To appear in Proc. of 2000 IEEE Symposium on Security & Privacy

Kronos: A Scalable Group Re-Keying Approach for Secure Multicast

Sanjeev Setia[*] Samir Koussih[*] Sushil Jajodia
Center for Secure Information Systems
George Mason University
Fairfax, VA 22030
{setia,skoussih,jajodia}@gmu.edu

Eric Harder
National Security Agency
Fort Meade, MD 20755
ejh@tycho.ncsc.mil

Abstract

In this paper, we describe a novel approach to scalable group re-keying for secure multicast. Our approach, which we call Kronos, is based upon the idea of periodic group re-keying. We first motivate our approach by showing that if a group is re-keyed on each membership change, as the size of the group increases and/or the rate at which members leave and join the group increases, the frequency of re-keying becomes the primary bottleneck for scalable group re-keying. In contrast, Kronos can scale to handle large and dynamic groups because the frequency of re-keying is independent of the size and membership dynamics of the group. Next, we describe how Kronos can be used in conjunction with distributed key management frameworks such as IGKMP [10], that use a single group-wide session key for encrypting communications between members of the group. Using a detailed simulation, we compare the performance tradeoffs between Kronos and other key management protocols.

1 Introduction

Many emerging Internet applications (e.g., real-time information services, pay per view, computer-supported collaborative work) are based upon group communications. As the next generation of the Internet is deployed, many of these applications are expected to increase in importance. Network protocols that support multicast communications in an efficient and scalable manner are essential for applications based on group communications. Consequently, issues such as reliable delivery of data and congestion control in the context of multicasting over the Internet have been active areas of research over the last few years.

An issue that is critical for mainstream adoption of multicast technology is the need for *securing* multicast communications. In other words, it is important to ensure that multicast communications can only be received by the intended recipients. While security mechanisms for supporting unicast communications over the Internet have been studied extensively [13], it is only recently that the research community has started to address the issues involved in supporting secure multicasting [5].

The multicast service currently supported in the Internet (IP Multicast) does not have any provisions for restricting delivery of data to a specified set of receivers. Any receiver can join or leave a multicast group (identified by a Class D IP address [20]) by sending IGMP (Internet Group Management Protocol) [8] messages to their local router. Further, any user can send data to a multicast group by addressing the message to the group address. In other words, IP multicast does not support "closed" groups.

To restrict the flow of multicast data to a specific set of users, it is necessary to use cryptographic mechanisms. Specifically, messages are encrypted by senders using a session key that is only distributed to members of the group. Participation in a group session is restricted by ensuring that only members of the group have possession of the session key at any given time. Thus key management (creating and distributing session keys to authorized group members) is a critical aspect of secure multicast.

One of the issues that has to be addressed by key management schemes for secure multicast is the need for forward and backward confidentiality [23]. In other words, new members joining a group should not be able to access previously multicast data and old members should not be able to continue to access data multicast after they have left the group. For applications that require perfect forward and backward confidentiality, the session key used for encrypting group communications needs to be changed on each membership change and securely redistributed to the existing members of the group. This is referred to as group re-keying.

For large groups with frequent membership changes, the costs of re-keying the group can be quite substantial. The

[*] also with Dept. of Computer Science, George Mason University

straightforward approach under which a new session key is generated on each join and leave, and securely transmitted to each existing group member is not scalable to large groups. This is because the session key will have to be encrypted individually for each group member and the costs of doing so increase linearly with the size of the group. Scalable re-keying is therefore an important problem that needs to be addressed in order to support secure communications for large and dynamic groups.

Recently several protocols for group key management for secure multicast have been proposed. The techniques that specifically address the problem of scalable group re-keying fall under two categories. The first set of approaches [23, 24, 3, 22], typically involve creating a logical hierarchy of keys. The main focus of these schemes is to reduce the overhead of re-keying the session key on a membership change. The schemes are scalable because the computational overhead of re-keying is logarithmic in the number of members of the group. However, these approaches do not attempt to reduce the frequency at which the session key needs to be changed and redistributed to the members of the group. As we discuss in Section 2, for large and dynamic groups *the frequency of re-keying* imposes an upper limit on the scalability of key management protocols that is independent of the efficiency of an individual re-keying operation. Secondly, with the exception of the Distributed Flat scheme of Waldvogel *et al* [22], these approaches are centralized. As such, they have all the well-known advantages (from the security point of view) and disadvantages (from the performance and availability point of view) of providing a centralized service on the Internet.

The second approach to scalable group re-keying employs a divide-and-conquer approach. This approach [15] (henceforth referred to as Iolus) is inherently distributed in nature. A group is divided into several sub-groups each with its own session key. Membership changes in a sub-group result in a change of the sub-group session key and do not affect the remaining members of the group. Thus, both the frequency and computational overhead of re-keying is determined by the size of a sub-group instead of the size of the whole group. Under this approach, however, a sub-group manager is responsible for re-encrypting and relaying all multicast traffic flowing between its members and the rest of the group.

In this paper, we describe a novel approach to scalable group re-keying for secure multicast. Our approach, which we call Kronos, is based upon the idea of periodic re-keying. Periodic re-keying decouples the frequency of re-keying from the size and membership dynamics of the group. As such, our scheme can easily scale to large and dynamic groups. However, this is potentially at the expense of increased latencies experienced by members joining and leaving the group. Further, if the period between key changes is too large, the delay in evicting members may be unacceptable for some high-security applications. In Sections 3 and 4, we explore this tradeoff and show that by appropriately selecting the period of re-keying, acceptable join and leave latencies can be obtained.

Another advantage of periodic re-keying is that it permits scalable group re-keying within distributed frameworks for key management such as Iolus [15] or IGKMP [10], while allowing the use of single group-wide session key for encrypting communications between members of the group. In Section 3, we describe a scheme whereby each sub-group key manager independently generates the same traffic encryption key at fixed intervals and multicasts it to the members of its sub-group. Since there is a single group-wide traffic encryption key, the sub-group manager is no longer responsible for re-encrypting and relaying all data between its members and the rest of the group. At the same time, group re-keying is accomplished in an efficient and scalable fashion. We note that periodic re-keying has long been recognized as necessary from the security point of view [23, 10]. However, our emphasis on periodic re-keying is motivated by the need for scalable group re-keying for large and dynamic groups.

We make three contributions in this paper. First, we evaluate the impact of group size and group membership dynamics on the scalability of re-keying schemes. We show that if a group is re-keyed on each membership change, as the size of the group increases and/or the rate at which members leave and join the group increases, the frequency of re-keying becomes the primary bottleneck for scalable group re-keying. Second, we propose a scalable group re-keying approach based upon periodic re-keying that can be used in conjunction with distributed key management frameworks such as Iolus and IGKMP. Third, using a detailed simulation, we explore the performance tradeoffs between Kronos and other key management protocols for secure multicast.

