An Information-Theoretic Approach for Design and Analysis of Rooted-Tree-Based Multicast Key Management Schemes

Radha Poovendran, *Member, IEEE,* and John S. Baras, *Fellow, IEEE*

***Abstract*—Recent literature presents several seemingly different approaches to rooted-tree-based multicast key distribution schemes [6]–[8], [28], [29] that try to minimize the user key storage while providing efficient member deletion. In this paper, we show that the user key storage on rooted trees can be systematically studied using basic concepts from information theory. We show that the rooted-tree-based multicast key distribution problem can be posed as an optimization problem that is abstractly identical to the optimal codeword length selection problem in information theory. In particular, we show that the *entropy of member deletion statistics* quantifies the optimal value of the average number of keys to be assigned to a member. We relate the sustainable key length to statistics of member deletion event and the hardware bit generation rate. We then demonstrate the difference between the key distribution on rooted trees and the optimal codeword-length selection problem with an example of a key distribution scheme that attains optimality but fails to prevent user collusion [7], [8].**

***Index Terms*—Collusion, entropy, member deletion, multicast security.**

I. Introduction

MULTICAST is a preferred communication model when an identical message has to be delivered to multiple intended receivers [24]. Multicast communication reduces overheads of the sender as well as the network medium. Many new real-time applications, such as stock quote updates, Internet newscast, and distributed gaming, can all benefit from multicast communication. Most of the commercial models will have a single sender and multiple receivers. This is the model of interest in this paper.

Ability to secure group communications from the rest of the world is an important issue that needs to be addressed for the wide deployment of many multicast (also noted in literature as restricted broadcast) applications [1], [6], [10], [25], [26], [28], [29]. Use of cryptography is a practical approach for securing multicast communications over an untrustworthy network medium [3], [18], [27]. When cryptography is used for securing communications, a session-encrypting key (SEK) is used to encrypt the data.

Since the data is distributed to multiple receivers, in order to reduce the amount of encryption at the sender node and to minimize the amount of packets over the networks, every intended receiver as well as the sender should share an identical SEK. In order to ensure that only the valid members of the group have access to the communications, the SEK needs to be changed whenever: a) the lifetime of the SEK expires, b) there is a change in membership of the group, or c) one or more members are compromised.

The SEK needs to be updated under membership change for the following reasons: a) when a new member joins, to ensure that the new member has no access to the past communication of the group and b) when a member departs or is deleted, to ensure that the departed or deleted member does not have access to future communications.

Ensuring that only the valid members of the group have the SEK at any given time instance is the key management problem in secure multicast communications [28], [29].

Since the group is distributed over the untrustworthy network, whenever the SEK is invalidated, there needs to be another set of keys called the key-encrypting keys (KEKs) that can be used to encrypt and transmit the updated SEK to the valid members of the group.

Hence, the key management problem reduces to the problem of distributing the KEKs to the members such that at any given time instant all the valid members can be securely reached and updated with the new SEK.

Developing *efficient* KEK distribution algorithms and protocols for secure multicast has been an active area of research for the past three years. Among several techniques that are available, a virtual tree-based approach, independently derived by Wallner, Harder, and Agee [28], and Wong, Gouda, and Lam [29] has led to a family of key distribution schemes for secure multicast [4]–[8]. This paper shows that these virtual tree-based KEK distribution models can be studied using basic concepts from information theory.

In this paper, we show that the rooted-tree-based KEK distribution problem can be studied as a convex optimization problem.We then show that this convex optimization problem is abstractly identical to the optimal codeword length selection problem from information theory. The main result is that the

Manuscript received June 14, 1999; revised February 7, 2001. This work was supported in part by the U.S. Army Research Laboratory under Contract ATIRP-DAAL01-96-2-002, by DARPA under Contract F30602002510, by the National Science Foundation under Contract ANI-0093187, and by NSA. The material in this paper was presented in part at CRYPTO'99, Santa Barbara, CA, August 1999 and at the 1999 IEEE Workshop on Information Theory and Networking, Metsovo, Greece, June 1999.

R. Poovendran is with the Department of Electrical Engineering, University of Washington, Seattle, WA 98195 USA (e-mail: radha@ee.washington.edu).

J. S. Baras is with the Department of Electrical and Computer Engineering and Institute for Systems Research, University of Maryland, College Park, MD 20742 USA (e-mail: baras@isr.umd.edu).

Communicated by D. R. Stinson, Associate Editor for Complexity and Cryptography.

Publisher Item Identifier S 0018-9448(01)08969-6.

optimal KEKs to be updated by the group controller (GC) under a member deletion is related to the *entropy of member deletion statistics*. We also show that the rooted-tree-based KEK distribution schemes in [28], [29] can be derived as a $\max$–$\min$ solution of our optimization. We also show where the optimal codeword length selection problem and the KEK distribution problem differ by showing that the Kraft inequality that is necessary and sufficient [9] for optimal codeword-length selection problem is only a necessary condition for the KEK distribution problem.

We note that information theory has been used in the past to study various aspects of cryptographic problems [15], [16], [19], [26]. However, to our knowledge, this is the first paper to use the basic results from information theory to analyze multicast key distribution on the rooted trees.

The paper is organized as follows: Section II presents a review of the non-tree-based key distribution approaches for secure multicast communication and motivates the need for *efficient* solutions. This section also shows that while the member addition can be handled well by these methods, the member deletion and illegal collaborations among members, called user collusion in the literature [11], [14], poses challenges as group size increases. Section III presents rooted-tree-based KEK distribution schemes and shows how the member deletion is handled *efficiently* by the rooted trees. Section IV presents the necessary definitions and observations that will allow us to formulate the KEK distribution on the tree as an optimization problem. Section V presents our formulation of the problem based on member deletion statistics and shows that the KEK distribution on the rooted trees is an optimization problem. Section VI presents a concrete example of a rooted tree that is *efficient* with respect to user storage but has user collusion problem.

Throughout this paper, we will use the term Group Controller (GC) to denote the entity that manages the key distribution problem. We will denote the group size by N. We now describe the non-tree-based KEK distribution schemes.

II. Non-Tree-Based KEK Distribution Schemes

We noted that since every member of the group shares the same SEK, when the group membership changes, the SEK needs to be updated. One inefficient but secure way to update the SEK is to allow the GC to share a unique KEK with every member. When there is a change in group membership, the GC uses the individual KEK of every valid member to encrypt the new SEK and update all the valid members. The cost of SEK updates grows linearly with the group size N.

In [13], a proposal called Group Key Management Protocol (GKMP) was proposed. In this scheme, the entire multicast group shares the same SEK and a group KEK. It is also implicitly assumed that every member of the group shares a unique KEK with the GC. When a new entity sends a "join" request, the GC first establishes a shared KEK denoted KEK$'$ unique to that member. The GC generates a new SEK denoted SEK$''$ and a KEK$''$ that will be distributed to the entire group. The GC then encrypts the SEK$''$ and the KEK$''$ with the old KEK of the entire group and transmits. Every member except the newly admitted one can decrypt the message and extract the new SEK$''$ and KEK$''$. The GC then encrypts the SEK$''$ and KEK$''$ with KEK$'$ and transmits. This allows the new member to update the SEK$''$ and KEK$''$. Hence, the member join event requires two encryptions under GKMP. When a member is deleted, the current SEK and the KEK are known to the deleted member. Hence, the old group KEK cannot be used to update the new SEK and group KEK. The GC has to individually contact every member and update the SEK and the group KEK. Hence, under member deletion, the GC needs to perform $(N-1)$ computations and communications whereas every member stores only three keys.

