Support for Mobility and Fault Tolerance in Mykil

Jyh-How Huang and Shivakant Mishra
Department of Computer Science, University of Colorado
Campus Box 0430, Boulder, CO 80309-0430, USA.
Email: {huangjh|mishras}@cs.colorado.edu

University of Colorado, Department of Computer Science Technical Report CU- CS-962-03

Abstract— **This paper describes the support provided for mobility and fault tolerance in Mykil, which a key distribution protocol for large, secure group multicast. Mykil is based on a combination of group-based hierarchy and key-based hierarchy systems for group key management. Important advantages of Mykil include a fast and efficient rekeying operation for large group sizes, continuous availability of the key management service in a disconnected network environment, an ability to map group structure to the underlying network infrastructure, fault tolerance, and support for member mobility and smaller hand-held devices. Mobility support in Mykil allows mobile group members to access a multicast service without any need for going through an extensive registration process. Fault tolerance support allows group members to access a multicast service despite communication failures, network partitions, and node failures in the network. A prototype of Mykil has been implemented. The paper describes this implementation and reports on the performance measured from this implementation.**

I. Introduction

Multicasting is a fundamental communication paradigm that allows an application to efficiently disseminate data to a group of recipients. With the ever-increasing popularity of the Internet, *secure group multicast* is increasingly being used to construct applications that require one-to-many and many-to-many communication mechanisms. Examples of such applications include pay-per-view programs, video-on-demand services, frequent stock quote updates, video conferencing, discussion forums, and advertising.

Support for secure group multicast is typically built on top of IP multicast. A *secure multicast group* is a multicast group in which members register and authenticate themselves with a designated registration authority, receive a set of *cryptographic key(s)*, and use these keys to encrypt the multicast data that they send and decrypt the multicast data that they receive. Figure 1 illustrates the steps followed when a new member joins a multicast group. After registration, users contact a key management server using the key(s) and other materials obtained from the registration server. A key management server manages a set of cryptographic keys used for various purposes in a secure multicast group, e.g. one or more *group key(s)* that is (are) used to encrypt and decrypt multicast data. It stores these keys, updates them when certain events occur, and distributes them to the group members using a key distribution protocol. The process of updating the cryptographic keys, and distributing them to the group members is called a *rekeying* operation. Rekeying is required in secure multicast to ensure that only the *current* group members can send encrypted multicast data, and decrypt the received multicast data.

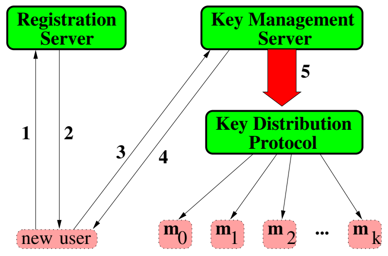

Fig. 1. Secure Multicast Components.

In this paper, we focus on large multicast groups with frequent membership changes, i.e., groups consisting of a significantly large number of members (100,000 members or more) with members joining or leaving quite frequently. There are a large number of multicast applications that exhibit these characteristics. For example, a popular pay-per-view program can have a very large number of subscribers, or a popular discussion forum can have a very large number of participants at certain times. When a group is large, the cost of key management can become prohibitively expensive. This is because a rekeying operation requires distributing various keys, including group key(s), to all group members. If this is done naively, it may require O(n) messages, where n is the number of members in the group. Furthermore, if a rekeying operation is performed after every membership change, and if the membership changes are frequent, key management will require a large number of message exchanges per unit time.

We have designed and implemented a key management protocol called Mykil (**M**ulti-Hierarch**y** Based **k**ey D**i**stribution Protoco**l**) for managing cryptographic keys in large multicast groups that exhibit frequent membership changes. Mykil cleverly combines two different types of hierarchy schemes—group-based hierarchy and key-based hierarchy, to provide an efficient and scalable solution for key management in large multicast groups. Mykil borrows several interesting ideas from the earlier work done in the area of key management for large group multicast, and provides a solution that is better than the previous solutions.

In addition to addressing the scalability problem of key management in large multicast groups, Mykil provides support for mobile group members and fault tolerance. Mykil ensures that group members that move from one location to another in the network can continue to avail the multicast service and the key management functionality without having to go through extensive registration process. Furthermore, Mykil ensures that the key management functionality remains available to group members, despite communication partitions or node failures.

Overall, Mykil makes four important contributions. First, it provides a very efficient and fast rekeying operation by ensuring that key updates take place at only a small number of group members during a member join or leave event. The cost of rekeying operation is further reduced by batching member join and leave events. Second, it is designed to support group members that access a multicast service using small devices such as PDAs or cell phones that have limited resources. This is done by minimizing the memory, bandwidth, and CPU requirements for key management at group members. Third, Mykil is designed to support both static and mobile group members. Finally, the key management functionality of Mykil is robust and remains available to all group members even when the underlying communication network partitions.

II. Related Work

Assume that the cryptographic keys of a large multicast group have been changed several times since the group was formed, and the sequence of these successive group keys is S. Four important cryptographic properties have been identified for managing a group key [18]:

1) **Key freshness:** The group key must be *new* at any time.
2) **Group key secrecy:** It is computationally infeasible for an adversary (non-member) to discover any of the group keys in S.
3) **(Weak) Backward secrecy:** A current or a former group member that knows a subset of keys in S cannot deduce any of the preceding keys in S.
4) **(Weak) Forward secrecy:** A current or former group member u cannot deduce any of the new group keys after u leaves (or left) the group.

Several novel and scalable key management schemes for large group multicast have been proposed, and some of them have been implemented. A detailed survey of these schemes is provided in [11], [5]. These schemes can be categorized under two different classes: (1) group-based hierarchy schemes, and (2) key-based hierarchy schemes.

Group-based hierarchy schemes address the scalability issue in key management systems by organizing a multicast group into a hierarchy of subgroups. A group member belongs to exactly one of these subgroup. The basic idea is to distribute the functionality of the key management service among the subgroups, and thereby achieve decentralization and scalability. Examples of key management systems that are based on group-based hierarchy include [7], [17], [12], [13].

Key-based hierarchy schemes on the other hand address the problem of scalability by organizing a tree-structured hierarchy of cryptographic keys. This hierarchy scheme is also known as logical key hierarchy (LKH). The root of this tree corresponds to the group key, and the rest of the nodes of the tree correspond to cryptographic keys (called auxiliary keys) that are selectively used for distributing various keys to the users. Each group member is associated with a different leaf in this tree and needs to know all the keys in the path from that leaf node to the root. Key management systems that are based on this basic idea of organizing keys in a tree-structured hierarchy include [21], [20], [3], [15], [4], [10], [23]. This tree-structured organization of keys is maintained by the key management server.

