Distributing Key Updates in Secure Dynamic Groups[⋆]

Sandeep S. Kulkarni and Bezawada Bruhadeshwar

Department of Computer Science and Engineering, Michigan State University, East Lansing MI 48824 USA Tel: +1-517-355-2387, Fax: 1-517-432-1061
`sandeep@cse.msu.edu, bezawada@cse.msu.edu`

Abstract. We focus on the problem of distributing key updates in secure dynamic groups. Due to changes in group membership, the group controller needs to change and distribute the keys used for ensuring encryption. However, in the current key management algorithms the group controller broadcasts these key updates even if only a subset of users need them. In this paper, we describe a key distribution algorithm for distributing keys to only those users who need them. Towards this end, we propose a descendent tracking scheme. Using our scheme, a node forwards an encrypted key update only if it believes that there are descendents who know the encrypting key. We also describe an identifier assignment algorithm which assigns closer logical identifiers to users who are physically close in the multicast tree. We show that our identifier assignment algorithm further improves the performance of our key distribution algorithm as well as that of a previous solution. Our simulation results show that a bandwidth reduction of upto 55% is achieved by our algorithms.

Keywords : Secure Multicast, Key Distribution, Descendent Tracking, Identifier Assignment

1 Introduction

In group oriented applications, such as conferencing, networked gaming and news dissemination, it is necessary to secure the data from intruders as the data is confidential or it has monetary value. In the algorithms for secure group communication (e.g., [1–6]), a group controller distributes a cryptographic key, called the group key, to all the group users. The group membership is dynamic. To protect privacy of the current users, the group controller changes and securely distributes the group key at each membership change. This rekeying is especially needed when a user leaves the group and should no longer understand the group communication.

[⋆] This work is partially sponsored by NSF CAREER 0092724, ONR grant N00014-01-1-0744, DARPA contract F33615-01-C-1901, and a grant from Michigan State University.

In the algorithms in [1–4], the group controller distributes additional keys which are shared by different subsets of users. These shared keys reduce the number of group key update messages the group controller needs to transmit. The group controller encrypts the new group key with a minimum subset of the shared keys and transmits it to the current users. To reflect current group membership, the group controller changes and securely transmits the shared keys known to the leaving user. Although each of the new shared keys needs to be transmitted to only a subset of the users, the current key management algorithms assume a broadcast primitive. Hence, *all* the current users receive *all* the key update messages. Of course, users cannot decrypt the key update messages that are not intended for them since they do not have the necessary keys.

From the above discussion, we observe that the current key management algorithms only focus on *what* key update messages are sent but do not emphasize on *how* they are distributed. This results in wastage of bandwidth as users receive key update messages for keys that they do not need. This wastage increases further if retransmission is required for any lost messages and such retransmission is done using a broadcast. Although solutions for reducing the number of retransmissions in secure group communication have been proposed in [7,8], the group controller still needs to broadcast them. Thus, efficient distribution of key updates is an important problem in secure dynamic groups.

In this paper, we propose an algorithm for the distribution of key update messages in secure dynamic groups. In our key distribution algorithm, we integrate the key management algorithms in [1, 2] with appropriate forwarding functionality. We assume that the users are arranged in a multicast tree which can be built using any of the IP [9–12] or overlay [13–15] multicast protocols. Depending on the multicast protocol used, an intermediate node in the multicast tree can be a router (IP multicast) or an overlay node. Hence, we only focus on the actions of an intermediate node as the implementation details are beyond the scope of this paper. The contributions of our paper are as follows:

– We describe a compact descendant tracking scheme to track the descendants of the intermediate nodes. The memory required at the intermediate nodes for our scheme is small and scales logarithmically in the group size. The advantage of our descendant tracking scheme is that this information can either be updated periodically or after the group communication has resumed.
– We describe the forwarding mechanisms used by the intermediate nodes to forward the key update messages.
– We describe a user identifier assignment algorithm. Using our assignment algorithm, the group controller assigns closer logical identifiers to users who are located close to each other in the multicast tree. We show that our assignment algorithm improves the performance of our key distribution algorithm as well as that of a previous solution in [16].

Organization of the paper. The paper is organized as follows. In Section 2, we describe the notations used in our key distribution algorithm and describe a previous solution. In Section 3, we describe our key distribution algorithm and our user identifier assignment algorithm. In Section 4, we present the simulation results. Finally, in Section 5, we conclude and discuss some future work.