Another extreme of the key distribution is to generate 2^N subsets of members and assign a unique SEK to every subset. Every member will need to store 2^{N-1} SEKs. When a member is deleted, the index of the subset that contained all the valid members except the deleted member is identified and transmitted. All the members of that subset use the unique SEK of that subset and continue the communication. For this model, under member deletion, there is no need for update key encryption. When a new member joins, the storage for every member goes up by a factor of two under this model.

From the previous two examples, we note that there is a tradeoff between the update messages under member deletion and the number of keys to be stored.

In [20], a hierarchical clustering scheme was proposed to reduce the amount of key update messages under member deletion. In this method, a given group is divided into clusters that are locally controlled by a cluster controller (CC). Each cluster has its own SEK and KEKs. Hence, any membership change is contained within the cluster. This strategy relies on a set of "trustable" intermediate nodes that can act as CCs. The CCs control the intercluster communications.

In [1], a key distribution scheme based on core-based trees (CBT) was proposed. The tree structure is also assumed to be a physical one in nature. Hence, every intermediate tree node represents a member of the group. Any group member of the tree is permitted to distribute the SEK and KEK shared by the entire group. Clearly, allowing any member to distribute the SEK and KEK reduced the computational overheads at the GC. However, permitting any member to distribute the SEK and KEK required placing same amount of "trust" on every member. Such an assumption is too restrictive in practice.

Two independent seminal papers [28], [29] presented rooted-tree-based KEK distribution schemes that are based on building a virtual or logical tree hierarchy for key distribution. These rooted trees do not depend on any trust assumption about the logical node. The virtual tree-based solutions have led to a family of solutions with seemingly different *efficient* values for the number of keys to be updated under member deletion. We now present the review of the rooted-tree-based key distribution.

III. Review of the Rooted-Tree-Based Key Distribution Schemes

The first use of rooted-tree-based key distribution approach for secure multicast communication was independently proposed in [28] and [29]. A rooted binary tree was used in [28]

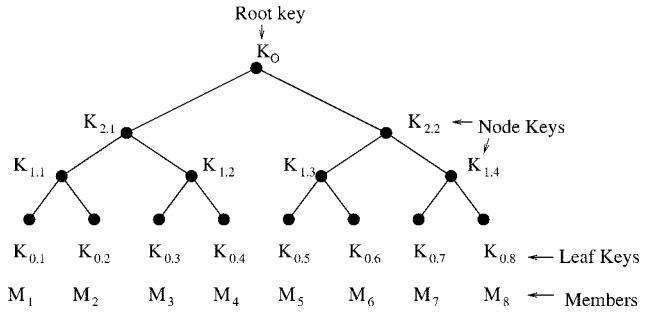

Fig. 1. The logical key tree of [6]–[8], [28], and [29].

and key graphs were used in [29]. Both these approaches construct a logical tree or a key graph based on the size of the group without making any assumptions regarding the relationship among the keys used. The key storage requirements of the GC of these two schemes grow as $\mathcal{O}(N)$ while the key update communications as well as the storage requirements of the users grow[1] as $\mathcal{O}(\log n)$. We now discuss the rooted tree-based key distribution schemes.

A. Distribution of Keys on the Tree

Fig. 1 presents a rooted binary key distribution tree for a group with eight members. The logical tree is constructed such that each group member is assigned to a unique leaf node of the tree. Every node of the logical tree is assigned a KEK. The set of KEKs assigned to the nodes along the path from a leaf node to the root are assigned to the member associated with that leaf node. For example, member M_1 in Fig. 1 is assigned key encrypting keys $\{K_O, K_{2.1}, K_{1.1}, K_{0.1}\}$.

Since the root key K_O is shared by all the members, if there is no change in group membership, K_O can be used to update the SEK for all the members. We note that some of the tree-based key distribution schemes [8] use the root key as the session key as well as the group KEK. In the security area, use of same key for different functionality is often prohibited to prevent security breaches.

The tree-based structure also induces a natural hierarchical grouping among the members. By assigning the members to appropriate nodes, the GC can form desired hierarchical clusters of members and selectively update, if needed, the keys of the group. For example, in Fig. 1, members M_5, M_6, M_7, and M_8 exclusively share the key $K_{2.2}$. The GC can use the key $K_{2.2}$ to selectively communicate with these members. The GC may decide such clustering of the members on the tree based on application-specific needs. Indeed, the GC can reach all 2^N subsets of users in a group of size N. In order to be able to selectively disseminate information to a subset of group members, the GC has to ensure that the common key assigned to a subset is not assigned to any member not belonging to that subset.

Using the notation $\{m\}_K$ to denote the encryption of message m with key K, and the notation $A \longrightarrow B$: $\{m\}_K$ to denote the secure exchange of message m from A to B, GC can selectively send a message m to members $M_5, \ldots, M_8$ by the following transmission:

$$\text{GC} \longrightarrow M_5, M_6, M_7, M_8\text{: } \{m\}_{K_{2.2}}.$$

If, however, the key $K_{2.2}$ is invalidated for any reason, GC needs to update the invalidated key $K_{2.2}$ before being able to use a common key for members M_5, M_6, M_7, and M_8. The tree structure allows this operation to be completed with two update messages. The GC can do so by first generating a new version of $K_{2.2}$, denoted $\hat{K}_{2.2}$, and then performing two encryptions, one with $K_{1.3}$ and the other with $K_{1.4}$. The following two messages are needed to update key $\hat{K}_{2.2}$ to the relevant members of the group:

$$\text{GC} \longrightarrow M_5, M_6\text{: } \{\hat{K}_{2.2}\}_{K_{1.3}}$$
$$\text{GC} \longrightarrow M_7, M_8\text{: } \{\hat{K}_{2.2}\}_{K_{1.4}}.$$

B. Member Deletion in Rooted Trees

Since the SEK and the root KEK K_O are common to all the members in the group, they need to be invalidated each time a member is deleted. Also, since more than one KEK may be shared with other valid members, they need to be updated. In the event where there is bulk member deletion, the GC has to: a) identify *all* the invalid keys, b) find the minimal number of valid keys that can be used to transmit the updated keys, and c) update the valid members with the new keys.

The general principle behind the member deletion for a d-ary ($d = 2$ in Fig. 1) is discussed below using member M_1 in Fig. 1 as an example. Member M_1 is indexed by the set of four keys $\{K_O, K_{2.1}, K_{1.1}, K_{0.1}\}$. Deleting member M_1 leads to invalidating these four keys and SEK, generating $\log_d N$ new key encrypting keys, and updating the appropriate valid members who shared the invalidated keys with member M_1. When M_1 is deleted, the following updates are necessary: a) all the member need new root key K_O and new session key SEK, b) members M_2–M_4 need to update $\{K_{2.1}\}$, and c) member M_2 needs to update $\{K_{1.1}\}$.

The following observations can be made toward the rooted-tree-based key distributions.

- Since each member is assigned $(2 + \log_d N) = \log_d N d^2$ keys, deletion of a single member requires $(2 + \log_d N)$ keys to be invalidated.
- Since each node of the rooted tree is assigned a key, and every member shares $\log_d N$ nodes with at least one more member, the total number of KEKs to be updated under a member deletion is $\log_d N$.
- For a d-ary tree with depth $h = \log_d N$, the GC has to store

$$1 + 1 + d + d^2 + \cdots + d^h = \frac{d(N+1) - 2}{(d-1)}$$

number of keys. Setting $d = 2$ leads to the binary tree for which the required amount of storage is $\frac{2(N+1)-2}{2-1} = 2N$. This result can be independently checked by noting that a binary tree with N leaves has $2N - 1$ nodes. Hence, the GC has to store the SEK as well as $(2N - 1)$ KEKs, leading to $2N$ keys.

