Scalable Secure Group Communication over IP Multicast

Suman Banerjee, Bobby Bhattacharjee

Abstract— **We introduce and analyze a scalable re-keying scheme for implementing secure group communications IP multicast. We show that our scheme incurs constant processing, message, and storage overhead for a re-key operation when a single member joins or leaves the group, and logarithmic overhead for bulk simultaneous changes to the group membership. These bounds hold even when group dynamics are not known a-priori.**

Our re-keying algorithm requires a particular clustering of the members of the secure multicast group. We describe a protocol to achieve such clustering and show that it is feasible to efficiently cluster members over realistic Internet-like topologies. We evaluate the overhead of our own re-keying scheme and also of previously published schemes via simulation over an Internet topology map containing over 280,000 routers. Through analysis and detailed simulations, we show that this re-keying scheme performs better than previous schemes for a single change to group membership. Further, for bulk group changes, our algorithm outperforms all previously known schemes by several orders of magnitude in terms of actual bandwidth usage, processing costs and storage requirements.

Keywords— **Group Communication, Hierarchy, Multicast, Security**

I. Introduction

IP multicast enables scalable wide-area multi-party applications over the Internet. In this paper, we describe a new algorithm for scalable, secure group communication over IP multicast. Our algorithm can be implemented over base IP multicast and does not require router support beyond best-effort forwarding. Since our scheme is completely end-host based, it can be used to implement group security over IP multicast-enabled untrusted, insecure networks.

A. Group Keys and Re-keying Groups

Many secure group communication systems [12], [2], [25], [9], [1], [7], [4], [15], [16], including ours, rely on the notion of a "group key" — a secret known only to the members of the secure communication group[1]. Once a group key is distributed to all current members of the multicast group, secure messages can be sent encrypted with the group key. The overall security of the group depends wholly on the secrecy and the strength of the group key. Since every group member has the group key, sending a message involves only a single encryption at the sender and a single decryption at each receiver. The routers on the way treat each message no different than any other IP multicast datagram. The only problem left to solve is to scalably and securely establish a group key known to all (and only) the members of the secure multicast group.

Since we assume the network infrastructure is insecure, it is possible for non-members to eavesdrop on the multicast group and store encrypted messages (that they cannot decrypt). It is also possible for members who have left the group to continue to decrypt messages and for new members to decrypt messages they had stored previously. Therefore, during each membership change, a *new* group key must be distributed and all subsequent communication must use this new key. This is the process of *group re-keying*: establishing a new group key upon a membership change in the secure multicast group. Note that depending on the requirements of the application, it may or may not be necessary to re-key the group when a new member joins; but it is almost always necessary to re-key the group when a member leaves.

Clearly, the overhead of the re-keying can be reduced (at the cost of reduced security) by *batching* the re-key operations, e.g. not re-keying every time there is a membership change but re-keying periodically or when a the group membership has changed sufficiently. Batching is important since re-keying operations affect every single member in the group, and can potentially be very expensive for large groups. Thus, re-keying schemes designed for large groups must be efficient when handling bulk simultaneous changes to the group membership.

The simplest solution for re-keying involves a pair-wise secure exchange of the group key between a central key server and each group member [12]. Unfortunately, this scheme incurs a $O(N)$ overhead, where N is the number of group members, and is not viable for large groups. A particularly elegant protocol using "logical key hierarchies" were independently presented in [25], [9]. This is the first protocol that describes a scheme that incurs sub-linear overhead for single membership changes to the group. In [7], a different scheme using boolean minimization techniques is described that is efficient for bulk membership changes. This technique, however, is susceptible to a collusion problem, where departed group members can collude to obtain future group keys. Both these schemes reduce the overhead for the group re-keying operation for single membership change to $O(\log N)$. If more information is known about group dynamics, then it is possible to do better. MARKS [4] is a scheme that assumes that the duration over which a member stays attached to the group is known at the time the member joins. Using this information a constant overhead solution is presented in [4]. In this paper, we present a constant processing, message, and stor-

The authors are with the Department of Computer Science, University of Maryland, College Park, MD 20742, USA. Emails: {suman,bobby}@cs.umd.edu

[1] It is possible to construct secure group communication systems without using a secret shared between all members. In such systems, trusted intermediaries, e.g. secure routers, must encrypt and decrypt messages en-route.

age overhead solution for the general problem when the membership durations are not known; to the best of our knowledge, ours is the first scheme with a provable constant bound. Further, we show that our scheme can handle $O(N)$ simultaneous changes to the group membership in $O(\log N)$ processing, and bandwidth complexity; this is a significant improvement on the previously known $O(N)$ bounds [7]. Obviously, the reduction of re-keying costs for single membership change from $O(\log N)$ to $O(1)$ is not significant unless the size of the group is very large. In simulations, our re-keying scheme performs better than previous schemes for single group changes. However, for bulk group changes, our algorithm outperforms all previously known schemes by *orders of magnitude* in terms of bandwidth usage, processing costs and storage requirements.

B. Overview of Our Approach

Our scheme is based on a particular size-bounded, non-overlapping, clustering of multicast group members, which we call "spatial clustering". Spatial clustering assures that members in the same cluster are near each other in the multicast tree. The constant overhead re-keying scheme is implemented using a hierarchy of spatial clusters. Our re-keying scheme has low communication overhead precisely because it is based upon a spatial partitioning of the multicast group which allows the key distribution scheme to exploit the parallelism inherent in different parts of a multicast tree.

This paper has two main contributions:

- We describe an efficient re-keying algorithm for implementing group security over IP multicast. Our analysis shows that this scheme has constant overhead for a single group re-key even when the group dynamics are not known *a-priori*. Further, we show that the overhead for bulk simultaneous changes is logarithmic in the number of group members. Both these results improve on previously known bounds.
- We present detailed simulation of our own scheme, and also of previously published schemes including [25], [7]. Compared to previously published work, our simulation results better measure the per router packet and bandwidth overheads inherent in implementing secure group communication schemes over IP multicast.

We also describe an algorithm based on IP multicast for creating bounded-size non-overlapping clusters. Such clustering is potentially useful beyond secure multicast; however, due to space restrictions, we do not explore this topic further in this paper.

C. Roadmap

In the next section, we describe existing approaches for key distribution for secure multicast. In Section III, we describe our multicast key distribution scheme. In Section IV, we describe the underlying clustering algorithm required to implement our re-keying scheme. In Section V, we describe the experimental methodology for our simulation experiments. In Section VI, we present simulation results comparing our scheme versus existing approaches and conclude in Section VII. We present a formal protocol specification and asymptotic complexity analysis for the re-keying algorithm in an Appendix.

II. Existing Approaches for Secure Multicast

Group Key Management Protocol (GKMP) [12] is a simple group management protocol in which a *Group Key Controller* is responsible for generating group keys. In GKMP, static keys —pairwise shared between the *Group Key Controller* and each group member— are used to establish a group key. The Scalable Multicast Key Distribution (SMKD) [2] works in conjunction with the Core Based Tree (CBT) [3] multicast protocol to securely distribute the multicast group key. The CBT root router initially operates as the entity responsible for generation and distribution of the group key. This responsibility is delegated to other routers as they join the delivery tree. SMKD requires explicit router support, and does not scalably solve the problem of group re-keying.

In Iolus [15], the scalability of re-keying is handled by dividing the secure multicast group into multiple sub-groups. Security in each sub-group is managed by a Group Security Agent (GSA). The GSAs (and by consequence, the sub-groups) are *statically configured* and located in different parts of the Internet. Each sub-group has its own sub-group key that is managed by the relevant GSA. Membership changes in any sub-group require local re-keying of the sub-group key. If a global shared group key is used for the data, this key needs to be distributed to members in each sub-group by sub-group specific means. Alternatively, if no global group key is used, changes in one sub-group do not affect the others, but expensive data-path encryptions and decryptions are necessary. Since Iolus does not define sizes of subgroups etc., it is difficult to provide analytic bounds on Iolus' performance (and hence it is not included in the simulation results). It is, however, possible to use an Iolus-like infrastructure to implement our scheme, and we discuss the details of such an implementation and compare our scheme with Iolus in Section IV-C.

As mentioned in Section I, the MARKS [4] scheme defines a constant overhead key distribution protocol. More precisely, members in MARKS incur a one time cost which depends on the length of the time they stay in the group.

MARKS is based upon the premise that many applications, e.g. pre-paid or subscription pay-TV or pay-per-view, do not (or rarely) require premature eviction. Thus, the protocol assumes that the duration over which a member stays in the group is known when the member joins. For single changes to the group, our scheme provides better performance bounds without the known *a-priori* membership duration requirement.

Two protocols that provably incurred sub-linear group re-keying overheads for each membership change are the Logical Key Hierarchies (LKH) scheme [9], [25] and the Boolean Minimization scheme [7]. Both these protocols and our protocol address the re-keying problem under the same assumptions (unlike MARKS), and we outline their operation in the next section. The Boolean Minimization

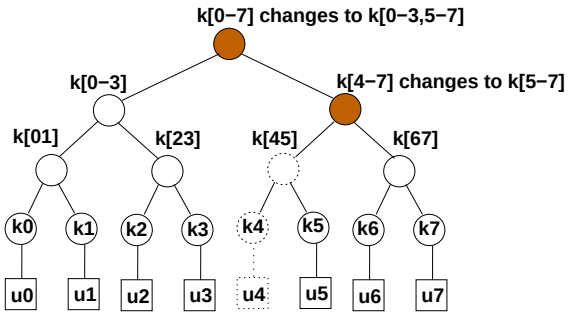


Fig. 1. LKH Scheme

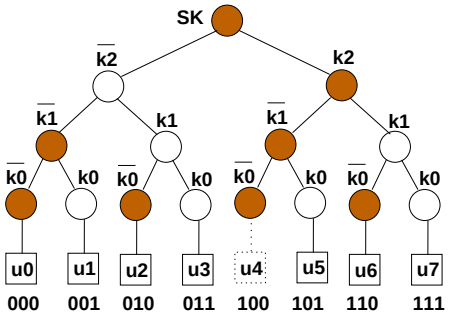


Fig. 2. Boolean Minimization Scheme

scheme, although efficient, suffers from a collusion problem as discussed later in this section. In our simulations, we compare the performance of our proposed scheme to both these schemes. There are two extensions to the LKH scheme: the scheme in [6] (called LKH+) uses input doubling functions to reduce number of re-keying messages in half, and the scheme in [14] describe how to effectively re-key during bulk membership changes. In our simulations, we have also implemented the protocols described in [14].

The LKH scheme defines a logical hierarchy *of keys* distributed between different sets of members, as shown in Figure 1. The leaf nodes on the tree represent the different members, while the circular nodes represent the different keys. Each member u_i possesses all the keys on its path to the root (from $k_i \rightarrow k[0-7]$ in Figure 1). The root key serves as the group key. A central key server is responsible for generating and distributing new keys as required. When a member (e.g. u_4 in the figure) leaves the group, all keys on the path from this member to the root needs to be changed. However, the updated keys can be multicast by the key server to the sub-groups instead of being unicast to individual members separately, and thus the re-keying cost is $O(\log N)$.

A distributed version of the LKH scheme was also proposed in [9], where trusted sub-root nodes handle group join and group re-key operations for subordinate users. In [25] different re-keying —key-, user-, and group-oriented— techniques for the LKH scheme are described, each with different processing, and message overheads. We have implemented all of these schemes in our simulation, and in our comparisons, we use the re-keying scheme that is most favorable to the LKH scheme for the particular experiment.