2 Notations

In this section, we describe the various components in our algorithms, i.e., the key management algorithms, the multicast tree and the problem of key distribution. We also briefly describe a previous solution to the problem of key distribution.

Key Management Algorithms. In the key management algorithms from [1, 2], the group controller arranges the users and the keys in a key tree. The leaf nodes in this key tree correspond to the users in the group (cf. Figure 1). Based on this arrangement of users and keys, we number them according to their location in the tree. For example, in a key tree of height h, the user $u_{i_1 i_2 .. i_h}$ denotes the user obtained by taking the i_1^{th} child at level 1, i_2^{th} child at level 2, and so on. The keys are also numbered likewise. Each node in the key tree is associated with a key from the logical key hierarchy [1] or a key from the complementary key hierarchy [2] or both. We use $k_{i_1 i_2 .. i_l}$ to denote the logical key at node $i_1 i_2 .. i_l$. And, we use $c_{i_1 i_2 .. i_l}$ to denote the complementary key at node $i_1 i_2 .. i_l$. For the levels where the group controller uses logical keys, the user obtains the keys on the path to the root. For the levels where the group controller uses complementary keys, the user gets the keys associated with the siblings of its ancestors. For example in Figure 1, user u_{1112} gets the group key (k_g), the logical keys (k_1, k_{11} and k_{1112}), and the complementary keys (c_{112}, c_{113}, c_{114}, c_{1111}, c_{1113} and c_{1114}). Finally, we use $k(m)$ to denote that message m is encrypted using the key k and, hence, only users that have the key k can obtain m.

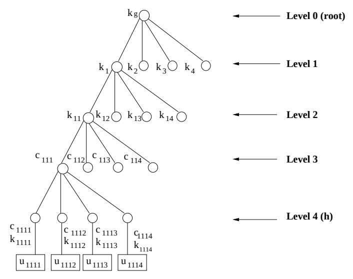


Fig. 1. Partial view of key tree

Multicast Tree. In our key distribution algorithm, we assume that the group controller distributes data and key update messages to the users and, hence the root of the multicast tree is the group controller. Any multicast protocol

IP [9–12] or overlay [13–15] can be used to build the multicast tree; our approach is independent of the protocol used to build the multicast tree. We define the *parent* of a user x, say $parent.x$, as the next hop node on the path from x to the group controller. For an intermediate node, we consider its children and all other nodes reachable through its children as its descendents. Each intermediate node performs reverse path forwarding, i.e., a message from the group controller is replicated on all outgoing links except on the link on which the message was received. We define a Key Distribution (KD) aware intermediate node as a node which supports our key distribution algorithm. Unless otherwise specified, in our paper, we assume that all intermediate nodes in the multicast tree are Key Distribution (KD) aware.

Problem of Key Distribution. In the current key management algorithms [1,2], when group membership changes, the group controller changes the keys in the key tree and securely broadcasts the new keys. Since all users do not need all the keys, this mode of key distribution is not efficient. Hence, we focus on the key distribution in algorithms where each user receives only a small subset of keys that includes all the keys it needs.

Previous Solution for Key Distribution. In [16], to distribute the changed keys in the key tree, the group controller encrypts the keys at the higher levels using changed keys at the lower levels. For example, when u_{1111} leaves, to distribute a new key k'_1, the group controller generates the messages $k'_{11}(k'_1)$, $k_{12}(k'_1)$, $k_{13}(k'_1)$ and $k_{14}(k'_1)$, where the key k'_{11} is already distributed using keys further down in the key tree. Before transmitting these messages, the group controller broadcasts the identifier of the leaving user to the current users. Using this information and knowledge of the structure of the key tree, each user calculates the level numbers of the changed keys it needs. This information is propagated towards the group controller. Upon reception of replies, the group controller transmits the key update messages and includes the level number, at which the key is changed, in each message. When an intermediate node receives a key update message with a level number l, it forwards the message to its descendants only if some descendant of l had requested that key. The main shortcoming of this key distribution algorithm is that, it is executed for every membership change which causes delay at the users for receiving key updates.