[1]Recent literature presents approaches that make certain assumptions about the relationship among the keys on a d-ary tree and significantly reduce the group controller key storage requirements [6]–[8] while maintaining the user key storage and the update communication as $\mathcal{O}(\log_d N)$.

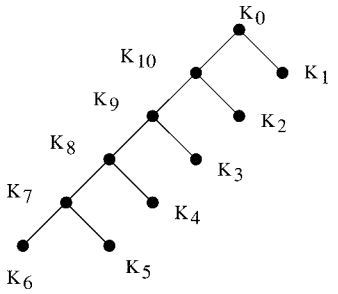

Fig. 2. A generic key distribution tree.

In [7], [8], binary rooted-tree-based key distributions which require[2] the GC to store a total of $2\log_2 N$ distinct keys were proposed. For a d-ary tree, the approach in [7], [8] will require $d\log_d N$ keys to be stored at the GC. However, the number of keys that need to be updated under a single member deletion remains at $\log_d N$ as in [28], [29]. Hence, the results in [8] reduce the storage requirements of GC by

$$\frac{d(N+1)-2}{d-1} - d\log_d N = \frac{d(N+1-(d-1)\log_d N)-2}{(d-1)} \tag{1}$$

number of keys without increasing the key storage requirements at the end user node. When the tree is binary, the reduction in storage due to approach in [7], [8] is $2(N-\log_2 N)$. For large N, the reduction in storage is $\mathcal{O}(N)$, but it comes at the cost of additional security problems under multiple member deletion, discussed later in Section VI.

IV. Preliminary Observations

We first show the need to optimize the rooted tree using a worst case example. Consider the binary rooted tree shown in Fig. 2. Every member is assigned to a unique leaf node. We assume that the group size is N. In this tree, the average number of keys to be updated after a member deletion is computed as

$$\frac{\sum_{i=1}^{N-1}(i+1)+N}{N} = \frac{N}{2}+1.5-\frac{1}{N}. \tag{2}$$

Hence, the average number of keys to be invalidated grows as $\mathcal{O}(N)$ for this model. However, in [6]–[8], [28], and [29] a virtual tree was built based on the group size N. Every member was assigned keys based on the observation that for N members, $\log_d N$ keys are sufficient for a d-ary rooted tree. For this model, the average number of keys to be updated grows as $\log_d N$.

From the results in [6]–[8], [28], and [29], we note that the average number of keys to be updated is almost equal to the average number of keys that are assigned to the deleted member (except for the leaf node key). Hence, the average number of keys to be updated (or stored by a user) can be considered as an efficiency parameter for these multicast key distribution models. The goal of this paper is to develop a systematic approach to compute the optimal value of the average number of keys to be updated by GC (or stored by a user). In order to compute the optimal value of the average number of keys to be generated by the GC, we study the member deletion process. We show that the ability of the GC to reach every member under member deletion in the rooted trees leads to a natural constraint that can be used to compute this optimal value. We also show that the optimization problem arising in the context of key assignment is abstractly equivalent to the optimal codeword-length selection problem.

We first define the necessary terminology and proceed to formulate the necessary optimization problem.

A. Cover-Free Key Distribution

In assigning keys to members, the GC needs to ensure that every valid member can be securely reached under member deletion. The GC also needs to make sure that illegal collaboration among two or more members does not enable them to *cover* all the keys assigned to a valid member. This cover-free property has been used in the context of broadcast encryption and traitor tracing in [11], [14]. In the context of tree-based key distribution, the cover-free property requires that regardless of how many members are being deleted (or how many collude) simultaneously, every valid member should be able to securely communicate with the GC. We formally define the cover-free property among sets below.

Definition: Given a collection of sets $\{S_1, \ldots, S_N\}$, a nonempty set S_i is said to be (k, α) cover-free if

$$\frac{\left|S_i \setminus \bigcup_{j=1; j\neq i}^{k} S_j\right|}{|S_i|} \geq \alpha_i \tag{3}$$

where, $0 \leq \alpha_i \leq 1$, $1 \leq k \leq N$. When $\alpha_i = 0$, there is no cover-free key distribution for key set S_i. For cover-free condition, $\alpha > 0$.

B. User Indexing and Key Indexing

Let $X_{n-1}X_{n-2}X_{n-3}\cdots X_0$ denote the binary user index (UID) string where X_i's take the value "0" or "1." In order to delete a member, the GC has to have a lookup table that contains a unique UID and the list of cryptographic keys assigned to the member to be deleted. Deleting a specific member involves deleting, and possibly updating, some of the keys assigned to that member. Since the GC should be able to securely communicate with every valid member, after deleting one or more members, every member should be assigned a unique set of keys (keys need to be in one-to-one correspondence with the UID of that member). Hence, if we concatenate the set of keys assigned to a member and form a key index (KID),[3] every member should have a unique KID.

Although the KID and the UID need to be in one-to-one correspondence, a KID needs to satisfy additional constraints that a UID does not need to satisfy. We first illustrate this by an example. Consider the alphabets $\{0, 1\}$ used for UID generation and the keys $\{K_1, K_2\}$ used for KID generation. The UIDs "01" and "10" can be generated and assigned uniquely to two different members. The KIDs K_1K_2 and K_2K_1, however, cannot be assigned to two different members. If we do assign them to two different members, the keys assigned to a member

[2]We will discuss the approach of [8] in Section VI.

[3]In forming the KID, we ignore the root key and the session key that are common to all the members. Unless explicitly mentioned,we also ignore these two keys in all the computations that follow.

can be completely covered by the keys assigned to the other member. *Although this is a special type of set covering resulting from the permutations of the keys, this is crucial in defining the KIDs.* From this example, we note the following property of the KIDs: Any permutation of the keys forming a KID will lead to a KID that is completely *covered* by the original KID. We use this property in formally defining the KID.

Definition: Key index (KID) of a member M_i is a string generated by the concatenation of the keys assigned to the member M_i, *taken in any order*. If the number of keys assigned to member M_i is denoted by l_i, then there are $l_i!$ possible different KID strings that can be generated using these l_i keys. Given a KID, all the KIDs that are generated by permuting and concatenating its keys are equivalent with respect to the cover-free property. If the set of keys generating KID_1 is denoted by S_1, and the set of keys generating KID_2 is denoted by S_2, we denote $\text{KID}_1 \equiv \text{KID}_2$ if $S_1 = S_2$.

From Fig. 1, the member M_1 is assigned four KEKs $\{K_O, K_{2.1}, K_{1.1}, K_{0.1}\}$. Since K_O is common to all the members, it can be ignored in defining the KIDs. Hence, the KID of M_1 is $K_{2.1}K_{1.1}K_{0.1}$. Since there are six different ways to concatenate these keys, there are five additional KIDs generated by permuting and concatenating the keys forming KIDs of M_1. This equivalence among the KIDs generated by permuting a set of keys is a feature that separates the conventional UIDs from KIDs.

C. Key Indexing and Kraft Inequality

We noted that in order for GC to securely communicate with every valid member under member deletion, every member should be assigned a unique UID as well as a KID. For a key assignment on the rooted trees, cover freeness is the requirement that the KID (UID) of a member should not be a *prefix* of the KID (UID) of any other member. On the rooted tree, this condition can be mathematically stated using the Kraft inequality.

Theorem 1 (Kraft Inequality for KIDs): For a d-ary rooted key tree with N members and KIDs satisfying the prefix condition, if we denote the number of keys forming the KID of member M_i by l_i, for a secure multicast group with N users, the set $\{l_1, l_2, \ldots, l_N\}$ satisfies the Kraft inequality given by

$$\sum_{i=1}^{N} d^{-l_i} \leq 1. \tag{4}$$

Conversely, given a set of numbers $\{l_1, l_2, \ldots, l_N\}$ satisfying this inequality, there is a rooted tree that can be constructed such that each member has a unique KID with no prefixing.