The organization of the rest of the paper is as follows. In Section 2, we analyze impact of group size and group membership dynamics on the scalability of re-keying schemes. In Section 3, we describe the Kronos approach for group re-keying. In Section 4, we evaluate the tradeoffs between group re-keying schemes both qualitatively and quantitatively. Finally, Section 5 contains our conclusions.

2 The Case for Periodic Re-keying

To provide forward and backward confidentiality, key management protocols for secure multicast change the session key used for encrypting traffic whenever there is a membership change. As discussed in the introduction, several researchers have developed techniques that minimize the overhead of generating a new session key and redistributing it securely to the existing members of the group.

However, the total overhead for re-keying over a given period of time depends not only on the cost of an individual re-keying operation *but also upon how often re-keying is done during that period.*

An increase in the rate of re-keying results in an increase in the overhead of key management for several reasons. First, the computational overhead at the key manager for generating, encrypting, and transmitting the session key increases with the rate of re-keying. Second, the computational overhead of the group members for receiving and decrypting the key increases with the rate of re-keying. This is an important consideration especially for delay-sensitive applications and for applications executing on computers with limited resources. Third, the network overhead (number of messages per unit time and bandwidth consumed) for the re-keying traffic increases. Another factor that should be considered here is the overhead for ensuring that keys are transmitted in a reliable manner to the members of the group; this overhead may arise from a reliable multicast protocol or from an application-specific protocol for delivering keys to the group members.

The frequency of group re-keying depends upon two factors: (i) the size of the group, and (ii) group membership dynamics, i.e., the rate at which members join and leave the group. In this section, we use a simple model to analyze the impact of these two factors on the rate at which groups need to be re-keyed.

Consider an application (e.g., a real-time information delivery service) that can be expected to have large and dynamic groups. In this application, there is a single source multicasting a stream of data to a changing set of recipients. Assume that there exists a population of N subscribers or viewers who are potentially interested in the information being multicast. Each subscriber can be modeled as alternating between two states: a state in which s/he is tuned in to the multicast (i.e., a member of the multicast group being used by the application) and a state in which s/he is tuned out.

An important factor that affects the membership dynamics of a multicast group is the statistical correlation between the joining/leaving times of the members of the group. Currently, there is little empirical evidence available about group membership dynamics for large scale multicast applications. However, there are two extremal assumptions one can make about the correlation between the behavior of the members of the group: (i) that the patterns of joining/leaving the group are *highly correlated*, i.e., all subscribers join and leave the group at nearly the same time (ii) each subscriber's behavior is *independent* of that of other subscribers. For several applications, we can expect there to be a high degree of correlation in subscriber joining and leaving patterns, e.g., for a pay per view scenario, we can expect a lot of subscribers joining the group at the beginning of the broadcast and leaving at the end of the broadcast. For other applications, e.g., a real-time information delivery service, individual subscriber joining/leaving patterns are more independent. We discuss both cases below.

Case 1: Correlated Subscriber Behavior Consider the situation where thousands of subscribers attempt to join or leave a group at roughly the same time. In this case, the group manager can expect to receive a flurry of join or leave requests over a short period of time at the beginning or end of a broadcast. In this situation, re-keying the group on each and every membership change is clearly untenable. Instead, it is preferable to use a scheme under which the group is re-keyed periodically and the new key distributed in a scalable manner to all the current members of the group.

Case 2: Independent Subscriber Behavior The motivation for periodic re-keying is less clear in the second case: when each subscriber's behavior is independent of that of other subscribers. We use a simple analytical model to analyze this case. Assume that the time for which a subscriber is tuned in to a multicast is exponentially distributed with mean $1/\mu$. Further, assume that the time during which a subscriber is *not tuned in* to the multicast is exponentially distributed with mean R/μ. Thus R is the ratio of the average time for which a subscriber is tuned out to the average time for which a subscriber is tuned in.

Under these assumptions, the number of subscribers that are tuned in to a broadcast can be modeled by a birth-death Markov process [21]. Note that the number of subscribers tuned in to the multicast corresponds to users who are members of the multicast group under consideration. Thus, each time there is a membership change the group will need to be re-keyed. We can show (see Appendix A) that the expected time interval between re-keys is given by

$$T_{rekey} = (1 + R)/(2\mu N)$$

This shows that as the product μN of the subscriber population (N) and the reciprocal of the time for which a user tunes in to a broadcast ($1/\mu$) increases, the time between re-keys decreases. In other words, the larger the subscriber population and the smaller the time for which a subscriber tunes in to a multicast, the smaller the interval between re-keys. The factor R has the opposite effect; the larger R is the larger the interval between re-keys.

Assume that the key manager generates a new key in response to a membership change and multicasts it to the current members of the group at time T_s. Assume that the average delay after which this new key reaches the members of the group, i.e., the network delay of propagating the key to the members of the group, is represented by Δ. Thus, on average, a member of the multicast group will receive the new key at time $T_r = T_s + \Delta$.

If the expected time between re-keys, T_{rekey} is smaller than Δ, it implies that the key being used to encrypt data will be need to be changed even before a majority of the members of the group have received the previous version of the key. Re-keying at a higher frequency will lead to situations where several keys are in transit at any given time increasing the probability of transient security breaches and confusion. From a practical viewpoint, it is difficult to conceive of applications that can tolerate average re-key periods smaller than Δ. In that sense, Δ represents a lower limit on the time interval between re-keys (or an upper bound on rate of re-keying). Note that this lower limit arises even if the computational overhead of re-keying is negligible, i.e., even if a key manager has infinite processing power available to it. The conditions under which this re-keying interval is reached thus represent limits on the scalability of approaches that re-key the group on each membership change.

We note that there is no inherent restriction that prevents an application from having T_{rekey} smaller than Δ. Indeed, it could be argued that the smallest T_{rekey} that can be tolerated depends largely on the application under consideration. Our reasoning for selecting Δ as a lower bound on T_{rekey} are based on the practical concerns expressed above.

We now compute the conditions that lead to $T_{rekey} = \Delta$. For large groups spread across the Internet, 50 ms represents a reasonable value of the average delay Δ before a new key reaches the majority of the members of the group. In Figure 1, we plot the values of N (the number of subscribers) and $1/\mu$ (the average time a subscriber tunes in to the broadcast) that will result in the expected time between re-keys being equal to 50 ms for $R = 0.5, 1, 2, 10$ and 100.