3 Proposed Improvements for Key Distribution

In this section, we identify our approach for reducing the cost of key distribution. In Section 3.1, we describe our descendant tracking scheme that enables the intermediate nodes to approximately track its descendants. In Section 3.2, we describe our algorithm for assigning identifiers to users. This algorithm can be used to improve the performance of the key distribution algorithms in Section 3.1 and in [16].

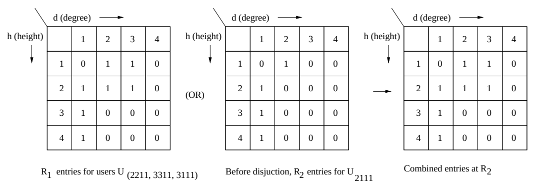
3.1 Descendant Tracking Scheme

A simple method to track descendents is to store the identity of each descendant user and forward the key update message only if any descendant user needs this key. However, this straightforward solution requires each intermediate node to store a large amount of information, especially in large groups.

To describe our scheme, first, we define the steady state configuration of the multicast tree. Then, we describe the technique used at the intermediate nodes to update the descendant tracking information to account for group membership changes. Finally, we describe how keys are forwarded by the intermediate nodes.

Steady State Configuration. At each intermediate node, we store a h x d matrix called DT (Descendant Tracker), where h and d are, respectively, the height and degree of the key tree. Each element in DT is a single bit. Thus, the information kept in DT is very small, even for large groups. For example, for a group of 1024 users, where the group controller maintains a key tree of degree 4 and height 5, each intermediate node stores only 20 bits of information in DT. To track a descendant user with identifier $u_{i_1 i_2 .. i_h}$, at an intermediate node, we set the elements, $DT[1, i_1], DT[2, i_2], ..DT[h, i_h]$, to 1.

When the group is initialized, the group controller assigns each user a unique identifier based on its location in the key tree. The users who are leaves in the multicast tree record their identifiers in their DT matrices. The intermediate nodes request their children for the DT information. The DT entries of a parent are the disjunction (binary OR) of the corresponding entries of its children. As an illustration, in Figure 2, we show the DT entries at a leaf node, R_1 with users U_{2211}, U_{3311} and U_{3111}, and the disjunction of these entries by its parent node, R_2 with user U_{2111}. Although, the descendants identified in a DT matrix are a superset of the actual descendant users, the DT matrix provides sufficient information about the descendants to reduce traffic.


d (degree)
h (height)
(OR)
R_1 entries for users $U_{(2211, 3311, 3111)}$
Before disjuction, R_2 entries for U_{2111}
Combined entries at R_2

	1	2	3	4
1	0	1	1	0
2	1	1	1	0
3	1	0	0	0
4	1	0	0	0

	1	2	3	4
1	0	1	0	0
2	1	0	0	0
3	1	0	0	0
4	1	0	0	0

	1	2	3	4
1	0	1	1	0
2	1	1	1	0
3	1	0	0	0
4	1	0	0	0

Fig. 2. DT entries at intermediate nodes R_1 and R_2

Tasks for Join. When a new user joins the group, the group controller distributes any keys, the new user needs, using a secure unicast channel. Also, the group controller distributes the new keys to the current users, where the intermediate nodes perform appropriate forwarding using the existing DT entries. The new user receives its identifier from the group controller and provides this information to its local intermediate node. The local intermediate node updates its DT matrix. Further, if the DT matrix has changed, it propagates this information to its ancestors in the multicast tree.

Tasks for Leave. When a user leaves the group, we do not update the DT entries at the intermediate nodes immediately. We perform an update of the DT entries only when the number of group membership changes exceeds a threshold level. Although this could cause some messages to traverse extra links, updating the DT matrix periodically allows us to achieve a tradeoff between the processing overhead and the amount of bandwidth reduced.

Forwarding Key Update Messages Encrypted with Logical Keys. In a key tree with only logical keys [1], each user knows all the keys associated with its ancestors. Thus, based on the naming scheme of the key tree from Section 2, the label of every key a user knows is a prefix of the user's identifier. Now, consider the case when some user, say u_{1112}, leaves the group. The group controller changes all the keys known to u_{1112} and distributes them to the remaining users who need these keys.