The boolean minimization technique [7] is another scheme that uses virtual key hierarchy-based scheme. There are $2 \log N$ auxiliary keys, namely $\{k_0, k_1, \ldots\}$ and $\{\overline{k_0}, \overline{k_1}, \ldots\}$ and one group key SK. Note that the keys k_x and $\overline{k_x}$ are not bitwise complements of each other, instead they represent two completely different keys. If $b_R \ldots b_1, b_0$ denotes the identifier of a member in binary notation, then that member has key SK and the auxiliary keys given by the following rule: For each i, if $b_i = 0$, then the member has key $\overline{k_i}$, else it has key k_i. This set of auxiliary keys held by each member can be represented using a binary key tree as shown in Figure 2. When a member leaves the group, all group key along with the auxiliary keys held by the member need to be changed. A new group key is generated by a group controller and is encrypted with the keys that are *complementary* to the keys held by the departing member. In the example in Figure 2, member u_4 leaves: the new group key SK' are therefore encrypted as $\{SK'\}_{\overline{k_2}}, \{SK'\}_{k_1}, \{SK'\}_{k_0}$ and distributed to the entire group. All and only the remaining members are able to decrypt the new group key. Further, a one-way hash function on the new group key is used to update the auxiliary keys known to u_4. Using boolean minimization techniques to determine how the group key is encrypted for distribution, this scheme is able to handle bulk membership changes more scalably than the LKH scheme.

However, the Boolean Minimization scheme is of limited interest because it suffers from a collusion problem. For example, in Figure 2, members u_0 and u_7 together have all the different keys in the hierarchy. Therefore, if they collude together after simultaneously departing from the group, they can decrypt any new group key that is distributed by the key server.

Balenson et. al. [1] define another efficient group key distribution mechanism based on a novel application of one-way function trees (OFT). This technique has overheads that are similar to the LKH scheme.

The VersaKey framework [24] introduces group re-keying schemes similar to the OFT approach and therefore, also has similar overheads. The key generation mechanism in this framework can be varied between completely centralized (where keys are generated by a single key server) and fully distributed (where keys are generated by any group member).

Naor et. al. [16] defines an efficient "subset-difference"-based group re-keying algorithm for bulk removals from the group. This scheme requires group members to store $\log^2 N$ keys and the group controller incurs a communication overhead of $2r$, where r is the number of members simultaneously leaving the group.

The Multicast Security (MSEC) Working Group in the IETF is currently pursuing an effort to standardize protocols for securing group network communication for large groups, with additional emphasis on groups that use network layer multicast for group data communication. The

working group charter classifies the work into three functional building blocks namely, the data security transforms functional building block, the group key management and group security association (GSA) building block and the group policy management functional building block. Our work proposes a scalable mechanism for one of these building blocks, namely the group key management and group security association (GSA) functional building block. This building block requires secure generation distribution, and update of the group cryptographic keys.

III. Secure Multicast using Clustering

In this section, we present our secure key distribution algorithm. For this description, we assume a member clustering protocol that maps multicast group members to clusters with the following properties:
Each cluster has between k and $2k-1$ members, for some fixed k. No two clusters share a member. Cluster members are close to each other on the multicast delivery path.
We describe an actual clustering protocol in Section IV. This protocol guarantees that no two clusters share more than one member. This weaker condition, however, is still sufficient to guarantee the clustering properties as required in our analysis of the re-keying algorithm.

A. Member Hierarchy for Key Distribution

Our key distribution scheme creates a member hierarchy as shown in the leftmost panel of Figure 3. A "layer" comprises of a set of members of the secure multicast group in the same level of the hierarchy. Layers are numbered sequentially with the lowest layer of the hierarchy being layer zero (denote by L_0). An instance of the clustering protocol is executed at each layer of the hierarchy to create a set of clusters, and all members of the secure multicast group are part of the lowest layer (L_0). The cluster leaders of all the clusters in layer L_i join layer L_{i+1}. For example, consider the arrangement of members in the initial configuration (Panel 0) of Figure 3. All ten members A–J are part of layer L_0. The clustering protocol has partitioned L_0 into three clusters: $[ABC]$, $[DEFJ]$, and $[GHI]$. (In referring to clusters in Figures 3 and 4 , we use the notation $[XY\ldots Z]$ to refer to a cluster with members $X, Y, \ldots, Z$.) The cluster leaders, C, E and H join layer L_1. Another instance of the clustering protocol executes at layer L_1 to create the single cluster $[CEH]$. The leader, H, of the layer L_1 cluster joins layer L_2 — the highest layer in this example. The procedure terminates when there is only a single member in any layer.

When a new member joins any layer, the clustering protocol places it into one of the clusters in that layer. Occasionally, arrival of a new member or the departure of an existing member from a layer can split or merge clusters. This decision is part of the clustering protocol.

B. Layer Keys, Cluster Keys, Key Servers and Group Communication

A secret *layer key* is associated with each layer of the hierarchy. A group member possesses a layer key for a specific layer if and only if it is a member of a cluster in that layer. Layer keys are generated, on-demand, by a *key server* whenever a new member joins or an existing member leaves any layer. A secret *cluster key* is associated with each cluster. Once again, a group member possesses a cluster key for a specific cluster if and only if it is a member of that cluster. The leader of each cluster is responsible for generating the cluster key for that cluster. Finally, in all clusters, a pair-wise key is shared between the cluster-leader and each cluster member. Since all members belong to L_0, the key for L_0 is used as a shared key for secure communication.

C. Key Distribution Protocol

The key distribution protocol ensures that:
the layer key of each layer is only available to members joined to that layer.
Therefore, whenever a member leaves (or joins) a layer, a new layer key is required for that layer. This ensures that the layer key of layer L_0 (which is the group key for the entire secure multicast group) is available only to members of L_0, i.e. all and only the current members of the multicast group. In the rest of this section, we use three examples to illustrate how our protocol efficiently changes layer keys and maintains security guarantees. In the examples, we assume that the cluster sizes must be bounded between 3 and 5. A formal specification for the protocol is given in the Appendix.

Example I: Member departure. In Figure 3, we first consider the case that member D leaves the secure multicast group. Since D belongs only to L_0, only the L_0 layer key needs to be changed. The following are the re-keying operations required :
1. A new L_0 layer key request is made by E, the cluster leader of $[DEFJ]$[2].Simultaneously, E sets up a new cluster key by pair-wise communication with each of the remaining members of this cluster (in this case F and J), as shown in Panel 1.
2. The key server generates the new L_0 layer key, and multicasts the new L_0 layer key to all members in the immediate higher layer, i.e. L_1, encrypted by the current L_1 layer key (Panel 1). (Recall that the members of layer L_1 are the cluster leaders of the clusters in layer L_0.)
3. On receiving the new L_0 layer key, the members of layer L_1 (i.e. C, E and H) extract the new layer L_0 key, and multicast it to the other members of their clusters in layer L_0, encrypted by the respective current cluster keys. Note that when member E transmits the new L_0 layer key in its L_0 cluster (now comprising of E, F and J), it uses the new cluster key that it had set up thus ensuring D cannot decrypt the new layer key. *The cluster multicasts occur in parallel and affect disjoint sets of the multicast group.*

Member Join. A joining member is assigned to a L_0 cluster by the clustering protocol at layer L_0. The cluster leader in this cluster generates and distributes a new

[2] If E leaves the group at the same time as D, then E's departure would cause H, the cluster leader of cluster $[CEH]$ in layer L_1, to initiate new layer key requests for both layers L_0 and L_1.

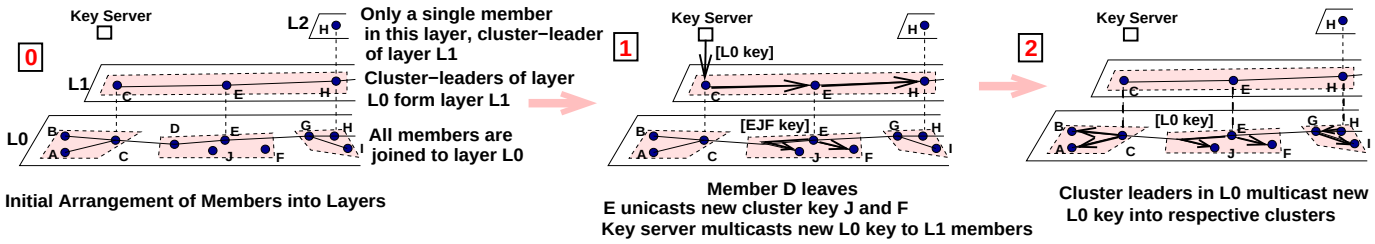


Fig. 3. Key distribution on a three layer hierarchy of members on the multicast delivery tree.

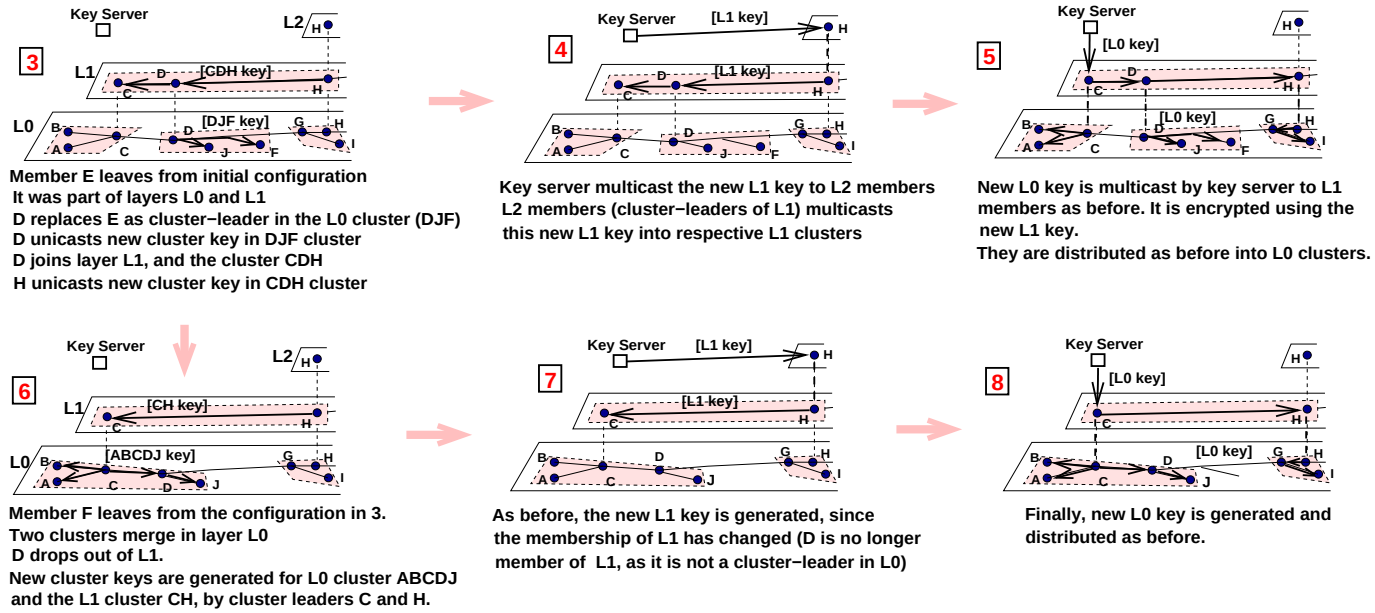


Fig. 4. Examples illustrating the key distribution scheme.

cluster key. Finally, the new L_0 layer key is distributed, as explained steps 2 and 3 in the previous example.