The area under the line for a particular value of R corresponds to combinations of N and $1/\mu$ for which the expected time between rekeys will be smaller than 50 ms. For example, consider the plot for $R = 1$. When $N = 10000$ the limiting value of the average tune-in time is 500 seconds. In other words, the time between re-keys will become unacceptably small for average tune-in times smaller than 500 seconds. Similarly, for $N = 100000$ and $N = 1000$, the smallest average tune-in times that can be handled are 5000 seconds and 50 seconds respectively. Note that these limits are independent of the processing power available to the key manager.

It is important to note that for large subscriber populations it is not necessary for the group membership to be very dynamic before the frequency of re-keying becomes unsustainable. For example, when $N = 500000$ and $R = 100$, the minimum tune-in time is over twelve minutes (see the point labelled Y in Figure 1). Note that N represents the maximum subscriber population and not the average size of the group that is tuned in. Conversely, when membership is very dynamic (tune-in time = 1 minute, R = 10), the maximum subscriber population that can be sustained is 6600 (see the point labelled X in Figure 1).

While these results have been obtained using simplifying assumptions (e.g., exponentially distributed tune-in times), similar results are obtained (albeit via simulation) if we assume that group membership dynamics are driven by long-tailed distributions similar to those observed for other network processes [17].

These results show that for large and dynamic groups the frequency at which the group is re-keyed imposes an upper limit on the scalability of the key management protocol that is independent of the efficiency of an individual re-keying operation. For a key management protocol to be able to scale to large and dynamic groups, it is necessary to address the factors that lead to high re-keying frequencies.

Previous works have attempted to reduce the frequency of re-keying in two ways. First, it is possible to avoid the overhead of distributing a new key to all the members of the group on joins by generating the new version of the session key by applying a one way function to the previous key. Under this approach, which has been adopted in the Versakey project [22] and in the LKH+[12] protocol, the overhead of transmitting a new key to the existing members of the group is only incurred when a member leaves the group [1]. Second, under the Iolus approach, the frequency of re-keying is reduced because sub-groups have both smaller sizes and smaller rates of joins/leaves than the entire group.

A third approach to reducing the frequency of re-keying is to decouple it altogether from group size and membership dynamics. This can be accomplished by periodic re-keying, i.e., by re-keying the group at fixed intervals instead of having the re-keying operation be driven by member joins and leaves. In the next section, we describe an approach to scalable group re-keying that is based upon this observation.

3 The Kronos Approach for Scalable Group Re-keying

Kronos is a scalable approach for re-keying large and dynamic groups. The analysis in Section 2 shows that as groups become large and/or dynamic, re-keying the group on each membership change becomes unsustainable irrespective of the efficiency of an individual re-keying operation. By re-keying the group at fixed intervals, we can decouple the frequency of re-keying from the group size and membership dynamics. If the interval between re-keys is large enough, the scalability of a centralized key management protocol that uses periodic re-keying is determined solely by the efficiency of an individual group re-keying operation. By using approaches such as LKH [23, 22, 24] or One-way Function trees [2], such a protocol can scale to large groups.

[1] For these protocols, $T_{rekey} = (1 + R)/(N\mu)$

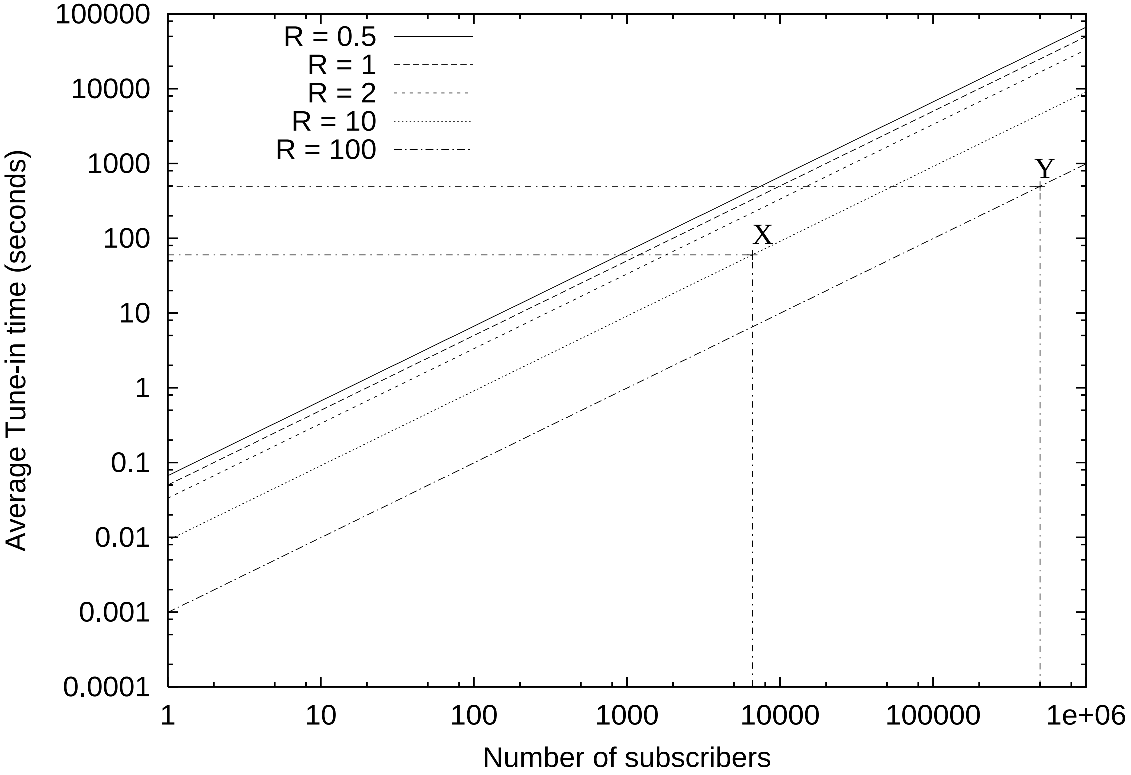


Figure 1. The plots for $R = 0.5, 1, 2, 10$ and 100 show the combination of values of N (number of subscribers) and $1/\mu$ (average subscriber tune-in time) that lead to an expected time between re-keys of 50 ms. The point labeled X corresponds to $1/\mu = 60$ seconds, $R = 10$, and $N = 6600$ while the point labelled Y corresponds to $N = 500000$, $R = 100$ and $1/\mu = 496$ seconds.

While a centralized key management service can be implemented without requiring a distributed trusted infrastructure, it suffers from all the well-known disadvantages of providing a centralized service on the Internet, viz. a single point of failure, high latencies for (network-wise) distant hosts, unavailability due to network partitions, etc. To robustly support large multicast groups with members spread across the Internet, it is necessary to distribute the key management protocol.

We note that several issues that need to be addressed for supporting distributed key management protocol such as multicast address allocation and the design of protocols for the creation, initialization, and advertisement of multicast groups are still at the research stage. We envision that Kronos would be used within a distributed key management framework such as that proposed by Hardjano *et al* [9, 10] or Mittra [15]. Our focus in this paper is not on how such a framework should be established but on how scalable group re-keying can be accomplished once such a framework is in place.