A rekeying operation results in changing all keys along a path from the root to one of the leaf node where join/leave event occurs. New keys are distributed to appropriate members by cleverly encrypting them using the previous auxiliary keys (See [11] for details). Scalability is achieved in key-based hierarchy protocols by reducing the number of message exchanges during a rekeying operation. For example, assuming that the tree is relatively balanced, only O($log\ n$) keys need to be changed, where n is the total number of group members. Furthermore, by cleverly encrypting the new keys using the old auxiliary keys, all new keys can be communicated to respective group members via a single multicast message.

Both group-based and key-based hierarchy protocols have their advantages and disadvantages. Both types of protocols provide significant scalability over a naive key distribution protocol mentioned in Section I. Because of decentralization of key management functionality, a protocol based on group-based hierarchy can tolerate network partitions. A protocol based on key-based hierarchy on the other hand cannot tolerate network partitions. Another advantage of group-based hierarchy is that the actual organization of different areas can be mapped quite well to the underlying network infrastructure. For example, all members located with in a subnet or an organization may belong to one area. Storage requirements for a key management server in a group-based hierarchy is low (one subgroup key and m pairwise secret keys, where m is the number of members in a subgroup), while it is 2^{n-1} in key-based hierarchy. So, the storage requirements for a key key management server in a key-based hierarchy protocol can become prohibitively high. Finally, key distribution during a rekeying operation in a group-based hierarchy protocol relies on a subgroup controller sending separate messages to area members. This can result in a subgroup controller becoming a performance and scalability bottleneck.

III. Mykil: Protocol Details

The main motivation of Mykil is to combine the two hierarchy schemes in such a way that the good features of the two schemes are preserved and the limitations of the two schemes are eliminated. Although such a combination has been hinted in [12] and [11], no protocol based on this combination has been developed so far. Mykil is the first protocol that exploits this idea and demonstrates its usefulness in terms of scalability and performance. Furthermore, Mykil is specifically designed to provide support for robustness and for members that are mobile and/or access a multicast service via small hand-held devices. Mykil also optimizes the rekeying operation by batching, and can save up to 40—60% key update multicast messages. Finally, a tree structure for organizing different areas and members with in each area is vulnerable to node/link failures. Mykil addresses this by including a fault-tolerance component that allows group members to move to other areas, and area controllers to change their parent nodes.

In particular, Mykil is based on Iolus[12] and LKH[21]. It uses the idea of group-based hierarchy of Iolus to divide a multicast group into several smaller subgroups called *areas* with a designated area controller (AC) for each area. There is a separate *area key* for each area. Different areas are linked with one another to form a tree structure, with ACs providing the links—an AC of an area A is also a member of another area B (area B is A's parent in the tree-structure organization). A group member belongs to exactly one area. Like LKH, Mykil builds a tree-structured hierarchy of cryptographic keys called *auxiliary-key tree* with in each area to facilitate key distribution to the area members. The area controller of an area maintains the auxiliary-key tree of that areas, and each member of this area is associated with a different leaf of this auxiliary-key tree. Multicast data propagation in Mykil is identical to the multicast data propagation in Iolus. A group member (sender) first generates a random key K_r and encrypts the data to be multicast using K_r. It also encrypts K_r using its area key. It then multicasts the encrypted data and encrypted random key in a single multicast message with its area. To forward multicast data (that has been encrypted using a random key) from area A to another area B, the AC of A first decrypts the random key K_r and encrypts it using the B's area key. Recall that an AC is a member of two areas, and so it knows the area keys of two different areas. An example of the organization of group members along with propagation of data multicast by group member m_s is shown in Figure 2.

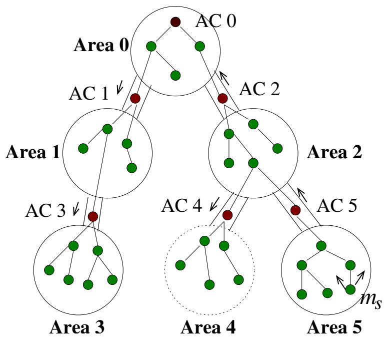

Fig. 2. Organization of group members in Mykil.

A. Group and Area Creation

A multicast group $\mathcal{G}$ is initialized by creating a root area with a designated area controller for this area. This root area forms the root in the tree-structured organization of the all areas comprising $\mathcal{G}$. The area controller of this root area is also the group controller of $\mathcal{G}$. In terms of functionalities, a group controller is no different from any other area controller of the group. Responsibilities of an area controller include: (1) managing cryptographic keys of its area; (2) forwarding multicast data as shown in Figure 2; (3) managing member mobility and failures; (4) maintaining the auxiliary key tree of its area; and (5) managing member join and leave events.

Creation of a new area in $\mathcal{G}$ is initiated by a designated area controller A_c. Before creating a new area, A_c must obtain an authorization information database $\mathcal{AI}$ needed to determine various keys needed for managing member mobility, and node or communication failures. While the exact mechanism of how this is done is out side the scope of Mykil, it is important to note that the security of a multicast group critically depends on this mechanism. A_c chooses another area to be the parent of its area. This choice can be based on network proximity, administrative policy, or any other appropriate criteria. A_c then joins the chosen parent area (as a regular group member) by contacting the area controller of the parent area as described in the next subsection.

B. Member Join

The protocol for joining a multicast group in Mykil ensures that only authorized entities are able to join the group. In addition, this protocol is designed to facilitate support for member mobility and fault tolerance. In particular, the join protocol is comprised of seven steps as shown in Figure 3. Each of these steps are carefully designed to prevent security attacks such as node impersonation, replay attacks, or man-in-the middle attacks. Both public-key cryptography and symmetric-key cryptography have been used. It is assumed that all entities in the protocol, clients that want to join a group, registration server, and area controllers have their own public/private key pairs. Further, it is assumed that the registration server and all area controllers know one another's public keys. Finally, it is assumed that the public key of the registration server is well known. A challenge-response based mechanism [19] has been used for authentication purposes, and nonces and time stamps have been used to prevent replay and man-in-the-middle attacks.

In the first step, a client K that wants to join a multicast group sends a join message to the registration server. This message is encrypted using the public key of the registration server, and contains four pieces of information: authorization information; client K's public key; $Nonce_{CW}$; and a MAC (message authentication code) computed over the first three pieces of information. Authorization information is used by the registration server to determine if K is eligible to join the group and the length of K's membership duration. For example, this can contain credit card information and the time period the client wants to stay as a member. By including its public key, K ensures that the registration server learns its public without any need for public key infrastructure. $Nonce_{CW}$ is a random number generated by K that is used as a challenge to the registration server to authenticate itself. Finally, MAC is used to ensure that the registration server can verify the integrity of this message.