Example II: Cluster Leader Departure. In our next example, we refer to Figure 4, and consider the case when E, a cluster leader in layer L_0 and a member of both L_0 and L_1, leaves the original multicast group. The re-keying operations proceeds as follows :

1. Some member, D, in this example, is chosen as the new cluster leader of the L_0 cluster ($[DFJ]$). Being a leader in L_0, D joins layer L_1. The two affected clusters $[DFJ]$ in layer L_0 and $[CDH]$ in layer L_1 require new cluster keys. As shown in Panel 3, these keys are established by the cluster leaders by pair-wise unicast communication, using the respective pair-wise keys.
2. New layer keys, for both layers L_0 and L_1, are requested by the cluster leader (H) of the highest affected cluster ($[CDH]$). The new L_1 key is unicast to other members of the L_1 cluster, $[CDH]$, by its leader, H. The new L_0 key is multicast to layer L_1 encrypted using the new L_1 key. It is then multicast, using corresponding cluster keys, onto clusters $[ABC]$, $[GHI]$, and $[DJF]$ by C, H, and D, respectively

It should be clear that whenever there is a change in membership in any specific layer L_j for $j > 0$, there is a corresponding change in all lower layers L_i, $0 \leq i < j$. Thus, in this example, H, which is the leader of the affected cluster in the highest layer, immediately requests a new layer key for L_1 and all lower layers.

Example III: Cluster Reconfiguration. As a final example, we consider the case, when member F leaves the group from the configuration in Figure 4, Panel 3 (member E has already left). In this case, the DFJ cluster in layer L_0 shrinks to two members (D and J), violating size lower bound of 3. As a consequence, the clustering protocol merges the clusters DJ and ABC to create a cluster within the required size bound. Since, D is no longer a leader of a L_0 cluster, it must be removed from layer L_1. Both layers L_0 and L_1 are rekeyed as in the previous example.

D. Authentication and Access Control

Every secure communication group typically requires a single entity where access to the group is controlled. We call this entity the Authentication and Access Control Server (ACS). For any key distribution scheme, the ACS needs to maintain some state for each current member of the secure group.

When a new member, A, joins the secure group, it registers and authenticates itself with the ACS. A new member

needs to be authenticated by the ACS only the first time it joins the group. The ACS maintains the authenticated list for all the members of the group. As part of this registration A acquires a time-stamped credential Cred_A from the ACS, which is a digital certificate signed by the ACS. Subsequently, when A joins a specific cluster in any layer, with leader B. The members A and B exchange the credentials to mutually authenticate each other and establish the pair-wise key between the leader and the member. The members establish the pair-wise key using a computationally less expensive variant of the Diffie-Hellman key exchange protocol by leveraging the authentication provided by the ACS.

This authentication and key establishment scheme is described in [19] and we outline this protocol next. We assume that a prime number, p, a generator, g and the public key of the ACS are publicly known across the system. We use modulo p arithmetic in this section. For simplicity of notation we use g^α to represent $g^\alpha (\text{mod } p)$. The following is the authentication protocol pair-wise key establishment between the new member, A and a cluster leader, B:

1. A generates a random number α and sends g^α to the ACS.
2. The ACS authenticates A and returns a signed credential to A. This credential, Cred_A, is the pair $\langle A, g^\alpha \rangle$ signed by the ACS. B is a part of the secure group and therefore, already has its own credential Cred_B which is the pair $\langle B, g^\beta \rangle$ signed by the ACS, where β is a random number chosen by B during its authentication with the ACS.
3. A sends $\langle A, g^\alpha, \text{Cred}_A \rangle$ to B. Since the credential is signed by the ACS, B can authenticate this message to have been originated at A.
4. B sends a similar tuple $\langle B, g^\beta, \text{Cred}_B \rangle$ to A. Therefore, A can also authenticate B.
5. A creates a pair-wise key $K = (g^\beta)^\alpha$.
6. Similarly, B creates the same pair-wise key $K = (g^\alpha)^\beta$.

An eavesdropper cannot construct the key, K, by monitoring the network traffic and acquiring the transmitted values, g^α and g^β without knowing the values of α and β. Note that, α and β are never transmitted by A and B respectively.

In our protocol, we do not use K as the pair-wise key between A and B. Having computed K, the members A and B can compute a pair-wise key $\chi_i(A, B)$ as K^{γ_i}, where γ_i is obtained from some generator function $\Gamma(i) = \gamma_i$. Thus, choosing different values of i it is possible to obtain different pair-wise keys for a given K. Since the key, K is not used to encrypt any network traffic, it has a lower probability of compromise through traffic analysis. Therefore the same K can be used to generate multiple pair-wise keys if needed. The first pair-wise key can be computed using a single modular exponentiation by A (and analogously by B) as: $\chi_0(A, B) = (g^\beta)^{\alpha\gamma_0}$.

Lastly, note that the pair-wise session keys $\chi_i(A, B)$ are used for symmetric cryptographic operations. Therefore, it is acceptable to use a short key length for K and $\chi_i(A, B)$ (e.g. 64 bits).

The cost of authentication of a new member and generation of the pair-wise key between a member and its leader involves a digital signature verification and a modular exponentiation. The cost of the exponentiation increases linearly with the length of the key, and therefore, using a key with a short length helps in significantly reducing its computational costs. Note, that a set of values for g^α and g^β are pre-computed off-line and stored at the members before they join the secure group. Therefore, the on-line computational overhead is half of the full Diffie-Hellman protocol.

Asymmetric Key Operations. The mutual authentication and establishment of a pair-wise key between a cluster leader and other members is computationally more expensive than the encryption and decryption operations for cluster and layer re-key. This is because the former involves asymmetric key operations, while the latter can be performed using symmetric keys. However, we show that the average overheads at a member for asymmetric key operations is negligible.

Consider the case when cluster sizes vary between 8 and 15 (as used in the simulations later). In this case 92% of the entire group belongs solely to layer L_0. When one of these members leave the group, no authentication or pair-wise key establishment is required so long as the resulting size of the affected cluster stays 8 or higher. Assuming equi-probable leaves for members, more than 80% of the leave operations, therefore, require no asymmetric key operations. In the worst case a member leaves from the highest layer and only $O(k \log N)$ members need to perform asymmetric key operations. Similarly for more than 80% of the member join operations, only two members (the new joining member, and its layer L_0 cluster leader) require to perform asymmetric key operations.

Our analysis shows that the amortized number of asymmetric key operations per member for joins and leaves, aggregated over all layers, is $<< 1$.

The ACS can also arbitrate the removal of a member from the group. In such a situation, the ACS instructs the Key Server to issue a new layer L_0 key (i.e. the group key) for the group. All subsequent data messages are encrypted with the new group key. The ACS also multicasts the identifier of the revoked group member that caused this group key change. Any member that first realizes that it does not have the new group key invokes the distributed re-key operation as described in detail in the appendix.

E. Complexity Analysis

Along with the formal protocol specification, we have presented the complexity proofs for our algorithm in the Appendix. We show that in our scheme the processing and communication costs at a member for a single membership change to the group and the number of keys stored at each member is of constant order. Also the number of asymmetric key cryptographic operations per member is significantly small (i.e. $<< 1$). Only the communication overheads incurred at the different routers (links) on the multicast tree depend on the physical topology. We define the physical cluster topology as the set of all and

Scheme	Storage		Processing (single change)		Processing ($O(N)$ change)		Communication	
	Member	Key Server	Member	Key Server	Member	Key Server	Single change	$O(N)$ change
GKMP	2	$N+1$	1	N	$O(1)$	$O(N)$	$O(N)$	$O(N)$
Key Graphs	$\log_d N + 1$	$Nd/(d-1)$	≤ 2	$d(\log_d N - 1)$	$O(\log N)$	$O(N)$	$O(d \log N)$	$O(N)$
Bool. Min.	$\log_2 N + 1$	$2\log_2 N + 1$	1	$\log_2 N$	$O(1)$	$O(N)$	$O(\log N)$	$O(N)$
Spatial Clustering	≤ 4	$\log_k N$	≤ 2	≤ 2	$O(1)$	$O(\log N)$	$O(1)$	$O(\log N)$

TABLE I

AVERAGE COSTS FOR DIFFERENT SECURE MULTICAST KEY DISTRIBUTION SCHEMES. FOR $O(N)$, I.E. BULK, CHANGES TO THE GROUP MEMBERSHIP THE OVERHEAD COSTS ARE TYPICALLY MAXIMIZED WHEN $N/2$ MEMBERS SIMULTANEOUSLY LEAVE THE GROUP.

only those routers (links) that need to carry traffic due to intra-cluster communication. We consider two clusters to be non-overlapping if their physical cluster topologies do not share any common router. For the analysis of communication costs at routers (links), we consider the case when the clusters of a layer are non-overlapping. In this case, we show that the communication cost per router (link) on the multicast delivery tree is also constant for a single membership change [3]. However, depending on the underlying physical router (link) topology, it is not always possible to guarantee this non-overlapping property. However, through simulations on the large Internet map, we demonstrate that it is possible to achieve significantly low-overlap between clusters on realistic network topologies. In Section VI we provide simulation results to illustrate that even the communication costs at routers (links) for realistic topologies are largely constant for single membership changes.

Next we analyze the overheads for bulk changes to the group. When we batch process multiple changes (or equivalently, there are multiple simultaneous changes in the group), the processing cost at the key server and the messaging overheads have logarithmic bounds, which is a significant improvement over previously known bounds. It is important to note that our results do not violate the worst case logarithmic lower bounds for single change in group membership presented in [21]. While the worst case cost of our scheme is logarithmic, we improve the best and amortized re-keying costs from logarithmic in currently known schemes to constant in our scheme. We summarize our results and previously known results in Table I.

IV. SPATIAL CLUSTERING

In this section, we outline an algorithm to partition the members of a multicast group into fixed sizeclusters, as required by the re-keying algorithm. The input to the clustering algorithm is a *member overlay tree* which contains only the multicast group members as nodes. As members join the multicast group, we use a *member discovery protocol* to establish a parent for each new member in the member overlay tree. We describe the member discovery protocol next and the clustering protocol in Section IV-B.

[3]Note, that the processing, storage and communication requirements at *members* do not depend on the physical multicast path topology, and all the stated results hold without any topology-related assumptions.

A. Member Discovery Protocol

The member discovery protocol takes a multicast topology as input, and outputs a member overlay tree. It thus defines parent-child relationships among the different members of the multicast tree. This is the only component that is inherently tied to the network layer multicast scheme being used. Network layer multicast schemes create data delivery trees, which are broadly classified to be either source-based (DVMRP [23]) or shared (CBT [3]) trees, each of which can be unidirectional (PIM-SM [10]) or bi-directional (CBT). We have defined different member discovery protocols tailored for each of these different network layer multicast schemes. In this paper, we focus on a network layer multicast scheme that creates shared bi-directional trees (e.g. Core Based Tree protocol [3]) and only describe the appropriate member discovery protocol. This member discovery protocol uses mechanisms similar to the low overhead technique of fault isolation in multicast trees [17]. Let $d(u, v)$ denote the distance, in router hops, between the members x and y along the multicast delivery with source S. A member y is considered to be a parent of member x, if and only if the following three conditions hold:
C1: $d(S, y) \leq d(S, x)$. This condition ensures that the parent is closer to the source than the child.
C2: $\forall$ z that satisfy **C1**, $d(y, x) \leq d(z, x)$. This condition chooses the closest member that satisfies condition **C1**.
C3: $\forall$ w that satisfy **C2**, $y \leq w$ (lexicographically). For a set of members that are equidistant from the source and each other, multiple members may satisfy condition **C2**. In such cases, we pick the lexicographically smallest member as a simple symmetry breaking technique to prevent loops.