Proof: Well known, and available in [9] and [12]. □

While this prefix-free condition is necessary and sufficient for indexing a member using UID, this is only a necessary condition for the KID to be cover-free. We first illustrate this difference. Consider the set of keys $\{K_1, K_2, K_3, K_4\}$ used to form the KIDs $\{K_2K_3K_4\}$, $\{K_1K_3K_4\}$, $\{K_1K_2K_4\}$, and $\{K_1K_2K_3\}$ assigned to members M_1, M_2, M_3, and M_4, respectively. It can be verified that no KID is a prefix of another. Also, the KID lengths satisfy the Kraft inequality since $((2^{-3}.4) = 0.5 < 1)$. Since no KID is a prefix of another, if a key set of a single member is deleted, there is at least one key in each of the remaining key sets that is not contained by the key set of the deleted member. Hence, a single user deletion does not invalidate *all* keys of other members and the key assignment is prefix-free. However, it can be verified that in the above given example, the union of any two sets of KIDs will cover the keys forming all other KIDs. Hence, collaboration or deletion of two or more members will compromise the keys of all other members.

Hence, the prefix-free property and Kraft inequality are only necessary for being able to reach a valid member under member deletion.

We note that this result can be explained by the fact that the Kraft inequality is a property exhibited by the tree structure and is independent of the nature of the elements of KID. *Hence, the selection of KIDs satisfying the prefix condition is not sufficient to safeguard against failure of the key distribution scheme under member deletion or user collusion*. We present an example [8] later in Section VI.

D. Making KIDs Cover-Free

If a KID assignment is cover-free, it has to be prefix-free. We showed that prefix-free does not imply cover-free. We now pose a question, *how to construct a prefix-free KID assignment on the tree that is also cover-free?* In order to provide a condition that a KID is cover-free, we need to consider the definition of the cover-free condition given earlier. Since every valid member needs to be able to securely communicate with the GC under deletion of one or more members, even if all the $(N-1)$ members are deleted, the remaining single member should be securely reachable. In terms of the parameters (k, α) given earlier, setting $k = (N-1)$ leads to the cover-free condition for the valid member M_i with key set S_i as

$$\frac{\left| S_i \setminus \bigcup_{j=1; j\neq i}^{N-1} S_j \right|}{|S_i|} \geq \alpha_i \tag{5}$$

with $0 < \alpha_i \leq 1$. Since $\alpha_i > 0$, the lowest possible value of α_i is $\frac{1}{|S_i|}$, with the physical interpretation that the set S_i should have at least one key that is different from the union of all other key sets.

In order to construct the tree-based key assignment scheme that satisfies this condition, we consider the manner in which the members are assigned to a logical tree. In a logical tree of [28], [29] each member is assigned to a unique leaf node. There is a unique path from the leaf to the root of the tree. Every member shares all the keys *except* the leaf node key with at least one more member. Hence, choosing the leaf node keys to be distinct will make sure that the key set S_i of member M_i has at least one element that is not covered by the union of all other key sets.

Hence, if we *choose all the leaf node keys to be distinct*, rooted-tree-based prefix-free KID assignment will be *necessary and sufficient* to: a) prevent user collusion from completely disabling the secure communication, and also b) reach a valid member under deletion of arbitrary number of members. Since there are N leaf nodes, the number of keys to be stored by the

GC grows as $\mathcal{O}(N)$ when there is no additional relationship among keys.[4]

With these preliminary observations, we now show how to minimize the average number of keys to be regenerated by the GC under member deletion.

V. Probabilistic Modeling of Member Deletion

Using l_i to denote the length[5] of the KID of member M_i, we note that in the rooted-tree-based key distribution model [28], [29], member M_i shares l_i number of KEKs with two or more other members. This count includes the rooted KEK and excludes the leaf node key of member M_i. In the event the member M_i is deleted, the number of keys to be updated is also l_i. Hence, if we can minimize l_i, this will minimize the user key storage.

In the rooted-tree-based key distribution of [28], [29], each member shares $\log_d N$ keys (excluding the root key and the SEK) with two or more members. At the time a member is deleted, $\log_d N$ keys are to be updated and communicated to other members. The only key that needs not be updated under member deletion is the leaf node key of a deleted member. Hence, we note that the user key storage and the keys to be updated under member deletion grows as $\mathcal{O}(\log_d N)$ for the rooted-tree-based key-distribution schemes in [28], [29] that do not make use of the physical process of member deletion in assigning the keys. We now show that the use of the statistics of the member-deletion process will enable us to further reduce the user key storage and hence the key update requirements of the rooted-tree-based models [28], [29].

A. Relating Statistics of Member-Deletion Process to the Key Distribution on the Rooted Tree

Let p_i denote the probability of deletion of member M_i.[6] We assume that this probability is computable either empirically or is known *a priori*. This paper does not make an attempt to develop methods for computing p_i. Noting that every time a member is deleted, all the keys assigned to that member are deleted, we make the following observations.

- Since every member is assigned to a unique leaf node, and every leaf node is also assigned a unique key, probability p_i of deletion of a member M_i is *identical* to the probability of deletion of the leaf node key assigned to M_i.
- Since every member has a unique KID and the KIDs are formed by concatenating the keys assigned to a member, the probability of deletion of member M_i is *identical* to the probability of deletion of the KID of member M_i.

Hence, we note that the probability of deletion of a member is identical to the probability of deletion of its node key as well as its KID. Given the knowledge about the probability of member deletion, we define the *entropy of member deletion* by the following formula.

Definition: The d-ary entropy H_d of the member deletion is

$$H_d = -\sum_{i=1}^{N} p_i \log_d p_i \tag{6}$$

where p_i is the probability of deletion of member M_i. A word of caution is in place since this formula of entropy is often used to describe the rates in the source coding literature. We use it in the context of its physical interpretation which is the amount of uncertainty about the member deletion statistics.

Since the member deletion event and the leaf node key deletion event have identical probabilities, the d-ary entropy of the member deletion event is the same as the entropy of the leaf key deletion event. Similarly, the entropy of KID deletion is identical to the entropy of member deletion. We will use the term entropy of member deletion event instead of entropy of leaf key deletion or entropy of KID deletion since they are equivalent.

In summary, a main outcome of these observations is that the *entropy of the KID deletion* is identical to the *entropy of member deletion* which is a physically observable process, and can be completely characterized once the entropy of member deletion is known.

B. Assigning Optimal Number of Keys per Member

When a member M_i is deleted with a probability p_i, the group controller has to generate and update l_i keys that were shared with other members. Hence, on average, the GC has to generate and update

$$\sum_{i=1}^{N} p_i l_i \tag{7}$$

number of keys. This is also the average number of keys that a member needs to be assigned. As noted earlier, we have chosen not to count the session key and the root key that are updated for every member deletion. The GC has to find an optimal key assignment that will minimize the average number of keys to be updated over the duration of the session. However, as noted earlier, any key assignment needs to satisfy the Kraft inequality. Hence, the optimization problem arising from the multicast key distribution on the rooted tree models of [28], [29] is

$$\min_{l_i} \sum_{i=1}^{N} p_i l_i \tag{8}$$

subject to the constraint

$$\sum_{i=1}^{N} d^{-l_i} \leq 1. \tag{9}$$

This problem can be written as a Lagrangian optimization problem as

$$\min_{l_i} \left\{ \sum_{i=1}^{N} p_i l_i + \lambda \left(\sum_{i=1}^{N} d^{-l_i} - 1 \right) \right\} \tag{10}$$

where λ is a Lagrangian multiplier. This optimization problem is identical to the well-known optimal codeword-length selection problem [9] for prefix coding in the context of information theory. This problem is well-studied and the optimal strategy yields the Shannon entropy of the random variable being coded as the optimal codeword length [9]. Since the abstract mathematical formulations are identical, we can use identical argu-

[4]Reduction of the key storage requirements of the group controller as a sublinear function of group size was presented in [5] using pseudorandom functions. In our study, we assume that the keys are distinct and have no relationship among them.