3.1 Distributed Key Management Frameworks

Mittra [15] and Hardjano *et al* [10] have proposed frameworks for distributing the task of key management for secure multicast groups. Under the Intra-domain Group Key Management Protocol (IGKMP), an administrative domain is divided into several “areas”. Each host-member of a multicast group is assumed to reside in a particular area. IGKMP distinguishes between multicast groups for the purpose of key management and payload delivery. There is a domain-wide key distributor (DKD) and an area key distributor (AKD) corresponding to each area. Each host-member in a specific area is a member of a multicast group established for the purpose of key distribution that includes its AKD. All the AKDs and the DKD are members of another multicast group that is used by the DKD for transmitting the multicast data encryption key to the AKD, which in turn, transmits it to each host-member in its area. Note that there is a single group-wide multicast data encryption key under this scheme.

Before a host can start participating in a group session, it has to join the group. Joining a group involves sending a request to the AKD. The AKD authenticates the request and checks the credentials of the member. If the member is allowed to join the group, the AKD will establish a private key that is shared with the member. This key will be used to encrypt the group wide data encryption key when it is sent to the member. Note that if an approach such as LKH is used at the sub-group or area level, a set of keys will need to be transmitted to the member [23, 24]. A member is considered to have joined the group only when it has received the group data encryption key.

The Iolus framework is similar to IGKMP in many respects. For example, each multicast group is divided into several sub-groups, each with their own manager (a Group Security Intermediary using Iolus terminology). Members join and leave requests are sent to the sub-group manager. However, Iolus differs from IGKMP in that its framework supports both data delivery and key management. There is no group-wide data encryption key; instead, there is a separate data encryption key for each subgroup. Each subgroup manager is responsible for re-encrypting and relaying all traffic flowing between the members of its sub-group and the other sub-groups.

Tradeoffs: The Iolus approach can scale to handle large and dynamic groups because joins and leaves within sub-group do not affect the rest of the group. However, sub-group managers are responsible for both payload delivery and key management. Under the IGKMP framework, however, key management and payload delivery are decoupled since there is a single group-wide key that is used for encrypting group traffic. This allows data packets to be routed using the best multicast communications scheme and removes the need for the packets to be re-encrypted by the sub-group manager. Having a single group-wide key, however, implies that the scalability of group re-keying is affected by the size and the membership dynamics of the whole group. If the DKD multicasts a new key to the AKDs on every membership change, the scheme will not scale to large and dynamic groups.

3.2 Our Approach

We now describe a scalable approach, which we call Kronos, for re-keying a large and dynamic group that can be used within a distributed framework such as IGKMP.

The operation of the Kronos protocol is similar to that of IGKMP as discussed in Section 3.1 with two key differences. First, the DKD or group manager is not directly involved in generating the new group traffic encryption key that is distributed by the AKDs to the existing members of the group in their area. Instead each AKD *independently* generates the *same* group-wide traffic encryption key *at the same time* and transmits it to the members in its area. Second, Kronos uses periodic re-keying to decouple the rate of re-keying from group size and membership dynamics.

Under Kronos, group re-keys are not driven by member join or leave requests. Instead, at periodic intervals, all the member join and leave requests that have accumulated at an AKD are processed and the new multicast traffic encryption key is securely transmitted to the existing members of the group. An algorithm such as LKH can be used by each AKD to accomplish this task in a scalable manner. Note that most of the processing required for joins and leaves can be done during the time interval between re-keys. Further note that under this approach a new traffic encryption key will be transmitted by an AKD to the members in its area even if there has been no membership change during the previous time period.

Two issues need to be addressed for this approach to work correctly. First, all the AKDs must use the same period for re-keying and must have their clocks synchronized so that they re-key at the same time. Second, the AKDs must share some state information that enables them to generate the same key without any communication. Further, no entity other than the AKDs should be able to generate the group key.

The first issue is addressed by having the AKDs agree in advance on the re-keying period and by using a clock synchronization algorithm such as the Network Time Protocol (NTP) [14]. NTP can synchronize hosts to within a millisecond on LANs and within a few tens of milliseconds on WANs relative to a server synchronized to Coordinated Universal Time via a Global Positioning Service (GPS) receiver. Further, NTP can be configured to use multiple redundant paths for reliability, and authentication to prevent accidental or malicious protocol attacks.

The second issue can be addressed as follows. First, all the AKDs need to agree on two shared secrets, say K and R_0. This can be accomplished by having the DKD (or the group coordinator) selecting K and R_0 and transmitting it to the AKDs using a secure channel. Alternatively, the AKDs can use a group key agreement algorithm such as Cliques[19] to generate K and R_0 in a contributory fashion. Once the shared secrets are established, every AKD generates the multicast group key, R_1, by applying a secret-key encryption algorithm, E, to R_0 using K as the secret key. Thus, $R_1 = E_K(R_0)$. R_1 is then securely transmitted to the members of the group in the AKD's area.

This process is repeated at each iteration, i.e., the AKD obtains the next multicast group key by applying the secret key encryption algorithm to the the previous group key. Thus

$$R_{i+1} = E_K(R_i), i \geq 0$$

The choice of the encryption function, E and the length of the key, K, is dictated by the security requirements of the application. Any function such as DES, triple DES, or IDEA [18] can be used. In addition, periodically, for enhanced security the AKDs should re-establish the shared secrets, K and R_0.

The choice of the re-keying period for our approach is largely dependent on the security and performance requirements of the application. However, the fact that the AKDs do not need to coordinate while re-keying enables us to select re-key periods that can be as small as one second. Using this approach, a distributed key management framework such as IGKMP can re-key large and dynamic groups in a scalable way while maintaining a single group-wide multicast key.

3.3 Discussion

Using periodic re-keying implies that join and leave latencies will be on average equal to half the the interval between re-keys plus the network delay for the join request and the reply. We assume that a join is considered complete when the user receives the current session key, and a leave is completed when the existing members of the group receive the new session key. If the interval between re-keys is large, this latency may be unacceptable for some applications. On the other hand by making the re-keying interval relatively small (of the order of seconds), this latency can be reduced to acceptable levels for most applications. We explore this issue in more detail in Section 4.

We note that periodic re-keying has long been recognized as being necessary from the security point of view [10]. This is because employing the same key for a long period of time increases the chances of that key being successfully crypto-analyzed by an attacker who has collected the ciphertext of messages encrypted with that key.