The protocol uses two periodic messages: the root S periodically multicasts a heartbeat packet to all the members of the group, from which each member infers their distance to S. Periodically each member, x, multicasts a TTL-scoped heartbeat message to parent and all of its children on the overlay tree. This message carries the tuple $\langle d(S, x), P(x) \rangle$, where $P(x)$ denotes x's parent in the overlay tree.

We explain the tree construction using an example: Consider the network in Figure 5. In Panel 0, E sends out a message with TTL five to reach its parent B. Assume a new member C joins the multicast group (as shown in Panel 1). Since C is part of the multicast group, the TTL-scoped message from B reaches C. Node C is able to infer the multicast distance between B and C and hence, using the two specified rules, concludes that B is its parent. If, however, the original messages from B does not reach C, then after a timeout, C initiates an Expanding Ring Search to locate

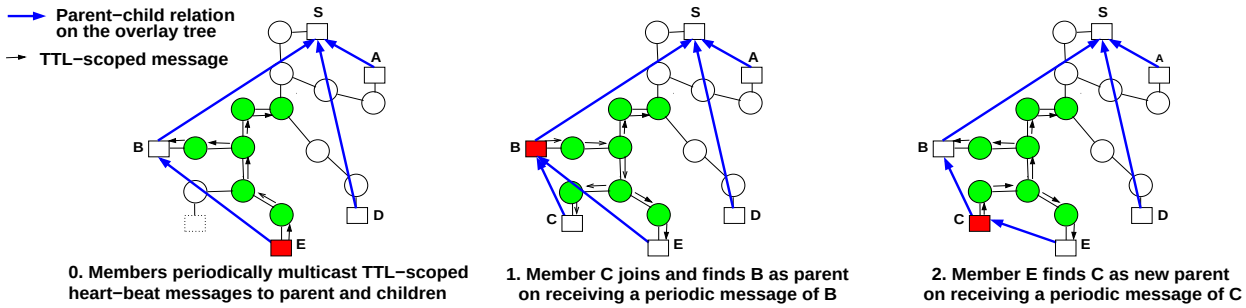


Fig. 5. Member discovery protocol example

a parent. Upon receiving a query from C, B updates its TTL-scope value such that its next heartbeat reaches C.

Finally, when C sends its periodic heartbeat scoped to reach B (Panel 2), it also reaches E. From condition **C1**, E realizes that C is its new parent on the overlay tree and adjust its TTL-scoping accordingly. B stops getting heartbeats from E and concludes that E is no longer its child. Note that even if the TTL-scoped message from C did not reach E, the heartbeat message from E, scoped to reach its current parent B, is guaranteed to reach C. C would realize that it is a better candidate to be E's parent and would adjust the TTL-scope of its heartbeat to reach E.

B. Clustering Protocol

The clustering protocol takes an integer k and a member overlay tree as input, and outputs clusters which are connected subsets of the tree, each between size k and $2k-1$. To create the clusters, the tree is logically traversed from the leaves, upwards to the root. In this traversal, whenever a set of members that fall within the size bounds is detected, they are grouped into a cluster. These members are considered pruned from the overlay tree for the remaining traversal. When this procedure terminates, it is possible that a single cluster, located at the root of the overlay tree, may have size less than k. Additionally, this protocol also guarantees that two clusters share no more than one member.

B.1 Protocol Description

We refer to clusters of size between k and $2k-1$ as *stable* clusters. Clusters of size less than k or greater than or equal $2k$ may occur in transience: we call these *unstable* clusters. In figure 6, A and B are stable clusters rooted at v.

Let τ_v denote the subtree, rooted at some node v which cannot be joined to any cluster rooted at v (doing so would render these clusters unstable). This subtree, which we call the *unstable subtree*, has to be joined to a cluster that is rooted at a node upstream of node v. In Figure 6, $\tau_v = C$. When the protocol stabilizes, C would be part of a cluster rooted at a node upstream of v or be part of the single unstable cluster rooted at the root of the overlay tree.

Protocol Operation. The clustering protocol proceeds as follows:

- Initially when a member u joins the multicast group, it creates an unstable cluster, comprising only of itself. This cluster is also an unstable subtree, i.e. $\tau_u = \{u\}$.
- Each member u periodically sends a message to its parent containing the value $|\tau_u|$.
- The periodic message from child v to parent u is either a notification of a new unstable subtree rooted at v or of an existing unstable subtree rooted at v.

If τ_v is an previously known subtree, then it part of some existing cluster (or the unstable subtree) rooted at u. In this case, u checks to see if the size of τ_v has changed. If the size has changed sufficiently, u may have to split or merge a cluster that τ_v is part of. If τ_v is part of the unstable subtree, u may now be able to create a new stable cluster. If the message is for previously unknown subtree, this new subtree is added to the unstable subtree rooted at u. Node u tries to merge the its new upstream cluster with its existing clusters. This procedure may cause a new stable cluster, rooted at u to be formed. All subtrees that cannot be put into any cluster form the new unstable subtree rooted at u.

We list the exact procedure for handling child messages in Figure 8.

Example. We illustrate the operation of the clustering protocol with the example shown in Figure 9.

- Initially, the overlay tree has a cluster A rooted at the member v. Member v has no unstable subtree, and sends $|\tau_v| = 0$ to its parent u. Member u has an unstable subtree, B, which is part of a cluster rooted upstream from u.
- At a later time (Panel 1), the cluster A reduces below size k, making it unstable and is advertised by v as an unstable subtree. As a consequence, at u, $|\tau_u| = 3k/4 + k/2 \geq k$. The procedure MergeSubtrees is called on the two subtrees A and B to create one single cluster. No unstable cluster is created rooted at u. So τ_u becomes empty.
- Next (as shown in Panel 2), a new member w joins the overlay tree. The member discovery protocol locates it between u and v but there is no change in the clusters.
- Eventually (see Panel 3), an entire subtree C of size $k/2$ joins w. This causes $|\tau_w|$ to increase beyond k and leads to the creation of a single cluster including subtrees A and C. The new τ_w becomes empty, and the subtree B again becomes a unstable subtree at u.

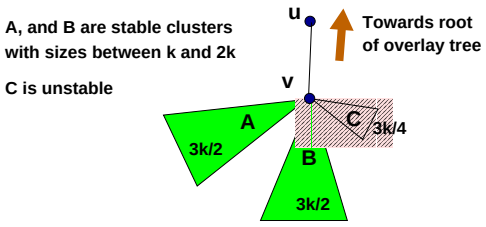


Fig. 6. Stable clusters and unstable subtrees.

Procedure : MERGESUBTREES$(u, STRootList, k)$
$TempClusters \leftarrow \perp$; $ThisCluster \leftarrow \perp$
for $v \in STRootList$
  $ThisCluster \leftarrow ThisCluster \cup T(v)$
  if $|ThisCluster| \geq k - 1$
    Add $\{u\}$ to $ThisCluster$
    Add $\{ThisCluster\}$ to $TempClusters$
    $ThisCluster \leftarrow \perp$
if $ThisCluster \neq \perp$
  $\tau_u \leftarrow ThisCluster$
return $TempClusters$

Fig. 7. Procedure MERGESUBTREES.

Procedure: PROCESSCHILDMESSAGE$(u, v, |\tau_v|_{new})$
$C_u \leftarrow \perp$; $\Delta_v \leftarrow |\tau_v|_{new} - |\tau_v|_{old}$
if v already belongs to some cluster C rooted at u
  Update cluster size of C to be $|C| + \Delta_v$
  if $|C| < k$ or $|C| \geq 2k$ { C *is disbanded* }
    $C_u \leftarrow \{w | w \in Children(u) \wedge w \in (C \cup \tau_u)\}$
else { v *was not part of any cluster rooted at* u *before, but may have been part of* τ_u *before* }
  { *Add* τ_v *to the unstable subtree,* τ_u *at* u }
  Update size of τ_u to be $|\tau_u| + \Delta_v$
  if $|\tau_u| \geq k$
    $C_u \leftarrow \{w | w \in Children(u) \wedge w \in \tau_u\}$
  else
    Merge τ_u with an existing cluster if possible
if $C_u \neq \perp$
  New Clusters rooted at $v \leftarrow$ MERGESUBTREES(u, C_u, k)

Fig. 8. Procedure PROCESSCHILDMESSAGE. C_u is a list of u's children s.t. the unstable subtrees rooted at these children need to be merged to create new clusters at u. Δ_v is the change in the subtree size between the previous message from child v and this message.

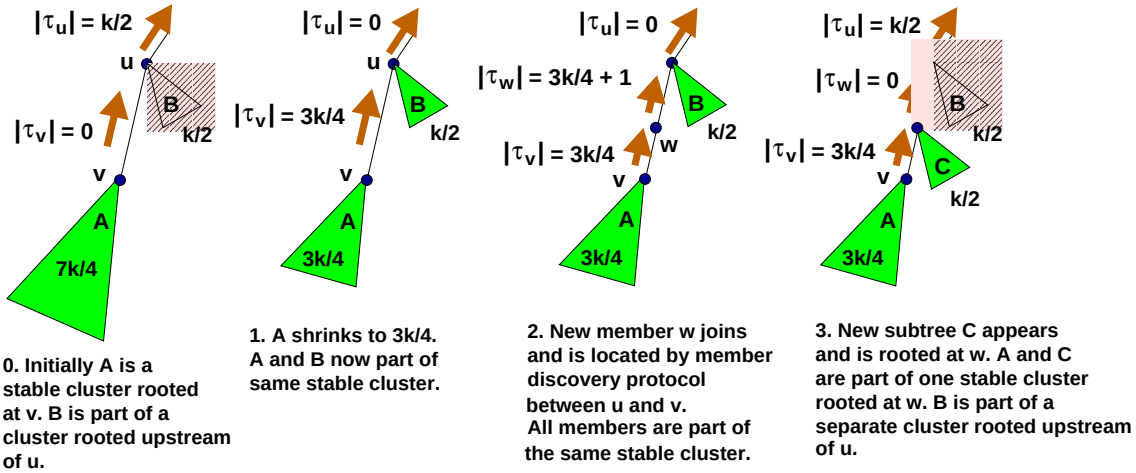


Fig. 9. Clustering protocol example

C. Implementing Secure Multicast over Clustering

We have implemented, in detailed simulations, our secure multicast solution using the clustering protocol as described in this section. Our solution proposes a framework that is similar to the Iolus framework [15]. In particular, the clusters in our scheme can be considered to be equivalent to subgroups in Iolus, and the cluster leaders behave like Group Security Agents (GSAs). There are, however, two basic differences:

1. The GSAs (in Iolus) are special entities located in the secure distribution tree, are chosen a-priori and the peering relationships between GSAs on the same level of the hierarchy are manually configured. In contrast, our proposed clustering protocol dynamically creates these clusters and their cluster leaders and adapts them with the changing structure of the secure group. Also the cluster leaders in our scheme are not specialized entities but are just regular members of the secure group.

2. The subgroups in Iolus are created based on administrative boundaries. This is one reason why these subgroups can be best assigned statically. In contrast, the clusters in our scheme are assigned based on topological proximity between members on the distribution tree. Additionally, we impose size bounds on the clusters. It is precisely these size bounded topological clusters that allow us to provide rigorous bounds on the overheads of our group re-keying scheme. The dynamic clustering that adapts based on the changing group membership is one of the main contributions of our work. Clearly, if similar size bounds and topological properties are imposed on the subgroups in Iolus, it will be possible to provide similar guarantees within the Iolus framework too.

A clustering-based implementation for group re-keying such as incurs an added overhead that is not present in other schemes. This is not a concern in Iolus since the sub-

groups are statically and manually configured. Further, in our solution, certain members have to act as cluster leaders, potentially leading to added trust and security issues. We address these two concerns next.