[5]Recall that the number of KEKs except the root KEK is noted as the length.

[6]We use the term deletion in general to denote deletion, voluntary leaving, and revocation under compromise.

ments to derive the optimal number of keys to be assigned to a member on the rooted tree. The derivation is standard [9], and leads to the following conclusion.

In the context of key distribution on the rooted trees, the optimal number of keys to be updated is the entropy of the member deletion process. We summarize this result as Theorem 2 without repeating the proofs [9].

Theorem 2: Let p_i denote the probability of deleting member M_i. Let the group size be N. Let the degree of the rooted tree of key distribution [28], [29] be d. Then, for a rooted-tree-based key distribution satisfying at least the Kraft inequality, the optimal average number of KEKs,[7] denoted by $\sum_{i=1}^{N} p_i \log_d p_i$, to be assigned to a member, is given by the d-ary entropy

$$H_d = -\sum_{i=1}^{N} p_i \log_d p_i \tag{11}$$

of the *member deletion* event. Including the root key and SEK, the optimal average number of keys per member is given by $H_d + 2$. For a member M_i with probability of deletion p_i, the optimal number of keys to be assigned (excluding the root key and the session key) is computed from (10) as

$$l_i^* = -\log_d p_i. \tag{12}$$

The number of keys assigned to member M_i with deletion probability p_i, including the SEK and the root key, is given by

$$l_i^* + 2 = -\log_d p_i + 2 = \log_d \frac{d^2}{p_i}. \tag{13}$$

Although the UID and the KID assignments need to satisfy a set of different requirements, the following features of key assignment are a direct consequence of the optimization results and hold for optimal KIDs as well as UIDs.

Lemma 2:

1) A member with higher probability of deletion should be given fewer keys compared to a member with lower probability of being deleted. If $p_i > p_j$, then
$$l_i(= -\log_d p_i) < l_j(= -\log_d p_j).$$
2) There must be at least two members with the longest KID strings.
3) Since the number of KEKs to be regenerated by the GC needs to be an integer and the value of H_d may not be an integer, the *true* average number of keys per member differs from H_d by a finite value. In fact, it differs from the true value by at most one digit . If the member deletion probabilities are d-adic then the optimal value of the keys to be regenerated is exactly H_d.

Sketch of the Proofs:

1) The logarithm being a monotone function, if $p_i > p_j$, then $\log_d p_i > \log_d p_j$. Hence $-\log_d p_i < -\log_d p_j$, leading to $l_i(= -\log_d p_i) < l_j(= -\log_d p_j)$.[8]
2) In order to prove this, we note that the KIDs need to be unique with a minimum possible number of keys. Hence, if there is only one member with the largest number of keys on the rooted trees, then we can reduce the largest number of keys held by at least one and still ensure that all members have a unique set of keys assigned. However, this reduction will contradict the optimality of the individual KID lengths. Hence, at least two members should be assigned the largest number of keys.
3) The proof follows by bounding the average number of keys to be regenerated by the GC $(1 + \sum_{i=1}^{N} p_i l_i^*)$. It is given below.

$$\begin{aligned} l_i^* &= \lceil -\log_d p_i \rceil \\ -\log_d p_i &\le l_i^* < -\log_d p_i + 1 \\ -p_i \log_d p_i &\le p_i l_i^* < -p_i \log_d p_i + p_i \\ -\sum_{i=1}^{N} p_i \log_d p_i &\le \sum_{i=1}^{N} p_i l_i^* < -\sum_{i=1}^{N} p_i \log_d p_i + \sum_{i=1}^{N} p_i \\ H_d &\le \sum_{i=1}^{N} p_i l_i^* < H_d + 1. \end{aligned} \tag{14}$$

Hence, the KID length will be at most one unit more than the entropy of member deletion. Since the SEK and the root KEK are common to all the members, we note that the average number of keys to be updated is at most three more than the entropy of member deletion event.

C. Maximum Entropy and the Rooted-Tree-Based Key Assignment

We now interpret the rooted-tree-based key distribution results reported in [7], [8], [28], and [29] with respect to the results derived in the previous subsection.

We showed that the average number of keys to be updated by the GC is

$$H_d = -\sum_{i=1}^{N} p_i \log_d p_i.$$

We now try to find the maximum value of the average number of keys to be generated. This problem is posed as

$$\max_{p_i} H_d = -\sum_{i=1}^{N} p_i \log_d p_i \tag{15}$$

subject to the condition

$$\sum_{i=1}^{N} p_i = 1. \tag{16}$$

This problem can be posed as the Lagrangian optimization problem with respect to the variable p_i

$$\max_{p_i} \left\{ -\sum_{i=1}^{N} p_i \log_d p_i + \lambda_1 \left(\sum_{i=1}^{N} p_i - 1 \right) \right\} \tag{17}$$

where λ_1 is a Lagrangian multiplier. Some algebra yields the solution to this problem as $p_i = \frac{1}{N}$. The members of the group have equal probability of being deleted. For this value of the

[7]Excluding the root key and the session key.

[8]Since the members with higher probability of being deleted are assigned fewer keys in this strategy, the GC can adaptively react to any possible coordinated attack effort by members to increase the frequency of rekeying by simply forcing deletion by leaving and joining at very high rates. The GC will be able to assign a lower number of keys to members with higher probabilities of being deleted. In the traditional models [28], [29], there is no explicit mechanism to include this knowledge into the key distribution.

probability of member deletion, the maximum entropy value is given by

$$H_d(\max) = -\sum_{i=1}^{N} N^{-1}\log_d N^{-1} = \log_d N.$$

When the member deletion probability of M_i is $p_i = N^{-1}$, the optimal number of keys to be assigned to M_i is $\log_d N$.

However, the schemes in [7], [8], [28], [29] assign $\log_d N$ keys per member on the tree. Since we showed that the entropy is the average number of keys to be assigned to a member, and the entropy is maximized when all the members have the same probability of being deleted, the key assignments in [7], [8], [28], [29] correspond to the strategy of assigning maximal average number of keys to every member.

In terms of the design of the rooted-tree-based schemes, what we have shown is that when there is no prior knowledge about the probabilities of member deletion, the assignment of $\log_d N$ number of keys per member corresponds to the optimal strategy that assumes the worst case in terms of the average number of key assignments. This can be written as the following $\max$–$\min$ problem:

$$\max_{p_i}\left\{\min_{l_i}\left\{\sum_{i=1}^{N} p_i l_i + \lambda\left(\sum_{i=1}^{N} d^{-l_i} - 1\right)\right\} + \lambda_1\left(\sum_{i=1}^{N} p_i - 1\right)\right\}. \quad (18)$$

The following theorem summarizes the problem and the solution.

Theorem 3: For a d-ary rooted-tree-based multicast key distribution scheme with N members and a member M_i with probability of deletion p_i, the average number of keys to be regenerated by the GC is upper-bounded by $\log_d N$ up to addition of a constant (the session key). This upper bound is reached when the entire group has identical probability of being deleted.