An advantage of periodic re-keying from the point of view of the application is that the overhead of re-keying is predictable and bounded. This is especially advantageous for applications executing on platforms with limited resources or for real-time applications. Another advantage of periodic re-keying is that typically several joins and/or leaves will processed at the same time. This allows the use of optimizations such as those proposed by Moyer *et al* [16] and Chang *et al* [3] that result in the size of the periodic re-key message being smaller than the sum of the sizes of the individual re-key messages transmitted under a protocol that re-keys the group on each membership change.

Variants: It is also possible to come up with variants on the basic periodic re-keying scheme. If the security requirement of perfect backward confidentiality is relaxed, then there is no need to change the current session key on a join. Thus, a user who makes a join request can be supplied with the current key right away instead of having to wait until the next re-key to join the group.

If an application distinguishes between member leaves and member ejections, and it is critical to eject a member as soon as possible, then another variant can be used in which re-keying is done either periodically or when a member is ejected from the group. Since it is reasonable to assume that ejections are less frequent than joins and leaves, this does not affect the scalability of the scheme.

Finally, we note that the re-keying interval for a group can be made adaptive so that it matches group membership dynamics. An example hybrid protocol would be one in which in times of heavy load, the Kronos approach would be used whereas at other times re-keying can be initiated by the DKD.

Trust Considerations: Kronos builds upon distributed key management frameworks such as IGKMP and therefore inherits the trust relationships assumed by these frameworks. Thus, it is assumed that each host member in an area trusts its AKD, and that all AKDs and the DKD trust each other. As such, a protocol such as Kronos may be more applicable for applications where the AKDs and DKD are all under the control of a single organization.

4 Evaluation

In this section, we use a detailed simulation to evaluate the performance tradeoffs between Kronos and other approaches for scalable key management, specifically LKH and Iolus.

4.1 Metrics

The performance metrics of interest are the following:

Join/leave latency This is the time that elapses between the submission of a join or leave request by a member and the receipt of the keying material that enables it to decrypt group communications. We assume that a user has previously established a private key with the group (or sub-group) manager using Diffie-Hellman agreement before submitting the join request. Thus, the components that make up the join/leave latency include (i) the network delay for the packets corresponding to the request and the reply (ii) the delay at the server corresponding to the computation time for receiving and authenticating the request, generating and encrypting the new keys, creating the message digest, signing the response, and transmitting it to the member (iii) the queueing delay at the server (iv) delay due to lost requests or responses.

Time between Rekeys As discussed in Section 2, the shorter the time between the rekeys, the higher the network overhead in terms of both the number of messages and the bandwidth consumed. In addition, the computational overhead at the member hosts for receiving and decrypting new keys increases with decreasing re-key times.

Data packet latency This is the average network delay before a data packet transmitted by the source reaches the members of the group. For the LKH and Kronos protocols, multicast data delivery is independent of the keying protocol. However, under Iolus, each packet sent by the source is re-encrypted and relayed by the sub-group manager to its subgroup. The extra delay for these actions is reflected in data packet latency.

4.2 The Simulation Environment

Network Model: Our simulations were written using the packet-level, event-based network simulator ns2 [25] from UC, Berkeley. We used the Tiers network topology generator [6] to generate the topologies used in our simulation. Tiers generates 3-level hierarchical networks consisting of WANs, MANs, and LANs. While we ran simulations for several different topologies, the base network topology for which results are reported in this section consisted of 360 nodes distributed over a WAN corresponding to the backbone, 10 MANs, and 50 LANs. The WAN backbone has 10 routing sites and each MAN also has 10 routing sites, while each LAN has 5 hosts. The average degree of redundancy (extra edges between nodes) for both WAN and MAN routers is 2. The bandwidth of the WAN, MAN, and LAN links are assumed to be 2248 Mb/s, 155 Mb/s and 100 Mb/sec respectively. The average link propogation delay is approximately 60 ms for WAN links, 17 ms for MAN links and 1 ms for LANs.

In order to simulate large topologies in a reasonable amount of time, we we only simulated traffic flows corresponding to the multicast application under consideration and the control traffic for group key management. To model the effect of background traffic and queueing, on each hop each packet experiences a random delay drawn from a uniform distribution between 0 and 2 ms. Further, each link has an associated loss rate, l_i. Thus each packet traversing the link gets dropped with a probability l_i. While we considered several different loss models corresponding to different network conditions, in the loss model used for the results reported in this section, higher packet losses occur in some of the MANs. Specifically, 30% of the MAN and MAN-WAN links have a loss probability of 2% while the remaining links have a loss probability of 0.5%. In the simulation, routers run dense mode multicast routing similar to DVMRP [7].

Workload Model: We considered two multicast scenarios: a one-to-many scenario and a many-to-many scenario. For both scenarios, there were 240 potential multicast group members (the number of subscribers, N, using the terminology introduced in Section 2) distributed among the LAN-level nodes of the network. As in Section 2, each subscriber independently alternates between a state in which it is member of the multicast group corresponding to the application and a state in which it is not a member. In our simulations, we assumed that the time spent in these states is exponentially disitributed. Further, we assumed that R, the ratio of the time spent in these states was 1, while the time for which a host is a member of the group $(1/\mu)$ was varied from 5 seconds to 12.5 seconds. Our intent was to simulate situations in which there were a large number of join and leave requests arriving per unit time at the group manager, i.e., situations in which the frequency of group re-keying would become a bottleneck for protocols which re-keyed the group on every membership change. As discussed in Section 2, the frequency of re-keying depends upon both group membership dynamics (as determined by R and $1/\mu$) as well as the size of the subscriber population (N). Since simulating large groups with thousands of members is not feasible[2], we achieved our goal of high rates of join and leave requests by selecting relatively small (and unrealistic) values of $1/\mu$ and R.

Each simulation run corresponds to 70 seconds of simulated time. Statistics are gathered only after the first 10 seconds of the simulated time have elapsed to remove cold-start effects. In the one-to-many scenario, a single CBR source multicasts data to the group at 224 Kb/sec. The CBR source starts transmitting data after 0.6 seconds in the simulation. In the many-to-many scenario, the fraction of the members that are senders is varied from 10% to 40%. These senders are uniformly distributed over the network topology. Each member that is a sender alternates between a state in which it is a CBR sender (sending state) and a state in which it does not transmit any data (quiet state). The time each sender spends in both the “sending state” and the “quiet state” is uniformly distributed between 0 and 12.5 seconds.

In our simulations for LKH, there is a single group manager that is co-located with the data source in the one-to-many scenario. Separate multicast addresses are used for data and control traffic. The logical keytree has a degree of 4 and key-oriented keying [24] is used. In our implementation of LKH, if multiple join and leave requests are present in the request queue, they are processed as a batch while re-keying the logical keytree. For Iolus and Kronos, the network is divided into “areas” each with their own AKD or

[2] Each run of our simulations for the network and workload models described in this paper took more than one day on a PC with a Pentium III 600 MHz CPU and required between 200 and 400 MB of RAM

sub-group manager. Each sub-group manager (AKD) uses the LKH algorithm for key management in its sub-group (area). For Iolus, the re-encryption done at the sub-group manager for each data packet simply involves decrypting and re-encrypting the message key that is associated with each message [15] (and not the entire payload in the packet).