Clustering Overheads. The main overhead of clustering is the time it takes for clusters to stabilize, which is, in turn depends on the heartbeat period for the member discovery protocol. In Section VI-B.1, we show that clusters stabilize relatively quickly, e.g. in less than 2 seconds for even clusters of size between 20 and 39, for the topologies we consider. The stabilization time can be reduced further by using an adaptive heartbeat period algorithm that sends control traffic more frequently while clusters are forming or changing. Lastly, we note that our experiments show that the cluster stabilization time is independent of group size and depends primarily on the size and depth of clusters.

Trust. Our group re-keying protocol assumes a certain amount of inter-member cooperation. However, this cooperation does not make data communication in the group any less secure. Non-members cannot access the data encrypted by the current group key, while all current members of the group are able to correctly receive it. We ensure the latter by allowing cluster members to validate that their cluster leader does not distribute an incorrect or old layer key during a layer re-key. This is done as follows. The key-server signs the layer keys, when it distributes them. The cluster leaders re-distribute these digitally signed layer keys within their clusters. Cluster members can then verify the authenticity of the layer keys they receive from their leaders. The key server periodically multicasts the identity of the most current layer key of a layer that it has distributed, to all members of the layer. This allows all the members to verify that they have the correct and latest layer key.

The cluster leaders in the group re-keying protocol need to maintain cluster membership information. The cluster leader of the highest layer cluster is aware of the identity of $O(k \log N)$ other members. The leaders can, therefore, potentially compromise this partial group *membership* information to non-authenticated entities. However, the protocol can be augmented so that the members use a virtual identifier to identify themselves in the group. When a new member joins a group, the ACS authenticates it and now also issues a virtual identifier to the member along with the signed credential (Step 2 of Authentication). The credential certifies that the holder of the belongs to the group without divulging the actual identity of the member. The signed credential cannot be used for replay attacks, because the attacker cannot compute the pair-wise key, K, generated in Step 5, without knowing, α, the exponent chosen randomly by the actual member.

The cluster leaders are therefore aware of only the virtual identifier and not the actual identity of the group member. The IP addresses of cluster members are, however, visible to cluster leaders. Note that for any protocol, an attacker can also obtain such IP-level information by passively monitoring IGMP traffic. However, our protocol is somewhat more susceptible to such "address" leaks because the cluster leaders can identify group members in any network without having to resort to packet sniffing.

Ultimately, if it is necessary for an application to protect such IP-level information, additional mechanisms, e.g. use of a set of IP anonymizers [8], [18], [20], have to be used in order to obfuscate the IP header information. However, use of such anonymizers will incur additional overheads, such as intra-cluster control traffic at the anonymizers. Group members that do not require IP-level anonymity can still achieve user-level anonymity using the virtual identifier scheme without incurring any additional overheads.

V. Experimental Methodology

In Table I, we presented theoretical bounds comparing various key distribution schemes for secure multicast. In this section, we present simulation results to precisely quantify and compare the actual overheads for the different schemes.

A. Experimental Setup

We begin with a description of the topology we used to conduct our experiments, and describe the implementation of the various schemes.

A.0.a Network Topology and Multicast Support. We experimented the key distribution performance of the different schemes on a large Internet map obtained by the SCAN project[4]. This map, created using the Mercator [11] tool, contains about 280,000 IP routers discovered using traces on the Internet. About 50% of all the routers in the map were edge routers, to which we attached between 10,000 and 500,000 hosts routers uniformly at random for different experiments.

We have simulated network infrastructures that do and do not support *directed multicast*. Directed multicast allows a sender multicast a packet to individual subtree(s) rooted at a specific router on the multicast delivery tree. This is not currently a part of the IP multicast standard, but is extremely useful for many network services, e.g. NAK suppression for reliable multicast. It has been proposed in research, e.g. AIM [13], and is currently being considered by the IETF as part of the PGM [22], and GRA [5] efforts. In our experiments, many re-keying schemes show marked performance improvements when implemented over a directed multicast capable network.

In the absence of directed multicast, the same effect can the achieved by using a different multicast address for each subset of nodes that have to be addressed. For our scheme, which requires addressing clusters individually, and for some key graph schemes, using a different address for each addressable subgroup is not viable since the number of multicast addresses required would be on the order of the number of receivers. Instead, we use TTL-scoping to limit packets to a certain part of the multicast tree. The network load for using such *scoped multicast* is often significantly higher than pure directed multicast.

[4] See http://www.isi.edu/scan for details.

Scoped multicast using a single multicast address and directed multicast are two extremes in multicast addressing capabilities. It is possible to approximate the effects of directed multicast by using multiple multicast addresses, and using TTL-scoped multicast on each of these addresses. In our experiments, we consider directed multicast, scoped multicast with only one address, and scoped multicast with a small, fixed number of addresses.

A.1 Implementing LKH and Boolean Minimization

We have implemented both the LKH and Boolean minimization schemes along with our own scheme. The key graphs scheme described in [25] proposes three different re-keying techniques: user-, key-, and group-oriented re-keying. The bandwidth requirements of the user- and key-oriented re-keying are of the same order; therefore, we present results from key-oriented and group-oriented re-keying only. When directed multicast is available, key-oriented re-keying has the lowest overhead, while group-oriented re-keying has lowest overhead when scoped multicast has to be used. In our simulations, we use key oriented re-keying for the directed multicast scenario and the group oriented re-keying scheme for scoped multicast. For all implementations of the key graphs scheme, we used 4-ary key tree graphs, as was proved to be optimal in [25].

Along with the different re-keying strategies, we implemented two variants of the key graph algorithm, which we refer to in the results as *LKH-sequential* and *LKH-spatial*. These two schemes differ in the way the unique member identifier —which defines the position of a member in the key tree— is assigned to each group member. For both schemes, members join the multicast tree in a random sequence.

In the *LKH-sequential* scheme, members are assigned sequentially increasing identifiers in the order they join. In the *LKH-spatial* scheme, members are not assigned identifiers according to their join sequence. Instead, we assign the identifiers in sequence via a post-order traversal of the multicast topology of all the members. This identifier assignment of the *LKH-spatial* scheme ensures that members that share keys are close to each other on the multicast tree. However, in order to ensure nearness on the tree, identifiers have to be reassigned as members join and leave the group: this process accrues a cost linear in the number of group members. In our performance comparison in Section VI, we *ignore* this cost. It should also be noted that this post-order traversal based identifier assignment in the *LKH-spatial* scheme does not guarantee optimal network load for group re-keying; unfortunately, we found that the problem is NP-complete for all key trees with tree degree greater than two. However, with this simple identifier assignment heuristic, the performance of the *LKH-spatial* variant is significantly better than *LKH-sequential* and any real implementation of LKH on a large topology must address the identifier assignment issue. Lastly, as noted before, for bulk simultaneous to the key graph, we implemented the improved batch update algorithm described in [14].

For the boolean minimization scheme we implemented the key distribution scheme described in [7]. We used the publicly available logic minimization tool, *Espresso*[5] to determine the necessary boolean reductions. Since the key distribution technique in the boolean minimization scheme encrypts all the necessary keys to be updated and multicasts them in a single message, we report the network load only for scoped multicast.

B. Experiment Description

For each experiment:

- We generated a random set of group members attached to the leaf routers of the Mercator Internet map;
- Next we create the multicast delivery tree, using the CBT protocol [3].
- For our scheme, we compute the member overlay tree and member clusters using the the protocol described in Section IV. The clusters sizes were bounded between 8 and 15.
- We implement the key distribution protocol for our scheme, LKH, and the Boolean Minimization on the same set of members
- Next we choose, uniformly at random, a set of members to leave the multicast group simultaneously and record the storage, processing, message, and byte cost overheads at each member, router and link of the tree.

In Section VI, we will present results as we vary two parameters: the group size and the number of members that simultaneously leave the group. For each parameter, we repeated each experiment 100 times, each with a randomly chosen set of departing receivers, and obtained 95% confidence intervals. In all cases, our confidence intervals were extremely tight and we do not report them in the results.

B.0.a Performance Metrics. In our results in the next section, we report the following metrics:

- *Key-normalized byte count*: This is the network overhead for re-keying at a single group member, router or link assuming unit (1 byte) key size. The actual byte overhead can be obtained by simply scaling the key-normalized byte count by the key size, and accounting for packet headers, etc. This metric is a measure of the total bandwidth requirements of a scheme.
- *Packet load*: This is a count of the number of packets processed by the routers on the multicast tree. To compute the packet load, we assume a key size of 512 bits, and the maximum IP packet size of 576 bytes.
- *Storage and Processing Overhead*: These numbers refer to the number of keys stored at each group member and the number of cryptographic operations at each member. These numbers are independent of the particulars of the topology.

In our analytic results we derived exact bounds for storage, processing and communication overheads at the group members. This is accurately reflected in the simulations. As stated in Section III-E, only the communication costs

[5] *Espresso* is publicly available at http://www-cad.eecs.berkeley.edu/Software/software.html

for the routers depend on the exact physical network topology. Through our simulations we observed that the communication costs at routers for large realistic topologies closely match the bounds derived under the assumption that the physical cluster topologies of a layer are non-overlapping. We also verify this through independent observations of the overlap between physical cluster topologies.

VI. Simulation Results

We simulated the various key distribution schemes over multiple topologies and different network configurations. We first present the communication overheads incurred by network routers.

A. Communication overheads at routers

We begin with the results for the case when the underlying network is capable of directed multicast.

A.1 Directed multicast

In Table II we present the results for two different LKH implementations and compare them to our scheme. (Recall that since Boolean Minimization sends all messages to the entire multicast group, its performance is equivalent under directed and scoped multicast. We defer results for Boolean Minimization till the next section when we consider scoped multicast.) For each scheme, we have tabulated the average byte overhead at each router (assuming one byte keys) when a single member leaves and when member leaves are batch processed after 1% of the members leave the group. We analyze the actual number of bytes and IP packets in Section VI-A.4, and other batch sizes in Section VI-A.3.

It is clear from Table II that for a single leave, both the communication overheads at routers for *LKH-spatial* and our algorithm are on par (and that they both have lower costs than the *LKH-sequential* scheme). However, as members leave simultaneously, or if number of member departures are processed in batch (e.g. shown by the 1%-leave results), our clustering-based scheme significantly outperforms both LKH implementations. Finally, we note that our *LKH-spatial* variant of the LKH scheme outperforms the originally published scheme by a factor of $2.5-3$. (Obviously, this is without accounting for the member–identifier assignment overhead of *LKH-spatial* scheme.)

A.2 Scoped multicast

In Table III we compare the key-normalized byte load for different re-keying schemes using when TTL-scoped multicast is the only primitive available from the network. Group-oriented re-keying for LKH is optimally suited for scoped multicast, and the member assignment issue is not relevant (both *LKH-sequential* and *LKH-spatial* have identical performance).

We repeated the scoped multicast experiments with different number of multicast addresses for the spatial clustering scheme. In Table III, *Spatial-i* indicates that i different multicast addresses have been used by the spatial clusters for intra-cluster communication. We use a simple decentralized address assignment scheme in which each cluster picks one multicast address (among the i available) at random, independent of each other. Given i addresses, the optimal assignment of addresses to clusters such that the extra traffic is minimized is also NP-complete.

As is evident in Table III, for a single leave, Spatial-1 (our scheme using only a single multicast address) has higher communication overheads at routers than all other schemes. This is because even with TTL-scoping traffic local to a cluster "spills over" to a large part of the multicast tree, and hence, increases the potential overlap between the physical cluster topologies of different clusters of a layer. As more addresses are used, the TTL-scoping becomes more effective and the spill over effect is effectively mitigated. For our scheme, the availability of directed multicast represents the best possible scenario since it effectively minimizes the overlap between the physical cluster topologies of different clusters. We have included the costs under directed multicast as a lower bound.

