different tree degrees versus the baseline communication required when using the polynomial interpolation scheme. The baseline communication corresponds to $B_\mu = 64$ bits.

2.6.4 Computational Complexity

We have seen that one advantage of broadcast schemes is that they reduce the amount of communication overhead associated with sending flagging messages. It should be apparent that a message form like equation (2.26) takes less computation to form than a message form like equation (2.27) assuming that calculating $E_K(K_\sigma)$ has comparable computation as $f(K_\sigma, \mu(t))$. Hence, to rekey using our message form requires more computation than when using a conventional rekeying message structure.

In the scheme we have described in this chapter, we have L levels of KEKs to update. At each level of the tree we must calculate the coefficients of a degree $a-1$ interpolating polynomial, except at the bottom level where we must calculate the coefficients of a degree $a-2$ polynomial.

In order to calculate the coefficients of a s-degree interpolating polynomial, we use the Newton form of the interpolating polynomial [16]. Algorithm 1 is a modification of the polynomial interpolation algorithm of [17], which can be used to determine the coefficients β_j of the s-degree polynomial that interpolates the points $(z_j, g_j) \in Z_p \times Z_p$, where $j \in \{0, 1, \cdots, s\}$. The algorithm writes the β_j values into the input array values g_j.

This algorithm requires addition, multiplication, inversion, and modulo operations to take place modulo p. The most intensive operation of these is that of inverting a number. Assume that the prime p is chosen to have B bits, then the amount of bits operations needed to calculate the inverse

for *k=0:s-1* **do**
  for *j=s:-1:k+1* **do**
    $g(j) = (g(j) - g(j-1))(z(j) - z(j-k-1))^{-1} \pmod{p}$
  end
end
for *k=s-1:-1:0* **do**
  for *j=k:s-1* **do**
    $g(j) = g(j) - g(j+1)z(k) \pmod{p}$
  end
end

Algorithm 1: Algorithm for determining the coefficients of an interpolating polynomial.

of a number modulo p using the Euclidean algorithm is $\mathcal{O}(B^3)$ [18]. The above algorithm requires $\frac{s(s+1)}{2}$ inversions in order to determine a degree s interpolating polynomial. Therefore, the amount of bit operations needed to update an L level degree a key tree using the polynomial interpolation scheme is $\mathcal{O}(a^2LB^3)$.

2.7 Chapter Summary

In order to address the problem of managing keys for securing multicasts, we proposed a framework that is suitable for dynamic group environments. Advanced protocol operations that update the keys during member joins, member departures, and the transferal of access rights were built using basic protocol operations which we call protocol primitives.

We described several desirable features for a multicast key management scheme, and which our scheme satisfied. In particular, our architecture provides a method for renewing session keys and key encrypting keys needed to control access to content. By using either the basic protocol operations, or more advanced protocol operations, the session key or key encrypting keys can be refreshed when a key's lifetime expires due to age or changes in membership. It is also evident that if users were to collude, they would not be able to figure out keys that they did not have. Users may survive accidents or move across terminals by sending a request for reinsertion to the server, upon which the server performs the member reinsertion protocol operation. We also provided a description of a protocol operation that would allow users to transfer their access rights to other parties. The server can revoke access to an individual by using the member departure operation to remove the member from the key hierarchy. Finally, our protocol uses a tree-structured key hierarchy in order to achieve desirable communication requirements during changes in the group membership.

A novel feature of this scheme is that it uses polynomial interpolation in conjunction with a broadcast seed to handle member departure operations. We studied the communication associated with performing member join and member departure operations. It was observed that higher tree degrees are best for member join operations, whereas a tree degree of 3 or 4 was best for the member departure operation. When equally weighting the join and depart operations, a degree 4 tree stood out as optimal. The communication overhead of the polynomial interpolation scheme is reduced in comparison to a model conventional scheme. We provided a comparison between the communication overhead of our scheme and the overhead of an example conventional scheme that used ID messages to flag the users which parts of the rekeying message were intended for them. As group size and tree degree increased, the communication overhead for the conventional scheme increases and ultimately becomes more burdensome than sending the broadcast seed. For example, when the group size was $n = 100000$ and the tree degree was $a = 4$, the communication overhead in the conventional scheme was approximately 25 % more than the overhead associated with a broadcast seed of size $B_\mu = 64$ bits. Finally, if one uses the previous session key $K_s(t-1)$ as the seed $\mu(t)$, then no communication overhead is associated with our protocol during member departures.

We presented a study of the communication needed when using our architecture to perform member joins and member departures. These two operations are the most important operations that a multicast server will have to face when operating in dynamic environments. The communication requirements of the member join and member departure operations lead to conflicting tree design considerations. By explicitly computing these two quantities as functions of the degree of the tree and computing the communication overheads, we studied the tree selection criterion. From our computations, the communication during a member join is reduced when using a higher degree tree, while the optimal tree degree for a member departure is either $a = 3$ or $a = 4$. We considered the average of the communications for the two operations, which gave strong support to choosing $a = 4$ as the optimal tree degree. We presented a stochastic population model that allows one to study the mean behavior of our architecture for varying amounts of users. It is observed that for both the join and departure operation, the amount of communication needed to update the key tree rapidly increases as the tree approaches 10% population. Above 10% occupancy, the communication needed for both operation stabilizes. We also examined the computational requirements of the tree-based rekeying schemes using polynomial interpolation.

http://www.springer.com/978-0-387-68846-6

Network-Aware Security for Group Communications
Sun, Y.; Trappe, W.; Liu, K.J.R.
2008, XVIII, 304 p., Hardcover
ISBN: 978-0-387-68846-6