To determine if a key update is needed by any descendants of an intermediate node, we need to determine whether the label of the encrypting logical key is a prefix of at least one descendant in the intermediate node's DT matrix. To allow the intermediate nodes to make this identification, the group controller includes the label of the encrypting logical key in the key update message and transmits it to the users. For example, to send $k_{12}(k'_1)$, the group controller appends the label 12, to the encrypted message. To identify if the label $l_1, ..l_k$ of the encrypting logical key is a prefix to any descendant user, an intermediate node checks whether the DT elements, $DT[1,\, l_1]$,..,$DT[k,\, l_k]$ are all set to 1.

Forwarding Key Update Messages Encrypted with Complementary Keys. We note that, in a key tree with only complementary keys [2], each user knows the keys associated with the siblings of its ancestor. Thus, the labels of these keys differ in the last position from any prefix of the user's identifier.

To determine if a key update is needed by a descendant of an intermediate node, we need to determine whether the label of the encrypting complementary key differs in the last position from a prefix of a descendant's identifier. As in the case of transmitting keys encrypted with logical keys, the group controller appends the label of the encrypting complementary key to the key update message. Thus, to distribute $c'_{112}(c'_{12})$, the group controller appends the label 112 to the message. To verify that the label l_1, l_2,.., l_k of the encrypting complementary key is matched, an intermediate node checks whether the entries, $DT[1,\, l_1]$, $DT[2,\, l_2]$ .., $DT[k-1,\, l_{k-1}]$ and any entry $DT[k,\, l_p]$ where $p \neq k$, are set to 1.

3.2 User Identifier Assignment Algorithm

The key distribution algorithms we described in Sections 2 (from [16]) and 3.1, route the key update messages based on the identity of the descendants. The performance of these algorithms can be improved if the distribution of leaf nodes (group users) in the multicast tree corresponds to the distribution of the leaf nodes in the key tree. In this ideal scenario, users close to each other in the multicast tree will need almost the same key update messages and hence, the cost of key distribution would be low. While such a scenario is not always possible, one heuristic to achieve a near ideal scenario is to assign a joining user a logical identifier that is closer to the logical identifiers of users who are close to this user in the multicast tree. In this section, we use this heuristic to describe a user identifier assignment algorithm.

When a user sends a join request, the group controller communicates with this user using a secure unicast channel and learns about the location of the user in the multicast tree. Now, the group controller can use a program such as *mtrace* [17] to identify other nearby users in the multicast tree and record their logical identifiers. The group controller selects a user $u_{i_1 i_2 .. i_t}$, at the first intermediate node which is closest to the joining user. To assign an identifier to the joining user, the group controller selects an identifier $u_{i_1 i_2 .. i_p}$ such that $i_p \neq i_t$ and $1 \leq p, t \leq d$, i.e., the identifiers differ in the last position. The group controller assigns this identifier to the joining user, if it is not already assigned to any other user. If this attempt fails, the group controller repeats this process with another user at the first intermediate node. As an illustration, consider that the group controller selects the user u_{1111} at the first intermediate node. The group controller generates the identifiers $1112, 1113, 1114$, which differ in the last position with 1111. If any of these identifiers are still available, the group controller assigns one of them to the joining user. If no more users exist at the first intermediate node, the group controller selects users at the second intermediate node and so on.

In our assignment algorithm, as the intermediate nodes are further away from the joining user, the group controller successively moves up the position at which the identifiers differ. For example, the group controller selects a user, $u_{i_1 i_2 .. i_k i_l}$, at the second intermediate node, and tries to assign the joining user, $u_{i_1 i_2 .. i_p i_l}$, such that $i_p \neq i_k$, i.e., the identifiers now differ in the second last position. The group controller repeats this process until it successfully assigns an identifier or stops, if the only users found are very close to itself.

In case the group controller finds that the only intermediate nodes with users are very close to itself, the group controller assigns the next available identifier to the joining user. We do not select the logical identifiers of users close to the group controller due to two reasons. The first reason is that these users are not in the proximity of the joining user. The second reason is that these users, being close to the group controller, receive almost all of the key update traffic anyway and, thus, there would be no performance gain if the joining user is logically closer to these users.

4 Simulation Results and Analysis

We simulated our algorithms using the NS2 network simulator [18]. We performed experiments on randomly generated network topologies for groups of 256, 512 and 768 users. We used the CBT [9] protocol to build the multicast tree with the group controller as the root node. For each experiment, we selected a random set of users to join or leave the group and recorded the number of messages in the multicast tree over the entire multicast session. We measure the total bandwidth reduction achieved in our algorithms using the formula: $TBRR = \frac{BW_{broadcast} - BW_{optimised}}{BW_{broadcast}}$, where BW stands for bandwidth and $TBRR$ stands for total bandwidth reduction ratio. We also measure the hopwise breakup of the $TBRR$ for a given network topology, termed $PBRR$, which gives an overview of the performance of our algorithms as a function of the network distance away from the group controller.