On receiving this message, the registration server first decrypts the message and verifies its integrity. If verified, it checks the authorization information of K. If K is determined eligible to join the multicast group, the registration server executes the second step by sending a message to K containing the following information: $Nonce_{CW} + 1$; $Nonce_{WC}$; and a MAC computed over the first two pieces of information. This message is encrypted using the public key of K that the registration server received in the first step. $Nonce_{CW} + 1$ is included to authenticate the registration server's identity as the one who knows the private key corresponding to the well-known public key that K used to encrypt its message in the first step. $Nonce_{WC}$ is a random number generated by the registration server and used as a challenge to K to authenticate itself. Finally, MAC is used to ensure that K can verify the message integrity.

On receiving this message, K first decrypts the message using its private key and verifies its integrity. If verified, K authenticates the registration server by checking $Nonce_{CW} + 1$. If the authentication process succeeds, K executes the third step by sending a message containing $Nonce_{WC} + 1$ and a MAC computed over it. This message is encrypted using the public key of the registration server. On receiving this message, the registration server verifies its integrity and then authenticates the client.

These three steps have accomplished the following: (1) The client has authenticated the registration server; (2) The registration server has authenticated the client; and (3) The client has communicated the authorization information to the registration server and the registration server has determined the eligibility of the client to join the multicast group. Message confidentiality is implemented by encrypting all messages using appropriate public keys. Message integrity is implemented by including appropriate MACs. Man-in-the-middle attack and replay attack [19] are prevented by including appropriate nonce and implementing a challenge-response protocol.

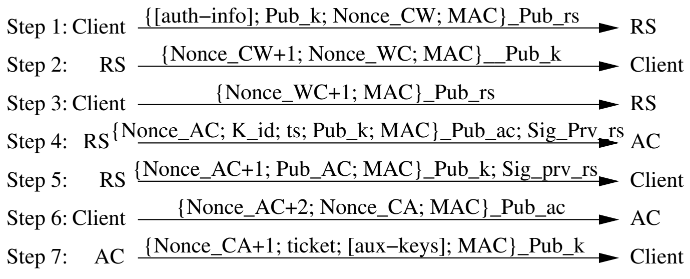
Step 1: Client {[auth−info]; Pub_k; Nonce_CW; MAC}_Pub_rs RS
Step 2: RS {Nonce_CW+1; Nonce_WC; MAC}__Pub_k Client
Step 3: Client {Nonce_WC+1; MAC}_Pub_rs RS
Step 4: RS {Nonce_AC; K_id; ts; Pub_k; MAC}_Pub_ac; Sig_Prv_rs AC
Step 5: RS {Nonce_AC+1; Pub_AC; MAC}_Pub_k; Sig_prv_rs Client
Step 6: Client {Nonce_AC+2; Nonce_CA; MAC}_Pub_ac AC
Step 7: AC {Nonce_CA+1; ticket; [aux−keys]; MAC}_Pub_k Client

Fig. 3. Join Protocol.

In the fourth step, the registration server first chooses an appropriate area for K and sends a message to the area controller of that area. The choice of an appropriate area can be based on factors such as proximity to the client, load balancing, etc. This message contains the following information: $Nonce_{AC}$; client K's id; Timestamp; K's public key; and a MAC computed over the first four pieces of information. This message is encrypted using the public key of the area controller and signed using the private key of the registration server. $Nonce_{AC}$ is a random number generated by the registration server that will be used later for authentication of the client to the area controller. Timestamp is included here to prevent a replay attack, in which an adversary that is able to intercept messages exchanged in this step and later in step 6, and gain access to the multicast group later on by replaying those messages. Timestamp is used by an area controller to catch such replayed messages. The signature generated using the registration server's private key is used by the area controller to authenticate that this message indeed originated from the registration server.

After sending the message in step 4, the registration server executes step 5 by sending a message to the client K. This message contains $Nonce_{AC} + 1$; public key and address of the area controller; and MAC computed over the two pieces of information. This message is encrypted using the public key of K, and signed using the private key of the registration server. Here $Nonce_{AC}$ is same as the one used in step 4. This will be used by K to authenticate itself to the area controller in step 6. Once again, the signature generated using the registration server's private key is used by K to authenticate that this message indeed originated from the registration server.

After receiving this message from the registration server in step 5, K first authenticates the sender of the message to be the registration server by verifying the signature. It then decrypts the message using its private key and verifies its integrity. Next, in step 6, it sends a message to the area controller. This message contains $Nonce_{AC} + 2$, $Nonce_{CA}$, and a MAC computed over these two numbers. This message is encrypted using the area controller's public key. $Nonce_{CA}$ is a random number generated by K that is used as a challenge to the area controller to authenticate itself.

After receiving the message from the registration server in step 4, the area controller verifies the signature of the registration server attached to this message. It then decrypts the message using its private key and verifies its integrity. Similarly, on receiving the message from client K in step 6, the area controller decrypts it using its private key and verifies its integrity. If verified, it authenticates the client by comparing $Nonce_{AC} + 2$ received in this message with $Nonce_{AC}$ received in the message from the registration server in step 4. If all information in these two messages are verified to be correct, the area controller adds client K in its area. It finds a position for K in its auxiliary key tree and updates the auxiliary keys in a similar fashion to the join protocol in LKH [21]. The details of auxiliary key tree update are described in Section III-C.

In step 7, the area controller sends a message to the client K containing the following information: all auxiliary keys in the path from K to the root of the auxiliary key tree; $Nonce_{CA} + 1$; a ticket; and a MAC generated over these three pieces of information. This message is encrypted using the public key of K. $Nonce_{CA} + 1$ is used by K to authenticate that this message indeed originated from the area controller that knows the private key corresponding to the public key used to encrypt the message in step 6. Ticket is a cryptographic entity that is by K to move from one area to another. Details of ticket structure and how it is used are described in Section IV-B.

C. Auxiliary Key Tree Update

The area controller creates a new area key $K_a^{'}$ and multicasts $E_{K_a}(K_a^{'})$ (encryption of $K_a^{'}$ using the previous area key K_a) to all current area members. In addition, it determines an appropriate (empty) leaf position in the auxiliary key tree of its area to place K. Area controllers in Mykil maintain a balanced tree structure of auxiliary keys such that each node (except leaf) in this tree has up to four children. This is based on the observation that a tree structure with each node having four children provides the best overall performance [21].