For Kronos (Iolus), the location of the AKDs (sub-group managers) affects the join and leave latency for the hosts in that area. In the case of Iolus, the latency of the multicast data is also affected by the location of the sub-group managers. We consider two cases in our simulations. In the first case, which we label as the “dense” mapping, each MAN is considered an “area”; thus, there are 10 areas each with an AKD that performs the key management for 24 hosts in the area. In the second case, which we label as the “sparse” mapping, the network topology is mapped into 8 areas each with 30 hosts. While most of the hosts in an area are located in the same MAN as their AKD, some of the hosts are in “nearby” MANS.

Finally, for Kronos, we assume that an algorithm such as NTP is being used to keep the clocks of the AKDs synchronized to within 25 ms. Thus, in our simulations, the clock skew between the AKDs is uniformly distributed between 0 and 25 ms. The costs assumed in our simulations for the various tasks performed by the LKH, Iolus, and Kronos are listed in Table 4.2. These costs assumed are based on results reported in the literature [3, 18].

Encryption algorithm	DES3
Key Length	168 bits
Key Encryption Time	1.64 msec
Key Generation Time	2.8 msec
Data packet size	500 bytes
Signature algorithm	RSA
Signature Length	512 bits
Authentication rate	1506 KB/sec
Signature rate	367 KB/sec

Table 1. The costs used in our simulations for the LKH, Iolus and Kronos protocols. The signing and authentication rates include the time for taking a MD5 hash.

Reliable key delivery: While UDP is used for both data and key messages, we use a simple receiver-initiated protocol for reliable-key delivery. Each message contains a clear text field that indicates the key version number used to encrypt the message. Group members use the version number field of each message to detect if they are missing some keys. If this is the case, a request for the missing keys is sent to the key manager. Each group member maintains an adaptive timer for detecting if its messages to its AKD have been lost. This timer is set based on the round-trip-time between the member and its key manager.

4.3 Results

Unless stated otherwise, the results discussed below use the network and workload models described in Section 4.2. The results for Iolus and Kronos assume the “sparse” mapping of AKDs on the network topology. The re-key interval used for Kronos is 1 second.

Join and Leave Latency: Our first set of results compares the performance of LKH, Iolus and Kronos for the one-to-many scenario described above. In Figure 2, we plot the join and leave latencies seen by the members of the group for these protocols as a function of the average time for which a user joins the group ($1/\mu$). We can make three observations from Figure 2. First, LKH has higher join and leave latencies than Iolus for all values of $1/\mu$. Second, the join (leave) latencies for Iolus and Kronos are not affected by changing $1/\mu$. Third, the join (leave) latencies for LKH increase dramatically as $1/\mu$ is decreased below 10 seconds.

The join and leave latency for a member depends on two components: the network delay for the request and reply and the delay at the key manager. While the network delay for the request and reply largely depends upon the network “distance” between the member and the key manager, the delay at the key manager depends on the load on the key manager. Recall from Section 2 that the load on the key manager depends upon the number of subscribers (N) and the group membership dynamics (as determined by $1/\mu$ and R). For a fixed R, the load on the manager increases as $1/\mu$ decreases. In the case of LKH, Figure 2 shows that the key manager becomes overloaded when $1/\mu$ is decreased below 10 seconds. Thus the major component of the join and leave latency under LKH is the queuing delay at the key manager.

In the case of Iolus and Kronos, the task of re-keying is distributed among the AKDs and sub-group managers. In our experiments, varying $1/\mu$ between 5 and 12.5 seconds did not impose a heavy load on the AKDs, and thus did have any noticeable impact on the join and leave latency. For Iolus, the major component of the join latency is the network delay for the request and the manager’s response. In the case of Kronos, the major component of the join (leave) latency is the queueing time at the AKD, which is on average equal to half the fixed re-keying period - in this case 0.5 seconds.

Data Latency: Figure 3 plots the average latency for the data packets multicast by the CBR source in the one-to-many scenario. In the case of LKH and Kronos, data delivery is independent of the group key management protocol. Thus, the data latency is not affected by changing $1/\mu$. On the other hand, in the case of Iolus, the CBR source first

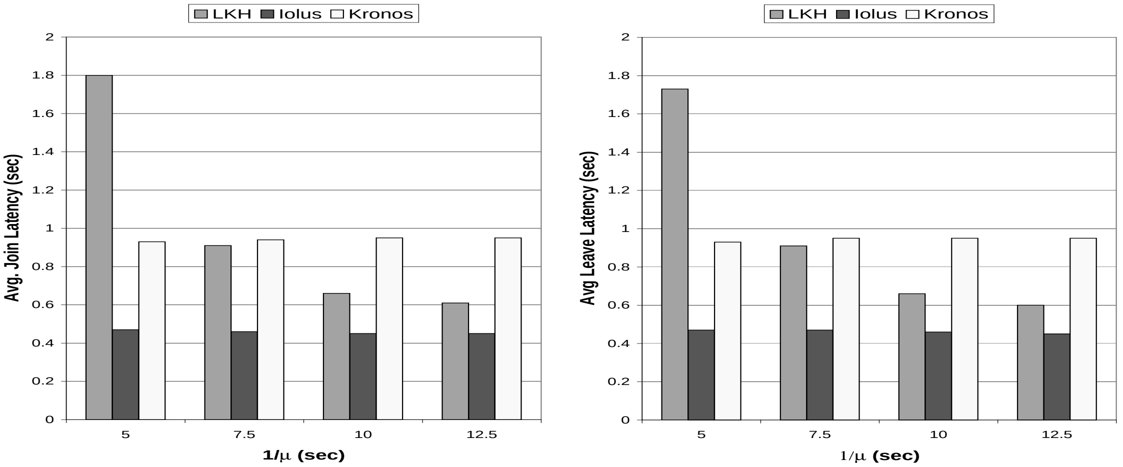


Figure 2. Average member join and leave latencies under LKH, Iolus, and Kronos as a function of $1/\mu$.

multicasts the data to the sub-group managers who are responsible for re-encrypting it and forwarding it to the members of their respective sub-groups. Figure 3 shows that the data latency under Iolus is significantly larger than under LKH and Kronos. Note that the higher data latency under Iolus relative to LKH and Kronos is *not* because of the cost of re-encryption which is less than 2 ms per packet. Instead, the difference in data latency is because of the extra network delay in routing each packet to its destination via the sub-group manager. Further note that the results in Figure 3 were obtained for the "sparse" mapping of sub-group managers on the network topology. Later in this section we examine the effect of changing the location of the sub-group managers on the data latency for Iolus.