D. Impact of Using Incorrect Probability on Key Length

In Fig. 2, we presented the effect of an unbalanced rooted tree on the number of keys to be assigned and to be invalidated. We note that this quantity can be completely characterized using basic results from information theory as well. Let us assume that the true deletion probability of member M_i is p_i and the used probability of deletion for member M_i is q_i. Hence, the optimal number of keys to be assigned to M_i, denoted by $\hat{l}_i^*$, is given by

$$\hat{l}_i^* = \lceil -\log_d q_i \rceil, \qquad -\log_d q_i \le \hat{l}_i^* < -\log_d q_i + 1. \quad (19)$$

On average, the number of keys that are assigned due to this incorrect computation, denoted by C, is bounded above and below by

$$C = \sum_{i=1}^{N} p_i \lceil -\log_d q_i \rceil$$

$$\sum_{i=1}^{N} p_i\left(\log_d \frac{p_i}{q_i p_i}\right) \le C < \sum_{i=1}^{N} p_i\left(\log_d \frac{p_i}{q_i} - \log_d p_i + 1\right)$$

$$H_d + D(p\|q) \le C < H_d + D(p\|q) + 1 \quad (20)$$

where

$$D(p\|q) = \sum_{i=1}^{N} p_i(\log_d \tfrac{p_i}{q_i})$$

is the information divergence [9], which is a measure of how far apart are two distributions. Hence, on average, the redundant number of keys assigned due to incorrect probabilities is $D(p\|q)$. This is stated as a theorem below.

Theorem 4: Let p_i and q_i denote the true and assigned deletion probabilities of member M_i, respectively. The average number of redundant keys to be assigned on the rooted-tree-based key distribution is $D(p\|q)$, where

$$D(p\|q) = \sum_{i=1}^{N} p_i \log_d \tfrac{p_i}{q_i}.$$

E. Bounds on Average Key Length

We now relate the hardware bit generation rate to the entropy of member deletion. We noted that the average number of KEKs to be regenerated by the GC is given by the entropy of member-deletion process. Since the total number of keys to be generated by the GC also includes the session key SEK, on average, the total number of key digits generated by the GC is $(1+H_d)$. If we assume all the keys have identical length L, and GC generates B digits per unit of time, then the number of digits to be generated by the GC and the key length are related by the inequality

$$B \ge L(1 + H_d). \quad (21)$$

Hence, the average key length is bounded by

$$L \le \frac{B}{1 + H_d}. \quad (22)$$

When a member M_i with deletion probability p_i is deleted, from (14), the GC needs to update $(1 - \log_d p_i)$ keys. Since $-\log_d p_i$ will attain its maximum value when p_i attains its smallest value $p_i = p_{\min}$, the maximum number of keys need to be updated when a member M_i with probability of deletion $p_i = p_{\min}$ is deleted. Hence, the bound on key length is

$$L \le \frac{B}{1 - \log_d p_{\min}}. \quad (23)$$

Combining (22) and (23), we have the following.

Theorem 5: For a rooted-tree-based multicast key distribution scheme in [28], [29], with a group size of N and member deletion probabilities $\{p_i\}_{i=1}^{N}$, if the bit generation rate of GC is B, then the sustainable key length that can be updated in unit time is

$$L \le \min\left\{\frac{B}{1 + H_d}, \frac{B}{1 - \log_d p_{\min}}\right\}. \quad (24)$$

F. Relationship to the One-Way-Function-Based Key Selection Schemes on the Tree

In [28], [29], there is no specification about the manner in which the KEKs are generated. New results on using pseudo-

random one-way functions for KEK constructions on the rooted tree were reported in [2], [6].

In [5], a hybrid scheme that combines the scheme that clusters a multicast group into clusters of size M and building a rooted tree with a unique cluster at each leaf node was proposed. Each cluster member is assigned a unique key that is generated using a pseudorandom function [17], [21] with a common seed [5], thus, reducing the storage for M members of the cluster to a single seed. Since each cluster is assumed to be of uniform size M, for a group of size N, there are $\lceil \frac{N}{M} \rceil$ clusters. Since each cluster is assigned to a unique leaf node of the rooted tree of degree d, the depth or the height of the tree is

$$h = \log_d \left\lceil \frac{N}{M} \right\rceil. \tag{25}$$

For the cluster of size $M = \log_d N$, the number of keys to be stored by the GC is

$$1 + d + d^2 + \cdots + d^h = \frac{d^{h+1} - 1}{d - 1} = \frac{dN - M}{(d-1)M}. \tag{26}$$

Since there cannot be any more entropy than that provided by the *member revocation event*, our formulation based on entropy will yield the lowest average cost of key generation when there are no additional relationships among keys. The only way to further reduce the average communication or storage overhead in key generation is to introduce relationships (for example, using pseudorandom functions) among the keys generated as in [5], [6].

We do, however, note that in our formulation, since we minimize the average number of keys assigned based on the member-deletion probabilities, the optimal KID length $\log_d \frac{1}{p_i}$ of member M_i will be larger than $\log_d N$ when the deletion probability p_i is small. Hence, the number of keys to be updated upon deletion of member M_i will be quite large.

Hence, the use of additional relationships among the KEKs, such as the manner in which they are generated, can help in further reducing the amount of keys to be stored by the GC. Our approach, however, does not attempt to minimize the key storage of GC, which was one of the main results in [5].

VI. A Rooted-Key Distribution With User Collusion

We now describe a rooted-tree-based key distribution scheme [8] that satisfies the maximum entropy bound $(\log_d N)$ for the number of keys assigned to a member while attempting to minimize the storage of the GC.[9] We use this scheme [8] to illustrate that while a key distribution scheme may attain optimality in user storage, it may not be collusion-free.

Let $X_{n-1}X_{n-2}\cdots X_0$ denote a binary UID of the members. Each of the bit X_i is either a zero or a one. There are 2^n possible different UIDs for this sequence. In [7], [8], the following direct mapping between the KIDs and the UIDs was proposed. In [7], [8], when $N = 8$, $\log_2 8 = 3$ bits are needed to uniquely index *all* eight members. The authors then proceeded to note that since each bit X_i of the UID takes two values, these two values can be mapped to a pair of distinct keys. For example, when X_i is "0," it is represented by key K_{i0}, and when it is "1," it is represented by K_{i1}. Table I reproduces the mapping between the index (ID) bit number and the key mapping for the case in [7] for $N = 8$ where, the key pair (K_{i0}, K_{i1}) represents the two possible values of the bit X_i of the member index.

[9]Minimizing the GC storage with additional constraints has been studied in [5], where additional assumptions have been made about the key generation and there is an increase in the amount of keys also stored by every member.

TABLE I
A Key Distribution With Collusion in [7], [8]

ID Bit X_0	K_{00}	K_{01}
ID Bit X_1	K_{10}	K_{11}
ID Bit X_2	K_{20}	K_{21}

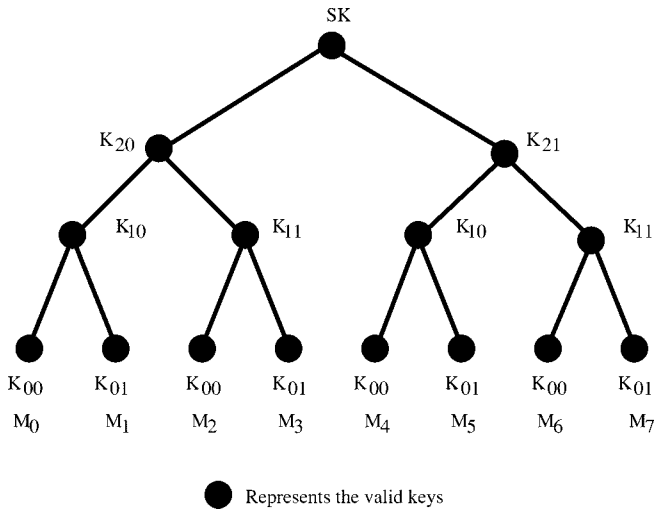

Fig. 3. Key distribution in [7] and [8].