If an empty leaf position is found in the current auxiliary-key tree, K is placed at that position. However, if no empty leaf position is present in the current auxiliary-key tree, the following procedure is followed. First, find the shallowest, left-most leaf node n. Let member m_c is currently associated with this leaf. Create four new auxiliary-key nodes, $n1^{'}$, $n2^{'}$, $n3^{'}$ and $n4^{'}$, place them as children of n, and respectively assign randomly generated auxiliary keys $k1^{'}$, $k2^{'}$, $k3^{'}$ and $k4^{'}$. Next associate member m_c with node $n1^{'}$ and K with node $n2^{'}$. Unicast the list of new auxiliary keys appropriately encrypted to m_c. An example of member join is illustrated in Figure 4.

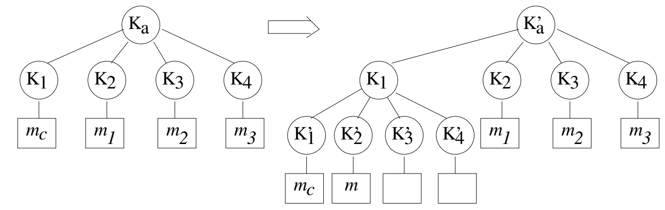

Fig. 4. A new member m joins an area.

D. Member Leave

The member leave protocol is again similar to that in LKH, except for one important step. Since the join operation is much less expensive if an empty leaf is already present in the tree, Mykil increases the likelihood of this scenario by not pruning the leaf after a member leaves. All keys along the path from the node corresponding to the leaving member to the root are changed, and the changed keys are multicast by encrypting them using appropriate auxiliary keys. An example when member m_1 decides to leave is shown in Figure 5 (shown as a binary tree here for simplicity). In this example, keys along the path from the root to the leaf corresponding to m_1, i.e., K_a, K_2, and K_4 are changed to $K_a^{'}$, $K_2^{'}$, and $K_4^{'}$ respectively. Next, the changed keys are multicast as $E_{K_2^{'}}(K_a^{'})$, $E_{K_3}(K_a^{'})$, $E_{K_4^{'}}(K_2^{'})$, and $E_{K_5}(K_2^{'})$.

E. Batching

An important technique called *batching* has been proposed [4], [22], [9], [17] to reduce the frequency of rekeying operations. The main idea of batching is to perform a rekeying operation only after a minimum number of member join/leave requests have been received, and/or a certain time interval has elapsed. Mykil employs batching to reduce the overhead of rekeying operations in three ways: (1) aggregation of join events, (2) aggregation of leave events, and (3) aggregation of join and leave events. The main idea is that all join and leave events are aggregated at an area controller until a new multicast data is received. The keys are updated just before the multicast data is forwarded.

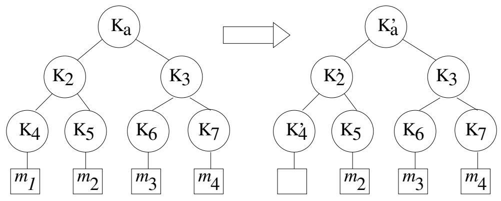

Fig. 5. Member m_1 requests to leave the group.

Assume that a new member m wishes to join an area A. To do this, the area controller A_c has to update the area key to say $K_a^{'}$. Assume right after that another new member m' wishes to join the same area. A_c has to now update the area key to $K_a^{''}$. If no data multicast occurs between these two join events, there is no need for A_c to multicast area key $K_a^{'}$ to the current area members. Aggregation of join events in Mykil prevents such needless multicast. To do so, when a new join request from a new member m arrives at A_c, A_c simply calculates a new leaf position for m, records the identity of m and other current members whose path may have changed due to this join, and sets an *update needed* flag. When a new multicast data packet arrives at A_c, A_c checks the update needed flag. If set, it multicasts a new area key to the current area members, sends appropriate unicast messages to the members whose identities were recorded, resets the update needed flag, and then forwards the multicast data.

In similar manner, all consecutive leave events are aggregated until a new multicast data packet arrives at A_c. We explain the distribution of updated auxiliary keys through an example illustrated in Figure 6. Suppose two members m_5 and m_8 leave the group consecutively. Rekeying operation involving leave event of m_5 will update keys K_a, K_3, K_6, and K_{12}. Rekeying operation involving leave event of m_8 will update keys K_a, K_3, K_7, and K_{15}. Notice that keys K_1 and K_3 are unnecessarily updated twice here. Aggregation of leave events avoids such unnecessary key updates. In this example, keys K_a, K_3, K_6, K_7, K_{12}, and K_{15} are updated only once to $K_1^{'}$, $K_3^{'}$, $K_6^{'}$, $K_7^{'}$, $K_{12}^{'}$, and $K_{15}^{'}$ respectively. These keys are multicast by encrypting as follows: $E_{K_2}(K_a^{'})$, $E_{K_3^{'}}(K_a^{'})$, $E_{K_6^{'}}(K_3^{'})$, $E_{K_7^{'}}(K_3^{'})$, $E_{K_{13}}(K_6^{'})$, and $E_{K_{14}}(K_7^{'})$. Aggregation of leave events can save a significant amount of bandwidth, because leave events typically occur together in several real-world applications. For example, members cancelling their cable memberships at the end of a month, or video-on-demand membership at the end of a movie.

Finally, both join and leave events are aggregated in Mykil. The procedure to do so is essentially a union of the join aggregation and leave aggregation procedures with minor changes. Notice that if a significantly large number of join and/or leave events are aggregated, a subsequent rekeying

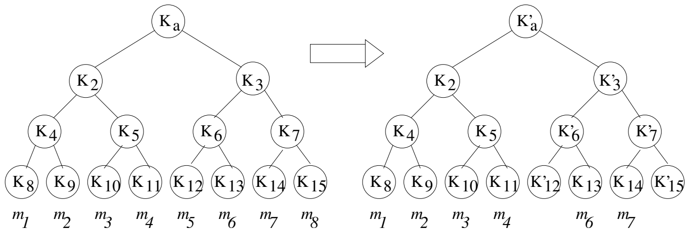

Fig. 6. Members m_5 and m_8 leave the group.

operation can become quite complex and time consuming. In particular, this may significantly delay the forwarding of multicast data. To address this, Mykil performs a rekeying operation under two conditions: (1) when a new multicast data packet is received by an area controller and the update needed flag is set, and (2) when a specific time interval has elapsed since the last rekeying operation. Rekeying under the latter condition preserves the freshness of the area key, and reduces the complexity of rekeying operation.

Key update messages sent by an area controller after a member join/leave must be authenticated. Otherwise, in the absence of authentication, any area member can send a key update message. In Mykil, each key update message is signed using the private key of the area controller. The use of private-key-based signature can be expensive. However, given that batching will be used in any practical multicast service, key update messages will not be very frequent. For authenticating the source of a multicast data, we can use faster methods such as those proposed in [16], [3].