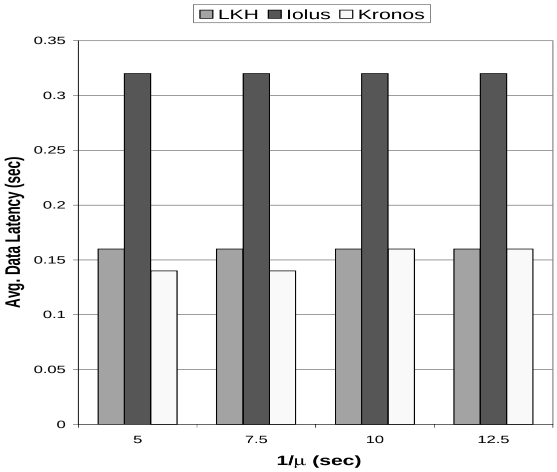


Figure 3. Average data latencies for LKH, Kronos, and Iolus as a function of $1/\mu$

We also examined the average data latency for the protocols for the many-to-many scenario described in Section 4.2. In contrast to the one-to-many scenario, in the case of the many-to-many scenario, the encryption bandwidth available at the Iolus sub-group manager has a significant impact on the average data latency. For the base parameters listed in Table 4.2, the Iolus sub-group manager is capable of re-encrypting approximately 610 packets per second[3]. In our many-to-many scenario, the number of packets that are delivered to the sub-group manager is much larger than this. As a result, each packet experiences long queueing delays at the sub-group manager leading to high data latencies[4].

In Figure 4, we plot the average data latency under Kronos and Iolus for the many-to-many scenario as a function of the fraction of group members who are senders. For Iolus, we consider the effect of doubling and quadrupling the encryption bandwidth at the subgroup managers. We observe that changing the fraction of senders has no impact on the data latency under Kronos. (Although not shown in the figure, the same is true for LKH.) For Iolus, it is clear that the encryption bandwidth becomes a bottleneck as the number of senders increases. Quadrupling the processing power of the sub-group manager results in the data latencies under Iolus being comparable to those of Kronos when the fraction of group members that are senders is below 30%. To

[3] Note that for each packet only a single key is re-encrypted, not the entire payload of the packet

[4] We assume that the sub-group manager has sufficient buffer space for these queued packets.

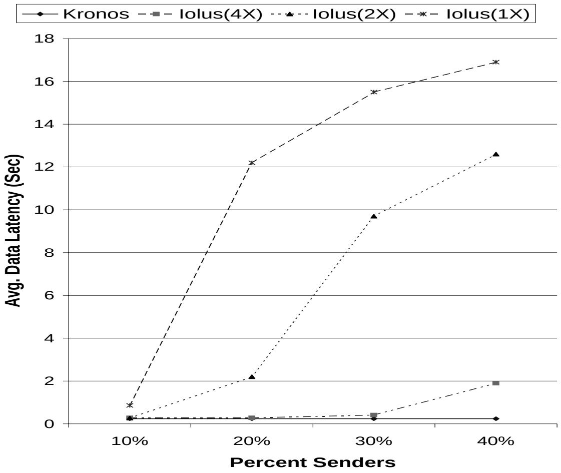


Figure 4. Avg. data latency under the Many-to-Many multicast scenario for Kronos and Iolus as a function of the percentage of group members that are senders. Note that Iolus(NX) represents the case where the encryption bandwidth of each sub-group managers is N times the baseline encryption bandwidth.

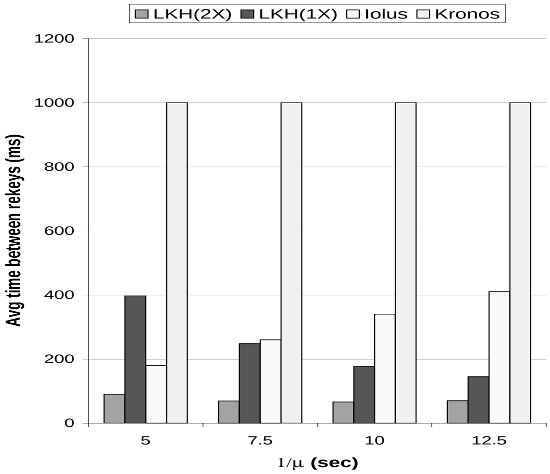


Figure 5. Avg. time between re-keys for the various policies. LKH(2X) represents the case where the processing power at the key manager is doubled.

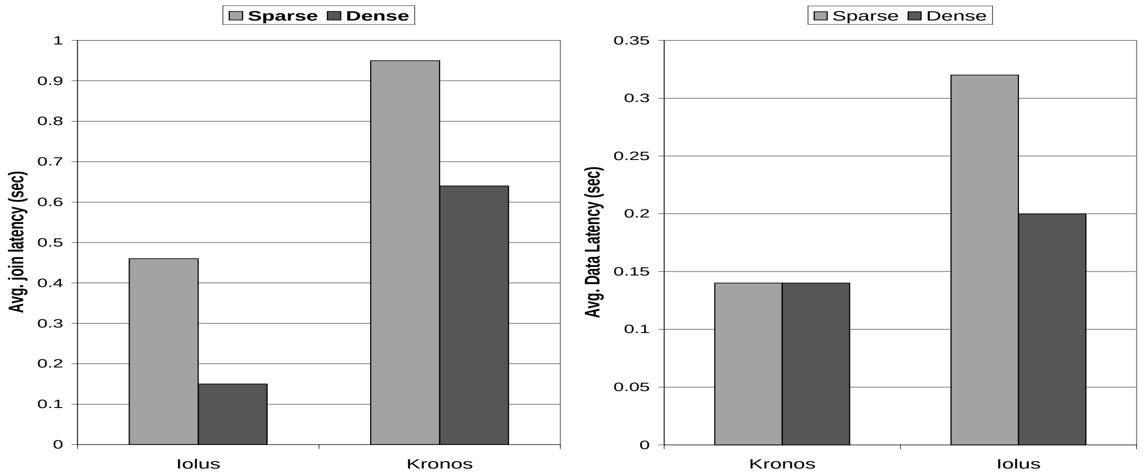
obtain comparable data latencies for the 40% percent case, it was necessary to increase the encryption bandwidth at the sub-group managers by a factor of 5.

Time between Group Re-keys: Figure 5 shows the impact of changing $1/\mu$ on the time interval between group re-keys. For Kronos, the interval between re-keys is fixed *a priori* so changing $1/\mu$ has no effect. For Iolus, decreasing $1/\mu$ increases the rate at which join and leave requests arrive at the sub-group key managers, thus increasing the frequency of sub-group re-keying. Thus, we see that the time between re-keys decreases from 400 milliseconds when $1/\mu = 12.5$ seconds to around 190 milliseconds when $1/\mu$ = 5 seconds.