We conducted experiments on three key management algorithms. The first algorithm is the logical key hierarchy algorithm, referred as LKH, in [1]. The second and third algorithms are from our earlier work which appears in [2]. In the second algorithm, the group controller uses complementary keys at every level in the key tree. In the third algorithm, the group controller uses both logical and complementary keys at every level in the key tree. We refer to these algorithms, respectively, as CKH and $LKH + CKH$. For comparison purposes we also simulated the previous key distribution solution [16] we described in Section 2. We refer to the various key distribution algorithms as follows: (a) *Id-based* – our key distribution algorithm from Section 3.1, (b) *Id-based-cluster* – combination of algorithms in Sections 3.1 and 3.2, (c) *Level-based* – the key distribution algorithm from [16] that is described in Section 2, and (d) *Level-based-cluster* – combination of algorithms in [16] and 3.2.

In Figure 3, we plot the $TBRR$ for a group of 512 users. The $TBRR$ achieved is in the range of 20-45%. We observe that using our identifier assignment algorithm improves the $TBRR$ of our key distribution algorithm as well as that of the level based key distribution algorithm.

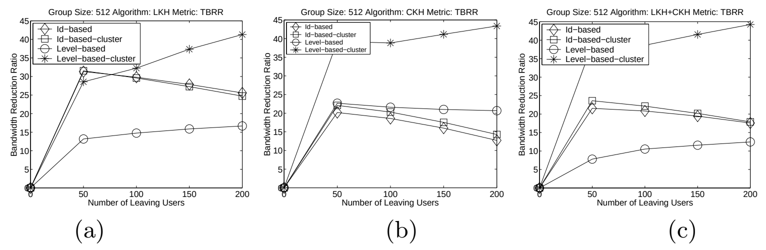
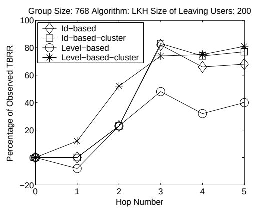
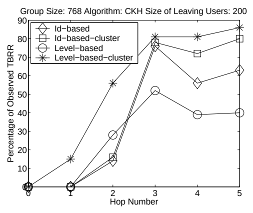


Fig. 3. TBRR for a group of 512 users using (a) LKH (b) CKH and (c) LKH+CKH

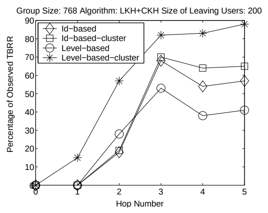
In Figure 4, we plot the $PBRR$ for a group of 768 users. The $PBRR$ value for *Hop Number* 1 indicates the $TBRR$ observed between hop 0 (group controller) and hop 1 (immediate children of group controller) and so on. From Figure 4, we observe that the *Level-based* algorithm causes more link stress near the group controller due to the responses by the users for each membership change. We note that, this problem is remedied by using our identifier assignment algorithm which reduces the link stress near the group controller in this algorithm.



(a)



(b)



(c)

Fig. 4. PBRR for a group of 768 users using (a) LKH (b) CKH and (c) LKH+CKH

5 Conclusion

In this paper, we addressed the problem of distributing key updates to users in secure dynamic groups. Towards this end, we described a descendant tracking scheme to track the descendants of the intermediate nodes in the multicast tree. In our descendant tracking scheme, each intermediate node stores a small information about its descendants. Next, we described the forwarding mechanisms used by the intermediate nodes based on the descendant tracking information. Each intermediate node forwards an encrypted key update message only if it believes that its descendants know the encrypting key. Using simulation results we showed that our key distribution algorithms reduce the bandwidth needed for distributing key updates in the key management algorithms in [1, 2] by upto 55% when compared to the broadcast of key updates.

Also, we described an algorithm for assigning identifiers to group users so that users who are physically close in the multicast tree are assigned logically close identifiers. We showed that our assignment algorithm improves the performance of our key distribution algorithm as well as that of the previous solution in [16]. Our key distribution algorithm can also be used to distribute data messages in secure interval multicast [19] where the group controller needs to send a message securely to a subset of the users.