IV. Support for Member Mobility and Fault Tolerance

An important contribution of Mykil is that it is designed to provide support for mobile group members and operation in a disconnected environment. Common failures such as network communication partitions or intermediate node/router crashes can result in a loss of communication between a group member and the area controller of its area. Member mobility can result in either a loss of communication, or degraded communication between the member and its area controller. Finally, failure of the area controller of an area will result in the loss of multicast service for all group members with in that area, and perhaps some other downstream areas.

The decentralized nature of Mykil allows operation in a disconnected environment. As long as a member can contact its area controller, it can continue to multicast data and receive data multicast by another member with in the same partition of the network. Furthermore, it can continue to receive all rekeying messages. However, if a member loses contact with its area controller, it can neither receive, nor send any multicast data. Support for member mobility and fault tolerance in Mykil is comprised of four parts:

1) When a group member detects that it can no longer communicate with its area controller, it attempts to join another area by contacting that area's area controller.
2) When an area controller detects that it can no longer communicate with one of its area member, it terminates the membership of that member from its area.
3) When an area controller detects that it can no longer communicate with the area controller of its parent area, it attempts to change its parent area by contacting another area's area controller.
4) Finally, an area controller itself is replicated to tolerate node failures.

A. Communication Failure: Detection

To enable group members and area controllers detect communication problems, area controllers of each area multicast *alive* messages with in their respective areas whenever they encounter an idle period. An idle period occurs at an area controller when it hasn't multicast any message in its area in the last T_{idle} time units. In addition, each group member sends an *alive* message to its area controller whenever it determines that it has not sent any message to its area controller in the last T_{active} time units. Typically the value of T_{active} is much larger than the value of T_{idle}. Based on this, each member or area controller can implement its own criteria to decide if it has been disconnected. For example, a member can decide that it has been disconnected from its area controller, if it has not received any message from it in the last $5 * T_{idle}$ time units. Similarly, an area controller can decide that one of its area member has been disconnected, if it does not receive any message from that member in the last $5 * T_{active}$ time units.

B. Member Rejoin

When a member determines that it has been disconnected from its area controller, it attempts to join another area. This can of course be done by contacting the registration server and repeating the entire join process described in Section III-B. However, there are two disadvantages in following this procedure. First, the entire join process is extensive and requires the client to submit authorization information. This can be an extra burden on a client. Second, since a member is typically granted access to a secure multicast group for a fixed period of time, it is important to make sure that the member is never denied access to the multicast group during this time period, and is not allowed to access the multicast group after this time period is over. For example, if the supported multicast application charges the users to become group members for a given period of time, a member that is only changing its area should not be charged again.

Mykil uses tickets to enable a member to join a new area without going through the extensive join process. This procedure is similar to the one used in Kerberos [14]. Recall that a member is given a ticket when it joins a multicast group. A ticket works like a ski pass. Intuitively, a ski pass has "Time of purchase", "Validity period", "Pass holder's picture/name/signature", and a "bar code" that cannot be tampered with. On similar lines, a ticket has the following information embedded in it:

- Join time: The time when the member joins the group.

- Validity period: Ticket expiry time.
- ID: a unique id for the member; e.g. MAC address of the NIC (Network Interface Card) used by the member.
- Public key: Public key of the member.
- Area controller: Area controller ID of the last area to which the member belonged.
- MAC: Computed over all the above information.

To ensure that the contents of a ticket cannot be changed by anyone without being detected, Mykil makes use of a secret symmetric key K_{shared} shared between all the area controllers. The information contained in a ticket is encrypted using K_{shared}. When a member attempts to join another area, it presents its ticket to the area controller of the new area. The area controller verifies this ticket before granting membership to the rejoining user. This is similar to a single ski pass that is good for skiing in five different resorts; all ski resorts will have the same scheme of scanning the bar code and verifying the validity of a ski pass. Figure 7 shows the details of the *rejoin protocol* that is executed when a member attempts to join a new area B with area controller AC_B. The area controller of the area A, which the member belonged last is AC_A.

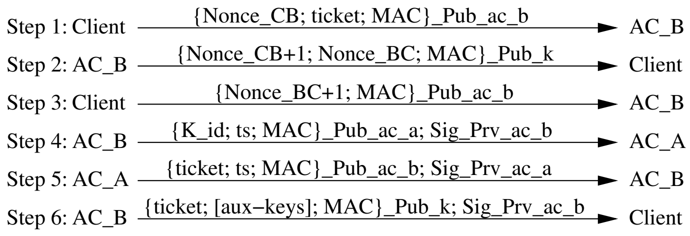
Step 1: Client
{Nonce_CB; ticket; MAC}_Pub_ac_b
AC_B
Step 2: AC_B
{Nonce_CB+1; Nonce_BC; MAC}_Pub_k
Client
Step 3: Client
{Nonce_BC+1; MAC}_Pub_ac_b
AC_B
Step 4: AC_B
{K_id; ts; MAC}_Pub_ac_a; Sig_Prv_ac_b
AC_A
Step 5: AC_A
{ticket; ts; MAC}_Pub_ac_b; Sig_Prv_ac_a
AC_B
Step 6: AC_B
{ticket; [aux−keys]; MAC}_Pub_k; Sig_Prv_ac_b
Client

Fig. 7. Rejoin protocol to join another area.

The rejoin protocol is comprised of six steps, and employs security mechanisms similar to those used in the join protocol (Section III-B) to address message confidentiality, message integrity, authentication, man-in-the-middle attack, and replay attack. In the first step, the member sends a message to AC_B containing the following information: $Nonce_{CB}$; Ticket; and a MAC computed over the first two pieces of information. This message is encrypted using the public key of AC_B. On receiving this message, AC_B first decrypts the message and verifies the integrity of the message and the validity of the ticket. If verified, it sends a message to the member containing $Nonce_{CB}+1$; $Nonce_{BC}$; and a MAC computed over the first two pieces of information. This message is encrypted using the public key of the member. $Nonce_{CB}+1$ is used by the client to authenticate AC_B, and $Nonce_{BC}$ is a challenge to the member to authenticate itself. In the third step, the member authenticates itself to AC_B by sending a message containing $Nonce_{BC}+1$ and a MAC computed over $Nonce_{BC}+1$ to AC_B. This message is encrypted using the public key of AC_B.