Fig. 3 presents the corresponding binary tree for the key assignment. This rooted tree has the special structure that at any given depth from the root, two new keys are used. At depth h from the root, the two new keys $K_{(\log_d N - h)0}$ and $K_{(\log_d N - h)1}$ are duplicated h times. For example, at depth two from the root, KEKs K_{10} and K_{11} are duplicated across the tree twice. The total number of keys to be stored by GC in this scheme is $2\log_2 N$. For a d-ary rooted tree, the total number of keys to be stored by GC in this scheme is $d\log_d N$. Every member has to store only $\log_d N$ keys (excluding the root key and the session encryption key) and the GC needs to regenerate $\log_d N$ keys under member deletion. Hence, this scheme is indeed an optimal solution with respect to a single member deletion and also has significantly less storage for the GC than that of seminal schemes.

Although the total number of keys to be stored by the GC is $d\log_d N$, deletion of more than one member may bring this key distribution scheme to halt. In the case of Fig. 3, this happens if the members M_0 and M_7 (or M_1 and M_6) need to be deleted. The KID of member M_0 is $K_{20}K_{10}K_{00}$ and the KID of member M_7 is $K_{21}K_{11}K_{01}$. The union of the keys forming these two KIDs includes all the keys used for key distribution on the tree. The corresponding keys to be deleted are shown in Fig. 4. Hence, if these two members need to be simultaneously deleted, the GC is left with *no key* to securely communicate with the rest of the valid members. The compromise recovery under simultaneous deletion of M_0 and M_7 requires that the *entire group rekey itself*.

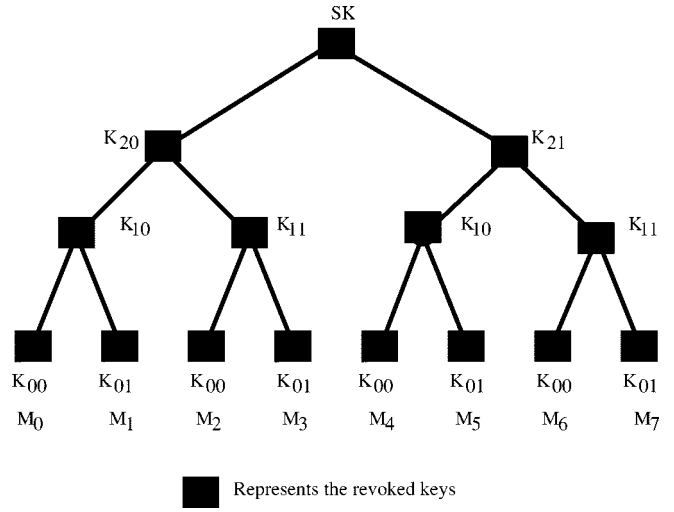

Fig. 4. Deletion of members M_0, M_7 in [7], [8].

Apart from member deletion, the key assignments in [7], [8] and their variations also allow the members to collaborate and compromise the system. We now interpret the user collusion on the rooted tree in [7], [8].

A. Interpretation Based on Minimal Number of Key Requirements

In Section IV-D, we noted that the sufficient condition for cover freeness requires that all the N leaf nodes are assigned distinct keys. Since the total number of keys to be stored in [8] is $d\log_d N$ for a group of size N, this model can be made cover-free if

$$d\log_d N \geq N \tag{27}$$

$$N \geq d^{\frac{N}{d}}. \tag{28}$$

At equality, the group size N should be a power of d. Setting $N = d^k$, where k is an unknown to be determined, leads to the equation

$$kd = d^k \tag{29}$$

$$k = d^{k-1}. \tag{30}$$

We note that $k = 1$ satisfies this equation with $N = d^k = d$ for $d \geq 2$. Table II summarizes the values of k that satisfy $k = d^{k-1}$ for different values of d.

The proof consists of two steps. The first step is to show that if at $k = k_0$, $k_0 < d^{k_0-1}$, then for all $k > k_0$, $k < d^{k-1}$, and hence $N < d^{\frac{N}{d}}$. The second part of the proof is to show that for $d = 2$, $k = 1, 2$ are feasible solutions and for $d > 2$, $k = 1$ is the only solution.

We first prove that if $k = k_0$, $k_0 > 1$, then

$$1 < k_0 \tag{31}$$

$$k_0 + 1 < k_0 + k_0 = 2k_0 < 2 * 2^{k_0-1} = 2^{k_0}$$

$$k_0 + 2 < 2(k_0 + 1) < 2^2 k_0 < 2^{k_0+1} \tag{32}$$

$$k_0 + 3 < 2(k_0 + 2) < 2^2(k_0 + 1) < 2^3 k_0 < 2^{k_0+2} \tag{33}$$

$$k_0 + i < 2(k_0 + i) < \cdots < 2^i k_0 < 2^{k_0+i-1}. \tag{34}$$

TABLE II
VALUES OF k THAT SATISFY $k = d^{k-1}$

$d = 2$	$k = 1$ or $k = 2$	$N = 2$ or $N = 4$
$d > 2$	$k = 1$	$N = d$

The first three steps are self-explanatory and the last step follows from inductive expansion of inequalities on k_0. Hence, we have shown that if at $k = k_0$, $k_0 < d^{k_0-1}$, then $\forall\, k \geq k_0$, $k < d^{k-1}$. This in turn implies that there is no integer $k > k_0$ for which $N = d^k \geq d^{\frac{N}{d}}$. Since the integers satisfy partial order, and $d \geq 2$ is an integer, if for $d = d_0$ and $k = k_0$, $k_0 < d_0^{k_0-1}$, then

$$\forall\, d > d_0, \qquad k_0 < d_0^{k_0-1} < d^{k_0-1}. \tag{35}$$

We note that for $d = 2$, $k = 1, 2$, satisfy $k = d^{k-1}$. When $d = 2$, if $k = 3$, $k < d^{k-1}$. Hence, the group size N does not satisfy inequality (24). From inequality (28), we can conclude that for a binary tree, if the group size is more than four, there cannot be a collusion free key distribution using the method in [8].

If we set $d = 3$ and $k = 2$, we have

$$k = 2 < 3^{2-1} = d^{k-1} = 3.$$

Hence, for $k > 1$, $d = 3$, there is no integer $N > 3$ which satisfies the inequality (24). Making use of the fact that if $k < 3^{k-1}$ then $k < 3^{k-1} < 4^{k-1} < 5^{k-1} \cdots$, we conclude that for $d > 2$, the only value of N for which $N \geq d^{\frac{N}{d}}$ holds is $N = d$.

Hence, we have shown that the key distribution in [8] will be collusion-free if the group size is identical to the degree of the rooted tree.

B. Another Interpretation of the Collusion Problem

The second interpretation of the collusion problem in [8] is based on the notion of sets, and is also discussed under the category of complementary variables in [10], [28]. In complementary variable approach, every member of the group is identified by a unique key. This unique key is distributed to everyone in the group excluding the member identified by that key. When a member is deleted, the index of the member is broadcast. For the next session, all the valid members set the key corresponding to the deleted member as the new session key. Under this model, for a set of N members, all the members will have $(N-1)$ keys that correspond to other members and no member will have the key corresponding to itself. If we consider any two members, the union of the keys stored by them will cover the keys stored by the entire group. Hence, this key assignment does not scale beyond deletion of one member. The scheme in [8] can be interpreted as a complementary variable approach as detailed in what follows.