There are two conclusions we can draw from Table III:

- Using a small number of multicast groups, the communication overheads at routers in our scheme is on par or better than existing schemes for single member departures.
- Our scheme incurs much lower communication overheads on routers than all existing schemes for batched updates to the group, upto $1-3$ order of magnitude better when the group membership changes is of order N (shown by the 1% set of results in the table).

These conclusions lead to two questions:

- How many departures have to be processed in bulk before we significantly outperform existing approaches? It is clearly not always feasible or desirable to wait till $O(N)$ members have left before the group is re-keyed.
- How many multicast addresses do we need to get decent performance, especially for large group sizes?

We addresses these two questions, in turn, in the next two sections.

A.3 Impact of batched updates and multiple multicast addresses

Batched updates are likely to be used in most realistic scenarios, especially with large group sizes. In Figure 10 we present results from an experiment in which we varied the number of members that simultaneously depart the multicast group. For each re-keying scheme, the figure plots the key normalized byte overhead for the entire group (added over all the tree routers) as different numbers of members simultaneously leave a 24,000 member secure multicast group. The effectiveness of the $O(\log N)$ bulk update provided by our scheme is clear in the plot from the shape of the *Spatial-x* curves in the plot. In comparison, all existing schemes incur $O(N)$ cost and perform worse than even *Spatial-1* when more than 128 members are processed in bulk. Using 16 addresses is enough to ensure that our scheme outperforms existing schemes for all batched updates to the group and we are a factor of two better than the previously best known scheme if we batch only 16 de-

Scheme	Single member leaves (varying group size)						1% of group leave (varying group size)					
	3000	6000	12000	24000	48000	96000	3000	6000	12000	24000	48000	96000
LKH-sequential	4.9	5.6	5.0	6.0	5.3	6.3	23.2	32.8	36.3	49.5	57.1	76.8
LKH-spatial	1.9	2.1	**1.6**	2.0	**1.5**	**1.8**	10.3	13.7	14.9	18.7	20.8	27.0
Spatial clustering	**1.6**	**1.6**	1.7	**1.7**	1.7	**1.8**	**3.2**	**3.3**	**3.4**	**3.4**	**3.5**	**3.6**

TABLE II

COMPARISON OF KEY NORMALIZED BYTE COUNT PER ROUTER ON A DIRECTED MULTICAST-CAPABLE NETWORK.

Scheme	Single member leaves (varying group size)						1% of group leave (varying group size)					
	3000	6000	12000	24000	48000	96000	3000	6000	12000	24000	48000	96000
LKH	22.2	23.9	26.8	28.5	31.0	32.4	394.2	792.7	1578.2	3092.7	6222.7	12609.6
Bool. min.	11.0	12.3	13.5	14.2	15.0	16.2	53.5	94.8	169.9	302.6	567.3	1077.0
Spatial-1	162.5	125.7	121.1	117.5	130.0	117.7	296.4	221.4	190.8	191.2	198.1	208.1
Spatial-16	8.7	11.0	13.9	16.5	19.3	20.6	10.3	12.8	16.2	19.1	22.4	24.1
Spatial-24	6.3	8.1	10.5	12.1	14.4	15.3	8.2	10.3	12.5	14.7	17.1	18.5
Spatial-directed	1.6	1.6	1.7	1.7	1.7	1.8	3.2	3.3	3.4	3.4	3.5	3.6

TABLE III

COMPARISON OF NORMALIZED BYTE LOAD PER ROUTER FOR THE DIFFERENT SCHEMES USING SCOPED MULTICAST.

partures for a 24,000 member group. Lastly, we note that the communication overheads at routers in our scheme improves by an order of magnitude if directed multicast is available.

Using multiple multicast addresses. It is clear from Table III that the performance of our scheme can be improved by using multiple multicast addresses. Unfortunately, it is difficult to optimally use a given set of addresses, and was not apparent how many addresses should ideally be used for a given group size. Fortunately, directed multicast provides bounds the performance of the scoped multicast implementations: the extra overhead of the scoped multicast implementations is precisely the "spill-over" traffic because of ineffective TTL-scoping.

In order to quantify the gains from using multiple addresses, we varied the number of addresses used and randomly assigned clusters to available addresses. We then noted, for each router in the multicast tree, the number of clusters for which the router carries any traffic. In the theoretically ideal case, for perfectly disjointed clusters, all routers should carry traffic from only a single cluster. (In comparison, in the very worst case, without TTL-scoping and if only a single address is available, all routers carry traffic from every cluster!). Our results are shown in Figure 11: for different number of multicast groups, we plot the cumulative distribution of the routers for different number of clusters whose traffic pass through the routers. We see that when a single multicast address is used, more than 70% of the routers carry traffic for at least 30 different clusters. The best case is observed for directed multicast, where 95% of the routers carry traffic for at most 2 clusters. Even with only 32 addresses, the cluster overlap falls significantly (80% of the routers carry traffic 10 or less clusters). We conclude that for groups with tens of thousands of members, few addresses $(16 - 32)$ approximate most of the benefits of directed multicast, and are sufficient to better all existing schemes.

A.4 Packet Processing Overheads

We now quantify the costs for our scheme in terms of the number of packets processed at the routers. In Table IV, we present a comparison of both the byte and packet loads of the various re-keying scheme. For this experiment, we assumed the 576 bytes maximum IP packet sizes (536 bytes payload, and 512 bit key sizes).

For a single member departure from a group of 24,000 initial members, the key server has to send 28 encrypted keys (4-ary key tree of height 8) for the key graphs scheme, which translates to a message of size about 1.8 Kbytes (4 IP fragments). Similar computations show that 2 IP fragments are processed per router for the boolean minimization scheme. In contrast, *Spatial-1* requires more significant processing at each router, because of the high spill-over traffic. As expected, the number of packets is significantly reduced by using a small number of multicast addresses. Like in the key-normalized cost metric, our scheme outperforms existing approaches with respect to the packet and byte count metrics. Depending on the level of batching and the number of addresses used, the gains from spatial clustering are quite dramatic, often using orders of magnitude less packets than previously known results.

B. Communication, Processing and Storage at Group Members

In this section, we quantify the costs of our scheme in terms of the overheads at the group members. As described in Section III-E, we have derived exact bounds for these metrics that hold independent of any network topology assumptions. The results in this section validate the analysis.

In Table V, we show the key storage requirements and the communication cost per member for a group of 24,000 members. In Table VI, we show the processing cost as the the number of cryptographic operations at members (including both symmetric and asymmetric operations) as the group membership changes. Asymmetric key cryptographic operations are computationally more expensive

Scheme	A single member leaves		1% of group leaves		10% of group leaves		25% of group leaves	
	KBytes	Packets	KBytes	Packets	KBytes	Packets	KBytes	Packets
LKH	1.8	4	197.9	370	905.2	1689	1273.1	2376
Boolean minimization	**1.0**	**2**	12.2	37	100.2	187	184.3	344
Spatial-1	7.5	118	12.2	191	12.8	200	12.3	193
Spatial-16	**1.0**	17	**1.2**	**19**	**1.5**	**23**	**1.5**	**23**
Spatial-directed	0.1	2	0.2	4	0.5	8	0.5	8

TABLE IV

COMPARISON FOR KEY-NORMALIZED BYTE AND PACKET LOADS PER ROUTER (GROUP SIZE: 24000 INITIAL MEMBERS).

Scheme	Number of Keys		Communication costs at members			
	Member	Key server	Single leave	1% leaves	10% leaves	25% leaves
LKH	9	32002	18.1	1970.9	8952.1	12671.0
Bool. min.	15	31	14.5	300.0	1427.3	2074.3
Spatial clustering	**3**	**6**	**1.1**	**1.4**	**2.3**	**2.3**

TABLE V

COMPARISON OF STORAGE AND KEY-NORMALIZED BYTE LOAD AT MEMBERS FOR THE DIFFERENT SCHEMES FOR A GROUP OF 24,000 INITIAL MEMBERS

Scheme	Processing at members				Processing at key server			
	Single leave	1% leaves	10% leaves	25% leaves	Single leave	1% leaves	10% leaves	25% leaves
LKH	1.7	5.5	6.4	5.7	28.0	3095.3	14310.5	19892.7
Bool. min.	**1.0**	**1.0**	**1.0**	**1.0**	15.0	302.9	1583.5	2680.2
Spatial clustering	1.2	1.5	2.4	2.4	**1.3**	**3.5**	**4.0**	**4.0**

TABLE VI

COMPARISON OF PROCESSING COSTS AT GROUP MEMBERS FOR THE DIFFERENT SCHEMES FOR A GROUP OF 24,000 INITIAL MEMBERS

than symmetric key cryptographic operations and are required for mutual authentication and pair-wise key establishment between a cluster-leader and a new cluster member. Our analysis shows that the number of asymmetric key operations required are insignificantly small and we validated this in our experiments. In a group of 24,000 initial members for a single membership change, the average number of asymmetric key cryptographic operations at a member was 0.0004. For batched updates, the number of such operations per member was 0.004, 0.02 and 0.07 for 16, 64 and 256 simultaneous changes to the group respectively. In contrast the total number of symmetric key operations required varied between 1.2 and 1.4 as the number of simultaneous changes in the group varied from 1 to 256. While the number of keys at each member is low for all the schemes, the number of keys stored at the server is significantly lower for both boolean minimization and spatial clustering scheme. Our scheme leads to lower processing at the key server for single leaves. For batch updates, the processing at the key server and the communication costs is bounded by $O(\log N)$ for our scheme, which is a substantial improvement over the $O(N)$ costs for both LKH and Boolean Minimization schemes.

B.1 Experiments with the Clustering Protocol

We have also implemented a packet-level simulator of the spatial clustering scheme on the `ns` (version 2) simulator to study the time dependent overheads of the protocol. We present only a synopsis of our main results, because of space constraints. Due to high processing and memory demands of `ns`, we were limited to simulations on topologies of upto 1000 routers. In Table VII, we present results from an experiment where a set of 250 members joined a single multicast group, all between simulation time 0 and 1 second. The member discovery and clustering protocols arrange the members into size bounded clusters; in all our experiments, the size bounds were met by all the clusters. The time required (in seconds) for clustering is shown in the last column of Table VII. For these experiments, we used four heartbeats per second which resulted in about 40 bytes of traffic per second, scoped within the cluster. We used only periodic heartbeats, and not the adaptive technique described in Section IV-C, which can further reduce the stabilization time.

Even though we only simulated on topologies on the order of 1000 nodes, we are confident that the stabilization times we present are representative of large topologies. This is because, in our experiments, we observed that in the vast majority of cases, independent of group size, a single join or leave to the group affects less than two clusters. As multiple simultaneous changes occur, the clustering occurs in parallel over the entire topology; the total time taken for stabilization depends primarily on the cluster size and depth, and not on the size of the group.