At this stage, AC_B has all the information to let the member join its area. However, there is a possibility that a malicious client C_1 may have shared its ticket and public/private key pair with its cohort C_2, and C_2 is attempting to join area B, even though C_1 is still a member of area A. Notice that the mutual authentication procedure implemented in the first three steps prevents an adversary C_a who happens to steal C_1's ticket from joining the group. This is because C_a does not know the corresponding private key of C_1, and so cannot authenticate itself (impersonate C_1) in the third step.

To address the issue of malicious cohorts sharing a ticket and and the corresponding public/private key pair, steps 4 and 5 are included in the rejoin protocol. In these steps, AC_B first contacts AC_A to ensure that the client is no longer a member of A. Recall that the identity of area A is included in the ticket. In step 4, AC_B sends a message to AC_A containing the following information: client's ID; Time stamp; and a MAC computed over the first two pieces of information. This message is encrypted using the public key of AC_A, and signed using the private key of AC_B. Here time stamp is included to prevent a replay attack by an adversary who happens to sniff this message. After verifying the validity of this message and if the client is no longer a member of A, AC_A sends a message to AC_B containing the following information: client's ticket; time stamp; and a MAC computed over the first two pieces of information. This message is encrypted using the public key of AC_B and signed using the private key of AC_A.

Finally, after receiving and verifying the validity of this message, AC_B lets the client join its area by adding it in its auxiliary key tree, and sending a message containing the following information: all auxiliary keys in the path from the client's leaf to the root of the auxiliary key tree; an updated ticket; and a MAC computed over the two pieces of information. This message is encrypted using the public key of the client and signed using the private key of AC_B.

Steps 4 and 5 require that the area controllers AC_A and AC_B exchange messages between each other. This may not be possible, if there is a communication partition between the two areas. In case AC_B cannot communicate with AC_A, there are two options: (1) AC_B denies the member to join its area; and (2) AC_B allows the member to join its area without verifying that the member has indeed left area A. The first option is unfair to a legitimate mobile client, but prevents misuse by malicious cohorts that share keys and tickets. The second option on the hand allows members to avail multicast service despite communication partitions, but is subject to the misuse bu malicious cohorts. One way to prevent this misuse in the second option is to verify the MAC address of the NIC being used by the rejoining member. Recall that the MAC address of the NIC is stored in a ticket. This will require a member to use the same NIC when rejoining. Of course, it is indeed possible for a malicious cohort to tamper with the NIC address. However, doing so will be difficult, and hopefully diminish any gains from sharing the ticket.

There is also the issue of how does a group member find out the address and public key of a new area controller while moving. It needs them to start the rejoin protocol. One way to do this is to have the registration server provide a list of all area controllers' addresses and public keys when a member registers with a multicast group (step 5 of join protocol).

C. Fault Tolerance: Area Controller

When an area controller determines that it can no longer communicate with one of its area member, it unilaterally terminates the membership of that member from its area. It essentially executes the protocol for member leave described in Section III-D.

When an area controller A_c detects that it can no longer communicate with the area controller of its parent area, it attempts to change its parent area by contacting another area's area controller. To do so, each area controller maintains a list of one or more preferred area controllers. A_c chooses an appropriate area controller AC_P from this list, and sends a *area-join* request to that area controller. This request contains the following information: A_c's identity; Time stamp; and a MAC computed over the first two pieces of information. This message is encrypted using the public key of AC_P, and is signed using A_c's private key. On receiving this message, AC_P verifies the signature and the integrity of the message. If verified, it send an *ack* message to A_c containing a its area key, time stamp and a MAC. This message is encrypted using the public key of A_c, and signed using the private key of AC_P.

Finally, Mykil replicates an area controller to tolerate the failure of a node on which an area controller is running. We assume a *crash failure model* [6] for node failures in our implementation. A primary-backup mechanism [1], [2] is used to manage this replication. To minimize the performance overhead of this primary-backup replication, only a minimal state information is replicated. This includes the complete auxiliary tree, public keys of the area members, area controllers and the registration server, and the identities of the parent area controller and all child area controllers. Primary and backup servers are synchronized during any key updates, and whenever there is a change in the parent/child area controllers.

Notice that the state information about the multicast messages being exchanged is not replicated. This is done to minimize communication between primary and backup servers. A consequence of this is that the area members may not receive some multicast messages while the backup server takes over the primary during a primary server failure. The backup server monitors the health of the primary server by exchanging heartbeat messages at regular time intervals.

V. Implementation and Performance

A prototype of Mykil has been implemented on a network of Linux workstations. TCP has been used for communication between area controllers for forwarding multicast data. OpenSSL libraries for cryptography has been used. We have used RSA_public_encrypt and RSA_private_decrypt for encryption and decryption, and RSA_sign and RSA_verify for digital signatures and signature verification. We used 2048 bit RSA keys in the join protocol and 128 bit symmetric keys for area and auxiliary keys. We evaluate Mykil with respect to 5 criteria: storage requirements; computation requirements; bandwidth requirements; join and rejoin protocol performance; and feasibility on hand-held devices.

A. Storage Requirements

In addition to its own public/private key pair, each member needs to store the public keys of its area controller and the registration server. Given a 2048 bit RSA key, a member will need at least 2048*4 = 8192 bits (1 KB) to store these keys. In addition, a member may store the public keys of other area controllers that are needed in the rejoin protocol. If there are 10 such other area controllers, a member will need an additional 2.5 KB of memory. Initial registration protocol is not described in detail for Iolus or LKH. However, public keys will definitely be needed in Iolus and LKH as well to facilitate initial registration. Our guess is that a member will need to store four or five public keys in these protocols, requiring approximately 1-2 KB of memory.

In addition, a group member needs to store symmetric keys as well. Suppose a multicast group consists of 100,000 members. In LKH, this will result in an auxiliary-key tree of depth 16 (4 children for each node). This implies that each member will have to store 16 auxiliary keys and a group requiring 128*17 = 2176 bits (272 B). On the other hand a member in Iolus will need to store 2 keys, an area key and a pairwise secret key with area controller. Assuming that we limit the membership size of an area to about 5000 members in Mykil, a member in Mykil will need to store about 11 keys. This means that a user needs 32 bytes in Iolus, 272 bytes in LKH, 176 bytes in Mykil to store the required symmetric keys. This shows that Iolus incurs minimum amount of storage overhead and LKH incurs the maximum amount of overhead. Mykil's storage overhead per member falls in between. An important point to note is that the memory requirements to store cryptographic are fairly small in all three protocols.

Storage requirements to store keys at the area controllers or key management servers are relatively high. In Mykil, an area controller needs to store public keys of all other area controllers and registration server, and all auxiliary keys. Again, for a group of 100,000 members divided into 20 areas, this requirement is about 132 KB (5 KB for 20 public keys; 127 KB for 8092 symmetric keys) in Mykil. In LKH, a key management server will have to store approximately 2^18 auxiliary keys. This will require about 4 MB of memory. In Iolus, a subgroup controller will need about 80 KB (5001 symmetric keys; some public keys). Thus the storage requirements for area controller in Mykil and subgroup controller in Iolus are moderate, while they are significantly larger for key management server in LKH.