If we use the notation $(k_j, \hat{k}_j)$ to denote the unique key pair representing the two possible binary values taken by the UID bit X_j, we note that the collusion or compromise of two members holding keys k_j and $\hat{k}_j$, respectively, will compromise the integrity of the key pair $(k_j, \hat{k}_j)$. In a d-ary rooted-tree-based key distribution in [8], each digit can take possible values between $(1, d)$ and the sum of these values is given by $\frac{d(d-1)}{2}$. Let the value of the bth-bit location of a member M_i be denoted as b_i.

Then, a set of k members can collude and compromise all the keys corresponding to the bit location b if

$$b_1 + b_2 + \cdots + b_k \equiv 0 \bmod \frac{d(d-1)}{2}. \tag{36}$$

VII. Conclusion and Discussion

This paper showed that the recently proposed [28], [29] rooted-tree-based secure multicast key distribution schemes can be systematically studied using basic information-theoretic concepts. By using the member deletion event as the basis of our formulation, we showed that the optimal number of keys assigned to a member is related to the *entropy* of the member deletion statistics. We derived the necessary and sufficient condition for the key assignment to be collusion-free and in general "cover-free." In particular, we showed that the cover-free condition on the rooted trees requires that the leaf node keys be all distinct when there is no additional relationship among keys. Under this condition, the storage requirements of the group controller is linear in group size N. We then proved that the currently available known rooted-tree-based strategies [28], [29] and their variations [7], [8] correspond to the maximum-entropy-based key assignment among all the rooted-tree-based strategies. Hence, the key distribution schemes in [7], [8], [28], and [29] correspond to a $\max$–$\min$ key assignment strategy. We also derived a relationship between the average key length, probability of member deletion event, and the hardware digit generation rate.

Acknowledgment

The authors wish to thank the anonymous reviewers for useful suggestions that significantly improved the presentation on the cover-free conditions. They also wish to thank the reviewers for suggestions to highlight the difference between our approach and the ones using pseudorandom function trees, [5]. R. Poovendran would like to acknowledge the many useful discussions with Dr. E. Harder that helped during the course of this work and revision. R. Poovendran would also like to thank Prof. A. Ephremides and Prof, J. Massey for providing early references on the rooted-tree-based analysis of algorithms.

References

[1] A. Ballardie, "Scalable multicast key distribution," report, RFC 1949, May 1996.

[2] D. Balenson, D. A. McGrew, and A. Sherman, "Key establishment in large dynamic groups: One-way function trees and amortized initialization," IETF Draft, draft-balenson-groupkeymgmt-oft-00.txt, Feb. 1999.

[3] M. Brumester and Y. Desmedt, "A secure and efficient conference key distribution system," in *Advances in Cryptology—Eurocrypt'94 (Lecture Notes in Computer Science)*. Berlin, Germany: Springer-Verlag, 1994, vol. 950, pp. 275–286.

[4] R. Canetti and B. Pinkas, "A taxonomy of multicast security issues," Internet draft, Apr. 1999.

[5] R. Canetti, T. Malkin, and K. Nissim, "Efficient communication-storage tradeoffs for multicast encryption," in *Proc. Eurocrypt 99*, pp. 456–470.

[6] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas, "Multicast security: A taxonomy and efficient reconstructions," in *Proc. IEEE INFOCOM'99*, pp. 708–716.

[7] G. Caronni, M. Waldvogel, D. Sun, and B. Plattner, "Efficient security for large and dynamic groups," in *Proc. 7th Workshop Enabling Technologies*. Cupertrino, CA: IEEE Comp. Soc. Press, 1998.

[8] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, and D. Saha, "Key management for secure internet multicast using Boolean function minimization techniques," in *Proc. IEEE INFOCOM'99*, pp. 689–698.

[9] T. Cover and J. Thomas, *Elements of Information Theory*. New York: Wiley, 1991.

[10] A. Fiat and M. Naor, "Broadcast encryption," in *Advances in Cryptology—CRYPTO'92 (Lecture Notes in Computer Science)*. Berlin, Germany: Springer-Verlag, 1993, vol. 773, pp. 481–491.

[11] E. Gafni, J. Staddon, and Y. L. Yin, "Efficient methods for integrating traceability and broadcast encryption," in *Advances in Cryptology—CRYPTO'99 (Lecture Notes in Computer Science)*. Berlin, Germany: Springer-Verlag, 1999, vol. 1666, pp. 372–387.

[12] R. Gallager, *Information Theory and Reliable Communication*. New York: Wiley, 1968.

[13] H. Harney and C. Muckenhirn, "GKMP architecture," *Request for Comments (RFC)*, vol. 2093, July 1997.

[14] R. Kumar, S. Rajagopalan, and A. Sahai, "Coding constructions for blacklisting problems without computational assumptions," in *Advances in Cryptology—CRYPTO'99 (Lecture Notes in Computer Science)*. Berlin, Germany: Springer-Verlag, 1999, vol. 1666, pp. 609–623.

[15] J. L. Massey, "An information-theoretic approach to algorithms," in *Impact of Processing Techniques in Communications*, ser. NATO Advanced Study Institutes Ser. E91, 1985, pp. 3–20.

[16] H. N. Lendal, Y. J. B. Khun, and J. L. Massey, "An information-theoretic approach to homomorphic substitution," in *Advances in Cryptology—Eurocrypt'89 (Lecture Notes in Computer Sceince)*. Berlin, Germany: Springer-Verlag, 1990, vol. 434, pp. 382–394.

[17] M. Luby, *Pseudo-Random Functions and Applications*. Princeton, NJ: Princeton Univ. Press, 1996.

[18] A. Menezes, P. van Oorschot, and A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL: CRC Press, 1997.

[19] U. M. Maurer, "Secret key agreement by public discussion from common information," *Trans. Inform. Theory*, vol. 39, pp. 733–742, May 1993.

[20] S. Mittra, "Iolus: A framework for scalable secure multicasting," in *Proc. ACM SIGGCOM'97*, Sept. 1997, pp. 277–288.

[21] M. Naor and O. Reingold, "From unpredictability to indistinguishability: A simple construction of pseudo-random functions from MACs," in *Advances in Cryptology—Crypto'98 (Lecture Notes in Computer Science)*. Berlin, Germany: Springer-Verlag, 1998, vol. 1462, pp. 267–282.

[22] R. Poovendran and J. S. Baras, "An information theoretic approach for design and analysis of rooted-tree based multicast key management schemes," in *Advances in Cryptology—CRYPTO'99 (Lecture Notes in Computer Science)*. Berlin, Germany: Springer-Verlag, 1999, vol. 1666, pp. 624–638.

[23] ——, "An information theoretic approach to multicast key management," in *Proc. IEEE Information Theory and Networking Workshop*, Metsovo, Greece, June 1999.

[24] B. Quinn, "IP multicast applications: Challenges and solutions," Internet draft, Nov. 1998.

[25] M. Steiner, G. Tsudik, and M. Waidner, "Diffie–Hellman key distribution extended to group communication," in *3rd ACM Conf. Computer and Communications Security*, 1996.

[26] D. R. Stinson and T. V. Trung, "Some new results on key distribution patterns and broadcast encryption," *Des., Codes Cryptogr.*, 1999.

[27] D. R. Stinson, *CRYPTOGRAPHY: Theory and Practice*. Boca Raton, FL: CRC, 1995.

[28] D. M. Wallner, E. J. Harder, and R. C. Agee, "Key management for multicast: Issues and architectures," Internet draft, Sept. 1998.

[29] C. K. Wong, M. Gouda, and S. S. Lam, "Secure group communications using key graphs," *IEEE/ACM Trans. Networking*, vol. 8, pp. 16–31, Feb. 2000. Also in *Proc. ACKM SIGCOMM'98*, Vancouver, BC, Canada, Sept. 2–4, 1998.