VII. SUMMARY AND CONCLUSIONS

In this paper, we have presented a new algorithm for efficiently implementing secure group communications over

Cluster size bound	Total Num. of Clusters	Avg. Cl. Size	Avg (Max) Stab. Time (s)
4 – 7	45	5.6	1.45 (2.63)
10 – 19	17	14.7	1.85 (3.87)
20 – 39	9	27.8	2.22 (4.60)

TABLE VII
CLUSTER CHARACTERISTICS

	250 initial group members No. of Simultaneous Changes				
	1	4	16	64	224
Join	1.0	1.8	2.4	3.6	4.3
Leave	0.5	1.6	3.3	4.0	5.1

TABLE VIII
CLUSTER STABILIZATION TIME (IN SECONDS)

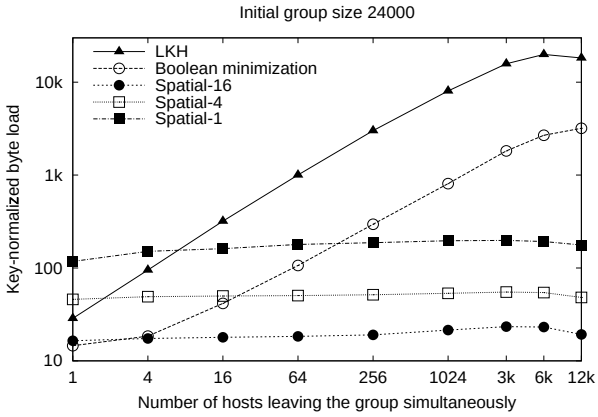


Fig. 10. Varying the number of simultaneously leaving members (Scoped multicast)

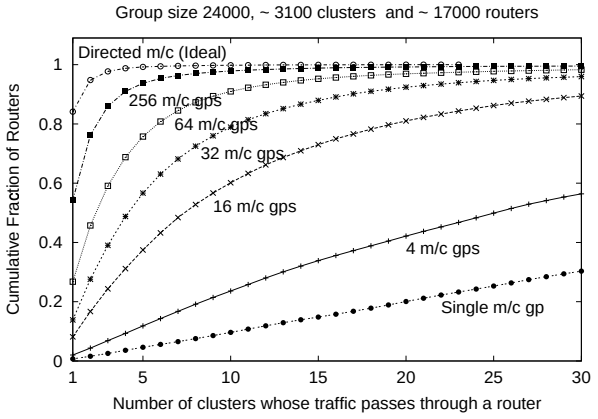


Fig. 11. Cumulative distribution of the routers that handle cluster traffic for different number of clusters

IP multicast. For all metrics, our scheme provides the best analytic bound, frequently improving on previously known results. For single group changes, our improvements reduce previously known logarithmic bounds to constants, are probably only of theoretical interest. However, for bulk simultaneous group changes, our theoretical results for network load and processing cost at the key server are significant, since they reduce previously known linear bounds to logarithmic bounds.

Unlike existing approaches, our re-keying algorithm was designed to utilize the parallelism inherent in the multicast tree topology. Therefore, we expect our algorithms to perform extremely efficiently in practice. As shown by our extensive simulations on a large realistic topology, for large groups, the number of messages and encryptions required by our scheme is often orders of magnitude lower than existing approaches, especially when we consider simultaneous changes to the group. Our experiments also show that directed multicast is an useful primitive for implementing many secure multicast schemes, including ours.

REFERENCES

[1] D. Balenson, D. McGrew, and A. Sherman. Key management for large dynamic groups: One-way function trees and amortized initialization. draft-balenson-groupkeymgmt-oft-00.txt, IETF, February 1999.

[2] A. Ballardie. Scalable Multicast Key Distribution. Network Working Group, RFC 1949., May 1996.

[3] Tony Ballardie, Paul Francis, and Jon Crowcroft. Core based trees (CBT). *Proceedings of SIGCOMM*, September 1993.

[4] B. Briscoe. MARKS : Zero side effect multicast key management using arbitrarily revealed key sequences. In *1st International Workshop on Networked Group Communication, Pisa, Italy, November 1999.*, Pisa, Italy, November 1999.

[5] B. Cain, T. Speakman, and D. Towsley. Generic Router Assist (GRA) building block motivation and architecture. Internet Draft, Internet Engineering Task Force, March 2000. Work in progress.

[6] Ran Canetti, Juan Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas. Multicast security: A taxonomy and efficient constructions. In *Proceedings of INFOCOM*, New York, March 1999.

[7] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, and D. Saha. Key management for secure internet multicast using boolean function minimization techniques. In *Proceddings of Infocom*, New York, March 1999.

[8] M.G. Reed D.M. Goldschlag and P.F. Syverson. Onion routing for anonymous and private internet connections. *Communications of the ACM*, 42(2), February 1999.

[9] E. Harder D.M. Wallner and R.C. Agee. Key management for multicast: Issues and architectures. RFC 2627, IETF, June 1999.

[10] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, and P. Sharma. Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification. *RFC 2362*, 1998.

[11] R. Govindan and H. Tangmunarunkit. Heuristics for Internet Map Discovery. In *Proceedings of Infocom*, March 2000.

[12] H. Harney and C. Muckenhirn. Group Key Management Protocol (GKMP) Architecture. Request for Comments (Experimental) 2094, Internet Engineering Task Force, July 1997.

[13] B.N. Levine and J.J. Garcia-Luna-Aceves. Improving Internet Multicast with Routing Labels. In *Proc. IEEE International Conference on Network Protocols*, pages 241–50, October 1997.

[14] X. Li, R. Yang, M. Gouda, and S. Lam. Batch updates for key trees. Technical report, University of Texas, Austin, September 2000.

[15] S. Mittra. Iolus: A framework for scalable secure multicasting. *Proceedings of SIGCOMM*, October 1997.

[16] M. Naor, D. Naor, and L. Lotspiech. Revocation and tracing schemes for stateless receivers. In *Proceedings of Crypto 2001*, 2001.

[17] A. Reddy, R. Govindan, and D. Estrin. Fault isolation in multicast trees. *Proceedings of SIGCOMM*, August 2000.

[18] M.K. Reiter and A.D. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1), April 1998.

[19] Bruce Schneier. *Applied Cryptography*. John Wiley and Sons, 1996.

[20] C. Shields and B.N. Levine. A protocol for anonymous communication over the internet. *Proceedings of the 7th ACM Conference on Computer and Communications Security*, November 2000.
[21] J. Snoeyink, S. Suri, and G. Varghese. A lower bound for multicast key distribution. *IEEE Infocom*, April 2001.
[22] T. Speakman et al. PGM reliable transport protocol. Internet Draft, Internet Engineering Task Force, April 2000. Work in progress.
[23] D. Waitzman, C. Partridge, and S. Deering. Distance vector multicast routing protocol. *RFC 1075*, 1998.
[24] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner. The Versakey Framework: Versatile Group Key Management. *IEEE Journal on Selected Areas in Communications, Special Issue on Middleware*, 17(9), August 1999.
[25] C.K. Wong, M. Gouda, and S. Lam. Secure group communications using key graphs. *Proceedings of SIGCOMM*, September 1998.

Appendix

I. Key Distribution Protocol

We assume a clustering protocol exists that creates and maintains clusters, as specified in Section III. The clusters do not share any members and have size between k and $2k-1$, for some fixed k. A hierarchy of members is created as shown in Figure 3 as follows:

All members are part of the lowest layer, L_0. Each layer, L_i, runs an instance of the clustering protocol to create a set of clusters, the leaders of which join the next higher layer, L_{i+1}.

Each layer has a secret layer key known to only the members of the layer. Similarly, each cluster in each layer has a secret cluster key, known to only all the cluster members.

A. Notation

- Members and Member Sets
  - $Cluster(u, j)$: Cluster of layer L_j, to which member u belongs.
  - $Ldr(u, j)$: Leader of the cluster in layer L_j to which member u belongs.
  - S : The key server for all layer keys.
  - ACS: The authentication and access control server.
- Keys and Messages
  - $\chi_G(t)$: The secret key of G at time t, where where G is a set of members. If G is a cluster, then this is the cluster key, if G is a layer, then this is the layer key. If G is a pair of members, then this is a key shared only by these two members.
  - $\{m\}_e$: Message m is encrypted by the key e.
  - $\langle Unicast :: u \rightarrow v : x\rangle$: u sends a unicast message x to v.
  - $\langle Multicast :: u \rightarrow G : x\rangle$: u multicasts message x to set of members G, where G is either a cluster or a layer.

B. Distributed Re-keying Operation

When a member u joins or leaves layer L_j, the following operations are performed distributedly by the key distribution protocol:

1. Cluster re-key: $Ldr(u, j)$ generates a new cluster key $\chi_{Cluster(u,j)}(t+1)$ and unicasts it to each current member of the cluster $Cluster(u, j)$ encrypted separately by the pair-wise key of the leader with each member.

$$\forall v \in Cluster(u, j)$$
$$\langle Unicast :: Ldr(u, j) \rightarrow v : \{\chi_{Cluster(u,j)}(t+1)\}_{\chi\{Ldr(u,j),v\}}\rangle \quad (1)$$

Obviously, in case u is leaving this cluster, this message is not sent to u. The total communication overhead of this cluster re-key is $O(k)$ at the cluster leader and $O(1)$ at each other member (due to $O(k)$ pair-wise communication between the cluster-leader and the cluster members).

2. Layer re-key: The key server,S, generates a new layer key for layer L_j, and multicasts it to all members of layer L_{j+1} — these are the cluster-leaders of the clusters of layer L_j. Each cluster leader of layer L_j then performs a cluster multicast to all the members of its cluster in layer L_j. The multicast messages are encrypted by the appropriate keys.

$$\langle Multicast :: S \rightarrow L_{j+1} : \{\chi_{L_j}(t+1)\}\chi_{L_{j+1}}(t)\rangle \quad (2)$$

$$\forall v \in L_{j+1}$$
$$\langle Multicast :: v \rightarrow Cluster(v, j) : \{\chi_{L_j}(t+1)\}_{\chi_{Cluster(v,j)}(t+1)}\rangle \quad (3)$$

Each member in L_{j+1} receives a single copy of the layer multicast message from S to layer L_{j+1}. Since the clusters of any layer are disjoint, each member in L_j receives only a single cluster multicast message sent by the leaders of the clusters in layer L_j. Thus, the combined communication cost of these multicasts is $O(1)$ per member.

The total communication cost for distributed re-key operation in each layer is, therefore, $O(k)$ per link.

C. Re-keying algorithm for member joins and leaves

1. Join: When a new member, u, joins the secure multicast group, it is first authenticated by the ACS. Subsequently when u joins a cluster, it needs to establish a pair-wise key with its leader. We describe this interaction using a variant of the pure Diffie-Hellman protocol as described in [19]. The pair-wise key can also be established by using a similar variant of other key exchange protocols, e.g. Elliptic Curve Cryptography. We assume that a prime number, p, a generator, g and the public key of the ACS are publicly known across the system. We use modulo p arithmetic in this description. For simplicity of notation we use g^α to represent $g^\alpha (\text{mod } p)$.

All interactions in this sequence are unicast.

(a) Interaction of u with ACS.

$$u \rightarrow \text{ACS} \quad : \quad \{u, g^\alpha\} \quad (4)$$
$$\text{ACS} \rightarrow u \quad : \quad \{u, g^\alpha, \text{Cred}_u\} \quad (5)$$

where, α is the random exponent chosen by u, and $\text{Cred}_u = \{u, g^\alpha\}_{\text{ACS}}$.