B. CPU Requirements

For a join event, the computational requirements at the joining member are similar in all three protocols. The joining member receives the new area/group key and some auxiliary keys. However, a join event requires existing members to do some computation as well. In particular, group key of all members is updated in LKH, while area key of the members of only one area is updated in Iolus and Mykil. So, on an average, the CPU requirements are larger in LKH compared to Iolus or Mykil during a join event.

For a leave event, each member of one area will receive a new area key in Iolus. For a group of 100,000 members with maximum area size of 5000 members, 5000 members will update one key. In case of LKH, on an average, 50% of members will need to update one key, 25% will update two keys, 12.5% will update three keys, 6.25% will update four keys, and so on. For a group of 100,000 members, this implies that 50,000 members will update one key, 25,000 members will update two keys, 12,500 members will update three keys, 6,250 members will update four keys, and so on. Finally, in Mykil, only the members with in one area are affected. For an area of 5000 members, 2500 members will update one key, 1250 members will update two keys, 625 members will update three keys, 313 members will update four keys, and so on. This shows that Iolus incurs minimum amount of CPU overhead per member, mykil incurs a little larger CPU overhead, and LKH incurs significantly larger overhead.

C. Bandwidth Consumption

The bandwidth consumption per group member during a rekeying operation depends on the length of the key update message. For a join event, the length of key update message that is multicast is same in all three protocols, i.e. the length of the encrypted new group/area key. In addition, LKH and Mykil also unicast the key path to the new member. In Mykil, this corresponds to 16*12 = 172 bytes in an area of 5000 members. In LKH, this corresponds to 16*17 = 272 bytes for a group of 100,000 members.

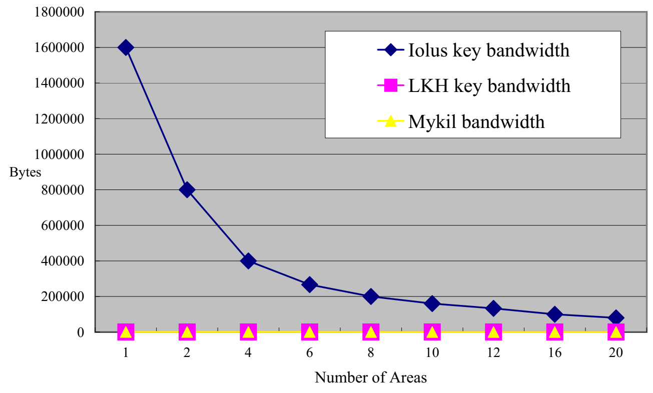

Fig. 8. Bandwidth consumption during a leave event.

For a leave event, the length of key update message in Iolus depends on the area size. For an area of 5000 members and assuming 128-bit keys, the length of this message will be about 80,000 bytes. In LKH and Mykil, the size of rekeying message during a leave event depends on the height of the tree. In particular, an updated key corresponding to a node n is encrypted separately by keys corresponding all children of n. Since, all keys along the path from the root to the leaf corresponding to the leaving member are updated, this implies a rekeying message of $2*17*16 = 544$ bytes in LKH (100,000 members in the group), and $2*12*16 = 384$ bytes in Mykil. Figure 8 shows the bandwidth requirements for the three protocols, and Figure 9 shows in detail the bandwidth requirements in Mykil and LKH. These graphs show clearly that both Mykil and LKH require significantly lower bandwidth at a member than Iolus. Bandwidth requirement in Mykil is further reduced by aggregating consecutive join or leave events. For example, Figure 10 shows the reduction in Mykil by aggregating ten consecutive leave events.

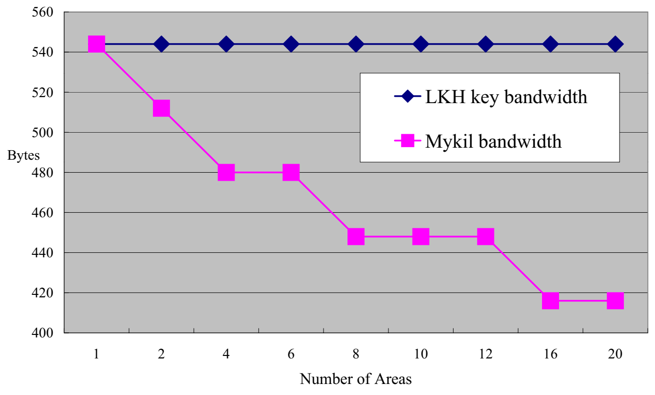

Fig. 9. Bandwidth consumption in Mykil and LKH during a leave event.

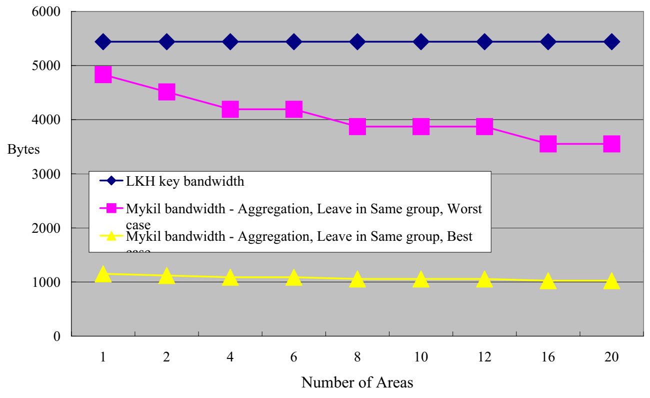

Fig. 10. Bandwidth consumption in Mykil with and without aggregation of leave events.

D. Join and Rejoin Protocol Performance

As mentioned earlier, we have used OpenSSL libraries for cryptography in our implementation. When a 2048-bit key is used, the length of data that can be encrypted by public key has an upper limit as 256 bytes in OpenSSL library. At least 41 bytes are needed for the padding method (RSA_PKCS1_OAEP_PADDING). So only 215 bytes of buffer can be used to store data in every encryption function call. We tried to fit our data into this 215 bytes to avoid the hassle of breaking a big chunk of data into smaller pieces, and reassemble them back together. In step 7 of the join protocol or step 6 of the rejoin protocol, an area controller needs to send a set of keys along the path from the joining member to the root in the auxiliary tree. This turned out to be too large to fit with in a 215 B buffer. So, in our implementation, the area controller creates a one-time symmetric key, communicates that key to the client by encrypting it using the public key of the client, and then send the set of auxiliary keys by encrypting them using the one-time symmetric key.