For overlay multicast protocols [13–15], where the end hosts attempt to reduce bandwidth usage, the use of our key distribution algorithm results in better performance. The processing overhead of users in overlay multicast protocols [13–15] is high as the users need to constantly monitor and reconfigure the overlay links and route multicast data. Our key distribution algorithm reduces the processing overhead of the users by reducing the key update traffic that the users need to process.

For our key distribution algorithm it is not necessary that all the intermediate nodes store the DT matrices. We note that, a few selected nodes at appropriate points in the multicast tree are sufficient. We are currently exploring the selection techniques for choosing the best set of intermediate nodes which will participate in the key distribution algorithm. Also, many overlay multicast protocols maintain more links connecting the users. We are exploring methods to exploit the added connectivity to distribute the key updates more efficiently.

References

[1] Chung Kei Wong, Mohamed Gouda, and Simon S. Lam. Secure group communications using key graphs. *IEEE/ACM Transactions on Networking*, 2000.

[2] Sandeep S. Kulkarni and Bezawada Bruhadeshwar. Adaptive rekeying for secure multicast. *IEEE/IEICE Special issue on Communications: Transactions on Communications*, E86-B(10):2948–2956, October 2003.

[3] Debby M. Wallner, Eric J. Harder, and Ryan C. Agee. Key management for multicast: Issues and architectures. RFC 2627.

[4] D.McGrew and A.Sherman. Key establishment in large dynamic groups using one-way function trees. Manuscript.

[5] H.Harney and C.Muckenhirn. Group key management protocol (GKMP) specification. RFC 2093, July 1997.

[6] S.Mittra. Iolus: A framework for scalable secure multicasting. In *Proc. ACM SIGCOMM'97*, pages 277–288, 1997.

[7] Sanjeev Setia, Sencun Zhu, and Sushil Jajodia. A comparative performance analysis of reliable group rekey transport protocols for secure multicast. In *Performance Evaluation, special issue on the Proceedings of Performance 2002*, volume 49, pages 21–41, Rome, Italy, 2002.

[8] Y. Richard Yang, X. Steve Li, X. Brian Zhang, and Simon S. Lam. Reliable group rekeying: A performance analysis. In *Proceedings ACM SIGCOMM 2001*, San Diego, August 2001.

[9] A.J.Ballardie, P.F.Francis, and J.Crowcroft. Core based trees. In *Proceedings of the ACM SIGCOMM*, October 1993.

[10] T.Pusateri. Distance vector multicast routing protocol. IETF Draft, update to RFC 1075, draft-ietf-idmr-dvmrp-v3-06.txt, June 1998.

[11] S.Deering et al. Protocol independent multicast, sparse mode protocol: Specification. IETF Draft, work in progress, 1995.

[12] S.Deering et al. Protocol independent multicast (pim), dense mode protocol: Specification. IETF Draft, work in progress, 1995.

[13] Y.-H.Chu, S.G.Rao, S.Seshan, and H.Zhang. A case for end system multicast. *IEEE Journal on Selected Areas in Communications*, 20(8):1456–1471, October 2002.

[14] B.Zhang, S.Jamin, and L.Zhang. Host multicast: A framework for delivering multicast to end users. In *IEEE INFOCOM*, March 2000.

[15] J.Liebeherr, M.Nahas, and W.Si. Application-layer multicasting with delaunay triangulation overlays. *IEEE Journal on Selected Areas in Communications*, 20(8):1472–1488, October 2002.

[16] Di Pietro, L. V. Mancini, Y. W. Law, S. Etalle, and P. J. M. Havinga. Lkhw: A directed diffusion-based secure multicast scheme for wireless sensor networks. In *32nd Int. Conf. on Parallel Processing Workshops (ICPP)*, pages 397–406, October 2003.

[17] Bill Fenner and Steve Casner. A traceroute facility for ip multicast. Internet Draft, July 2000.

[18] Ns. ucb/lbnl/vint network simulator - ns (version 2). http://www-mash.cs.berkeley.edu/ns.

[19] Mohamed G. Gouda, Chin-Tser Huang, and E.N.Elnozahy. Key trees and the security of interval multicast. In *22nd International Conference on Distributed Systems*, pages 467–468, 2002.