(b) Interaction of u with v, the leader of the cluster to which u joins. Let β be the random exponent chosen by v when it had joined the secure group.

$$u \rightarrow v : \{g^\alpha, \text{Cred}_u\} \quad (6)$$
$$v \rightarrow u : \{g^\beta, \text{Cred}_v\} \quad (7)$$

u and v then can compute a shared key $K = (g^\alpha)^\beta = (g^\beta)^\alpha$ independently.
The Distributed Re-keying Operation is also invoked, i.e. v performs a cluster re-key in its cluster and the key server performs a layer re-key for layer L_0.
2. Leave: When a member, u, leaves a secure group, it leaves from all the layers to which it was joined. This departure may be voluntary or can be due to an explicit removal of u from the group by the ACS. If u was part of layers $L_0 \ldots L_j$ and no other layer, first a new leader is set up in the single cluster in layer $L_i, i \in [0, j-1]$ of which u was the leader. This requires mutual authentication and pair-wise key establishment as described in Steps 6 and 7 above. Finally, the Distributed Re-key Operation is invoked for each of the layers to which u was previously joined. For the layer re-key operation of a layer L_{j+1}, the new layer key is first distributed to members of layer L_j. The key server tags this re-key message with the identity of the member, u, whose group membership is being revoked. In both cases, the clustering protocol also appropriately re-clusters the affected layers, if needed.

D. Analysis

We analyze three different metrics — key storage requirements at each member, the processing costs due to encryptions and decryptions at each member and the communication overheads per link of the multicast delivery tree for each re-key. In our analysis we present the processing and communication costs for member departures only. The overheads incurred for group re-keying due to member departures are higher than the overheads for group re-keying for new members joining the group and therefore serve to upper-bound for the costs of member join operations.

Assume that there are N members currently joined to the secure multicast group. Let L_R be the highest layer in the hierarchy (it contains a single member). The following properties hold :
• $R \leq \log_k N$. For all j in $[0, R]$, layer L_j has not more than $N/(k^j)$ members.
• If a member u, is present in L_j, it is present in all lower layers $L_0 \ldots L_{j-1}$. If a member u is not present in layer L_j, it is not present in any of the higher layers, $L_{j+1} \ldots L_R$.

Communication Cost at a member. Most members (i.e. at least $N(1-\frac{1}{k})$ of them) are joined only to the lowest layer, L_0. Hence, a majority of member joins and leaves affect only layer L_0 and the communication cost incurred for the necessary re-keying is $O(k)$, a constant (as shown before). In the worst case, the single member at the highest layer L_R leaves, all the layer keys would need to be changed. Each layer key change requires $O(k)$ communication cost per link, which makes the *worst case* communication cost of $O(k \log_k N)$ for leaves (and analogously for joins).

Under the assumption that each member of the group is equally likely to leave (and join), we now show that the *amortized cost* for joins and leaves is constant. We consider the case of a member leaving the group. Consider the case when a member u, joined only to layers $L_j \ldots L_0$, leaves the secure multicast group. Each of these $j+1$ layer keys need to be changed. The maximum communication cost per member for each layer key change is $O(k)$. Additionally, in each layer, it can be shown that at most two clusters would need to be re-clustered (if size of the affected cluster falls below the size lower bound, k). Due to the clustering protocol, this would involve communication cost of at most $O(k)$ per member. Hence, the total communication cost per member due to re-keying of all the layers, for member u leaving the group is $O(k(j+c))$, where c is some constant. For any layer, the number of members in layer L_j is bounded by N/k^j. Thus, the amortized cost of for a member leaving the multicast group is given by :

$$\begin{aligned} \frac{1}{N}\sum_{j=0}^{R} |L_j| k(j+c) &\leq \frac{1}{N}\sum_{j=0}^{\log N} \frac{N}{k^j} k(j+c) \\ &= c\frac{k}{k-1} + \frac{k^2}{(k-1)^2} + O(\frac{\log_k N}{N}) \end{aligned}$$

However, k and c are constants and $\frac{\log N}{N} \rightarrow 0$ asymptotically with increasing N. Hence, the amortized cost of a member leaving (and also for joining) is constant.

Communication Cost at a router (link). The communication cost at routers (links) depend on the underlying physical topology of the multicast tree. It is easy to see that if the physical cluster topologies of the clusters of a layer are non-overlapping, then using the same argument as above, the amortized cost at any router (link) of a member leaving (or joining) the group is constant. In Section VI we have studied the topological properties for clusters on a real Internet map. Our study concluded that the physical cluster topologies have very low overlap.

Storage. Let us assume that the clusters are of size k each [6]. The total number of keys at a member that belongs to layer L_j and no layer above that, are as follows : $j+1$ layer keys, 1 pair-wise key with the leader of its cluster in layer L_j, and $k-1$ pair-wise keys with the other members in each of the clusters in layers $L_0, \ldots, L_{j-1}$, leading to a total of $jk+2$ keys. Then number of such members in L_j is $\frac{N}{k^j}(1-\frac{1}{k})$.
Hence, the average number of keys per member is: $\frac{1}{N}\sum_{j=0}^{\log N} \frac{N}{k^j}(1-\frac{1}{k})(kj+2) = 2 + \frac{k}{k-1} + O(\frac{\log N}{N})$. Asymptotically, the average number of keys at a member is, therefore, $2 + \frac{k}{k-1} \leq 4$.

Processing cost at a member. We first consider the processing cost at members due to symmetric key operations. When a member, that belongs to only L_0 leaves, the number of symmetric key-based decryptions can be broken down as follows: 1 at each of the $k-1$ members in the affected cluster of L_0 (Step 1, cluster re-key in Distributed Re-keying Operation), 1 at each N/k member in layer L_1 (Step 2, layer re-key) and 1 at each of $N(k-1)/k$ members that occurs in layer L_0 only (Step 3, layer re-key). Therefore, the average cost per member is $\frac{N+k-1}{N}$.

[6] Although cluster sizes can vary between k and $2k-1$, this assumption does not change the order of the results.

Similarly, when a member that belongs to layers $L_j \ldots L_0$ leaves, the total number of symmetric key-based decryptions can be broken down as the cluster and layer re-key cost for each layer $L_i, i \in [0, j]$ as follows: $k-1$ decryptions for a cluster in layer L_i (Step 1, cluster re-key), $\frac{N}{k^{i+1}}$ decryptions of the layer L_i key by members in layer L_{i+1} (Step 2, layer re-key) and $(1-\frac{1}{k})\frac{N}{k^i}$ decryptions for subsequent retrieval by layer L_i members of the same layer key (Step 3, layer re-key). Therefore, the cluster and layer re-key cost for re-key of layer L_i is $(k-1)+\frac{N}{k^i}$. Then, the average symmetric key-based decryption cost per member for a departure from layer L_j is obtained as: $\phi_j = \frac{1}{N}\sum_{i=0}^{j}(k-1)+\frac{N}{k^i} = \frac{(k-1)(j+1)}{N}+\sum_{i=0}^{j}\frac{1}{k^i}$. Aggregating over departures of members from any arbitrary layer, the amortized decryption cost per member is: $\frac{1}{N}\sum_{j=0}^{\log N}\frac{N}{k^j}(1-\frac{1}{k})\phi_j = \frac{k^2}{k^2-1}+O(\frac{\log N}{N})$, which is ≤ 2 for asymptotically increasing N.

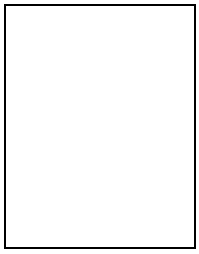
The number of symmetric key-based encryptions for a departure of a member that belongs to layers $L_j \ldots L_0$ can be computed in a similar manner. The cluster and layer re-key cost for each layer $L_i, i \in [0, j]$ is: $k-1$ for a cluster in layer L_i (Step 1, cluster re-key) and $\frac{N}{k^{i+1}}$ encryptions of the layer L_i key by members in layer L_{i+1} (Step 2, layer re-key). Then, the average symmetric key-based encryption cost per member for a departure from layer L_j is obtained as: $\Phi_j = \frac{1}{N}\sum_{i=0}^{j}(k-1)+\frac{N}{k^{i+1}}$. Aggregating over departures of members from any arbitrary layer, the amortized encryption cost per member is $\frac{1}{N}\sum_{j=0}^{\log N}\frac{N}{k^j}(1-\frac{1}{k})\Phi_j = \frac{k}{k^2-1}+O(\frac{\log N}{N})$, which is < 1 for asymptotically increasing N and $k > 1$.

We now consider the processing cost due to asymmetric key operations at the members. These operations are due to mutual authentication and establishment of the pair-wise keys between a (new) cluster leader and the cluster members. In the most typical case (i.e. a member that belongs to layer L_0 leaves the group) no asymmetric key operations are required, since there are no new authentication and pair-wise key establishment required with the remaining members of the affected cluster. When a member that belongs to layers $L_j \ldots L_0$ leaves, the total number of asymmetric key operations at layer $L_i, i \in [0, j-1]$ is bounded by $4(k-1)$ (Steps 6 and 7 of Authentication). These are the pair-wise interactions between a new leader and the members of the corresponding cluster in each of these layers. The total cost for a departure from layer L_j is bounded by $4(k-1)j$. Hence, the average number of asymmetric key-based operations at a member for such a departure is $\frac{4(k-1)j)}{N}$. Therefore, the amortized number of asymmetric key operations at the members is $\frac{1}{N}\sum_{j=0}^{\log N}\frac{N}{k^j}(1-\frac{1}{k})\frac{4(k-1)j}{N} << 1$ for asymptotically increasing N.

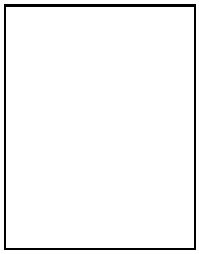
Processing cost at key server. The key server encrypts new layer keys prior to layer multicast. The amortized number of encryptions for a single departure is: $\frac{1}{N}\sum_{j=0}^{\log N}\frac{N}{k^j}(1-\frac{1}{k})(j+1) = 1+\frac{1}{k-1}+O(\frac{\log N}{N})$, which is asymptotically bounded by 2.

Bulk simultaneous departures. For a group of size N, assume xN of the groups leave, where $0 < x < 1$. From layer L_j, the number of leaving members is $x\frac{N}{k^j}(1-\frac{1}{k})$. The total number of cluster key decryptions for the members in layer L_j (Step 1, cluster re-key) is upper-bounded by the total number of members in the layer, which is $\frac{N}{k^j}(1-\frac{1}{k})$. The total number of layer key decryptions of layer L_j includes one decryption for each of the $\frac{N}{k^{j+1}}$ members of layer L_{j+1} (Step 2, layer re-key) and one decryption for each of the $\frac{N}{k^j}.(1-\frac{1}{k})$ members that occur in layer L_j and no other higher layer (Step 3, layer re-key). Therefore, the decryption overheads due to layer L_j is $\leq \frac{1}{N}\frac{N}{k^j}(2-\frac{1}{k})$. Aggregating over all layers, the decryption cost per member due to the layer and cluster key updates at this layer is bounded by $\sum_{j=0}^{\log N}\frac{1}{k^j}(2-\frac{1}{k}) = \frac{2k-1}{k-1}+O(\frac{\log N}{N})$, which is asymptotically $O(1)$. Similar analysis yields similar bounds for symmetric key encryptions and asymmetric key operations at members.

The processing cost at the key server is $\log N$ (the maximum number of layers). Similarly, the communication overhead at each member (and also for routers and links under the assumption that physical cluster topologies are non-overlapping) is $O(\log N)$.



Suman Banerjee is a Ph. D. candidate at the Department of Computer Science of University of Maryland at College Park, USA. He received the B.Tech. degree in Computer Science and Engineering from the Indian Institute of Technology, Kanpur, India in 1996 and the M.S. degree in Computer Science from the University of Maryland at College Park, USA in 1999. His research interests are in protocols and services for networking and distributed systems.



Bobby Bhattacharjee received his Ph. D. in Computer Science from Georgia Tech. in 1999. Since 1999, he has been an assistant professor in the Computer Science department at the University of Maryland, College Park. His research interests are in network protocols, network security, and distributed systems. He is a member of the ACM.