Average time for a member join (join protocol) was measured to be about 0.45 seconds on network of three Pentium III 1.0 GHz PCs running RedHat Linux 8.0. With RSA_blinding_on option on, which blinds the correlation between the amount of time taken for encryption/decryption and key value, time increases by only about 0.01 seconds for each join operation. Average time for a member rejoin (rejoin protocol) was measured to be about 0.4 seconds under the same computing environment. While there is not much difference in the performance of join and rejoin protocols, the rejoin protocol does not require any participation of the registration server, thus reducing communication and computation load on that server. Furthermore, if steps 4 and 5 are removed from the rejoin protocol (option 2 discussed in Section IV-B), the rejoin time is reduced to about 0.28 seconds.

E. Hand-held Devices

An important goal of Mykil is to enable clients access a multicast service via smaller, hand-held devices, such as PDAs and laptops. Current state-of-the-art PDAs running Linux have 400-500 MHz CPU and 16-32 MB memory [8]. Based on our discussion on storage requirements in Section V-A, it is clear that the storage requirements of Mykil can easily be satisfied by modern PDAs. To evaluate the computational feasibility of running Mykil on such devices, we ported Mykil on a low-end laptop (Celeron; 600 MHz; 64 MB RAM) running Linux. We experimented with RC4 encryption algorithm to encrypt/decrypt multicast data on it. We observed that it took about 0.32 seconds to encrypt/decrypt a 16 MB file, i.e. data can be encrypted or decrypted at about 50 MB/sec on this device. This much computation power is more than adequate for processing multimedia. For example, a 10 MB file can store one minute of a high resolution MPEG-4 film (resolution 720 x 416; sound quality 128 KBit/s 44 KHz). It will take only about 200 milliseconds to encrypt/decrypt this file using RC4 on a modern PDA. These preliminary experiments suggest that there should be no problem in porting Mykil on a modern PDA. We plan to so in the near future.

VI. Conclusion

We have proposed a key management protocol called Mykil that combines two hierarchy schemes in such a way that all important advantages of the two schemes are retained. These include scalability, mapping to the underlying network infrastructure, and operation in a disconnected environment. Mykil improves on LKH by reducing the resource requirements for a group member, providing support for operation in a disconnected environment, and providing an ability to map the group organization to the underlying network infrastructure. Mykil improves on Iolus by reducing the bandwidth requirements and eliminating the performance bottleneck of area controller. In addition, Mykil provides support for smaller, hand-held devices, user mobility, and robustness. An analysis shows that the resource requirements for a group member in Mykil are reasonable, and a client can avail of this protocol via smaller, hand-held devices. We have implemented Mykil at the application level on a network of Unix/Linux system. Performance measurement from this implementation shows that the performance of the two protocols (join and rejoin protocols) that are critical in providing support for member mobility and fault tolerance is adequate. Furthermore, we have also provided a proof-of-concept that Mykil is appropriate for group members using hand-held devices.

References

[1] N. Budhiraja and K. Marzullo. Highly-available services using the primary-backup approach. In *The 2nd Workshop on Management of Replicated Data*, Monterey, CA, 1992.

[2] N. Budhiraja, K. Marzullo, F. Schneider, and S. Toueg. Primary-backup protocols: Lower bounds and optimal implementations. In *The 3rd IFIP Conference on Dependable Computing for Critical Applications*, 1992.

[3] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: A taxonomy and some efficient constructions. In *Proceedings of INFOCOMM'99*, March 1999.

[4] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, and D. Daha. Key management for secure Internet multicast using boolean function minimization technique. In *ACM SIGCOMM'99*, March 1999.

[5] W. Chen and L. R. Dondeti. Recommendations in using group key management algorithms. URL: http://www-net.cs.umass.edu/papers/papers.html.

[6] F. Cristian. Understanding fault-tolerant distributed systems. *Communications of ACM*, 34(2):56–78, Feb 1991.

[7] T. Hardjono, B. Cain, and I. Monga. Intra-Domain group key management protocol (IGKMP). Internet draft, IETF, February 2000.

[8] S. Lemon. AMD demos Alchemy-based PDA running Linux. URL: http://www.infoworld.com/article/03/08/06/HNalchemy_1.html.

[9] X. Li, Y. Yang, M. Gouda, and S. Lam. Batch rekeying for secure group communications. In *The WWW Conference 10*, Hong Kong, May 2001.

[10] D. McGrew and A. Sherman. Key establishment in large dynamic groups using one-way function trees, May 1998. Available at *http://www.cs.umbc.edu/~sherman/Papers/itse.ps*.

[11] S. Mishra. Key management in large group multicast. Technical Report CU-CS-940-02, Department of Computer Science, University of Colorado, Boulder, CO., 2002.

[12] S. Mittra. Iolus: A framework for scalable secure multicasting. In *Proceedings of the ACM SIGCOMM'97*, September 1997.

[13] R. Molva and A. Pannetrat. Scalable multicast security in dynamic groups. In *Proceedings of the 6th ACM Conference on Computer and Communication Security*, November 1999.

[14] C. Neuman and T. Theodore. Kerberos: An authentication service for computer networks. *IEEE Communications*, 32(9), September 1994.

[15] A. Perrig, D. Song, and J. Tygar. ELK, a new protocol for efficient large-group key distribution. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, 2001.

[16] P. Rohatgi. A compact and fast hybrid signature scheme for multicast packet authentication. In *Proceedings of the 6th ACM Conference on Computer and Communication Security*, November 1999.

[17] S. Setia, S. Koussih, and S. Jajodia. Kronos: A scalable group re-keying approach for secure multicast. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, May 2000.

[18] G. Steiner, G. Tsudik, and M. Waidner. Key agreement in dynamic peer groups. *IEEE Transactions on Parallel and Distributed Systems*, August 2000.

[19] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 2002.

[20] D. Wallner, E. Harder, and R. Agee. Key management for multicast: Issues and architectures. Request For Comments (Informational) 2627, Internet Engineering Task Force, June 1999.

[21] C. Wong, M. Gouda, and S. Lam. Secure group communication using key graphs. In *Proceedings of the ACM SIGCOMM'98*, October 1998.

[22] R. Yang, S. Li, B. Zhang, and S. Lam. Reliable group rekeying: A performance analysis. In *ACM SIGCOMM'01*, August 2001.

[23] S. Zhu, S. Setia, and S. Jajodia. Performance optimizations for group key management schemes. In *The 23rd IEEE International Conference on Distributed Computing Systems*, Providence, RI, May 2003.