For LKH, the re-key period increases as $1/\mu$ decreases. While this may seem counter-intuitive, it can be explained by the fact that (as discussed above) the key manager becomes overloaded as $1/\mu$ is decreased. Consequently, long queues of join and leave requests build up at the key manager. In our simulation implementation of LKH, the key manager processes all the join and leave requests present in the queue as a single batch. Thus, the number of re-keys is not as large as would be the case if each and every join or leave request resulted in a group re-key. Figure 5 also shows the average re-key period for LKH if the processing power at the key manager is doubled. In this case, we see that the re-key period for LKH is much smaller; it decreases from around 90 ms to 60 ms as $1/\mu$ decreases from 12.5 to 5 seconds.

Impact of increasing processing power for LKH: As the results above show, for LKH the key manager becomes a bottleneck as the rate of join and leave requests increases. Increasing the processing power at the key manager will obviously remove this bottleneck. Figure 6 shows that when the processing power at the key manager is doubled (relative to the baseline parameters listed in Table 4.2), the join and leave latencies are significantly reduced. However, we note that increasing the processing power also results in a reduction in the time between re-keys as shown in Figure 5. Further note that for the network topology considered in our simulations, the average latency for keys to distributed to the members of the group is around 150 msec which is larger than the re-key period (90 msec). This implies that the group key is changed by the key manager even before all members have received the previous key. These results show that for protocols that re-key the group on every membership change the frequency of group re-keying imposes limits on the scalability that are independent of the processing power available at the key manager.

Impact of manager location on Iolus and Kronos: The results reported above for Iolus and Kronos are for the "sparse" mapping of sub-group managers described in Section 4.2. In the case of the "dense" mapping, each manager is located in the same MAN as its members so that messages between members and their key managers never traverse the network backbone. Figures 7 (a) and (b) compare the join and data latencies for the dense and sparse mappings under



(a) (b)

Figure 7. The Impact of Manager Location on Iolus and Kronos (a) Join latency (b) Data Latency

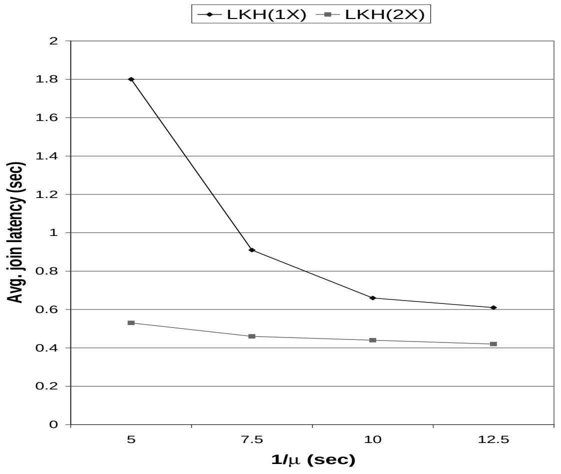


Figure 6. The impact of increasing the processing power at the key manager on LKH performance. LKH(2X) represents the case where the processing power at the key manager is doubled.

Kronos and Iolus. We observe that the average join latencies for both Iolus and Kronos are significantly reduced (by around 300 ms) for the dense mapping. In the case of Iolus, the data latency is also reduced (by around 120 ms) for the dense mapping. On the other hand, for Kronos, the AKDs are not involved in data delivery so their location has no impact on the data latency.

Impact of changing the re-key period for Kronos: Our last set of experiments examines the impact of changing the re-key time period on the join and leave latencies under Kronos. In Figure 8, we plot the avg. join latency for the "dense" manager mapping for re-key periods of 0.5, 1, and 1.5 seconds. As expected, the average join latency increases with the re-key period.

4.3.1 Summary of Results

From the results above, we can draw the following conclusions:

- The increase in the frequency of re-keying with increasing join and leave rates imposes limits on the scalability of LKH that are independent of the processing power available at the key manager.

- For most scenarios, Iolus results in the lowest join and leave latencies among the three protocols. However, Iolus has the highest data packet delivery latencies among the three protocols because of the need for re-encryption and re-transmission. The location of the Iolus sub-group manager has a significant impact on the data latency. Further, in the case of the many-to-many scenario, the encryption bandwidth of the sub-group managers become a bottleneck as the number of senders in a group and the rate at which they are transmitting data increases.

- The join and leave latencies under Kronos depend upon the re-keying period. By selecting a re-key period of 1 second, we can obtain join and leave latencies that are comparable to those of Iolus. Further, Kronos has the attractive properties that the data latencies are independent of the location of the AKDs, and that the

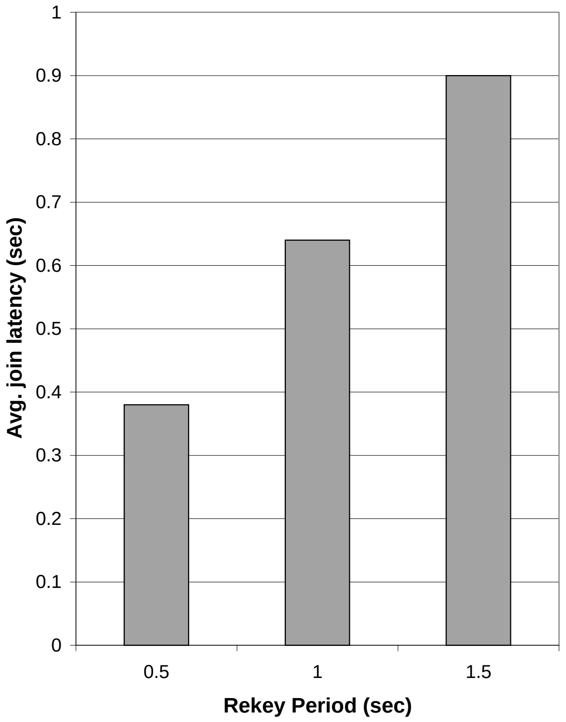


Figure 8. The effect of changing the re-key period on the join latency under Kronos.

group re-keying rate is independent of group size and membership dynamics.

5 Conclusions

In this paper, we have described Kronos, a novel approach to scalable group re-keying for secure multicast. We showed that if a multicast group is re-keyed on each membership change, as the size of the group increases and/or the rate at which members leave and join the group increases, the frequency of re-keying becomes the primary bottleneck for scalable group re-keying. In contrast, Kronos is based on periodic re-keying which decouples the frequency of re-keying from the size and membership dynamics of the group. Another feature of Kronos is that it can be used in conjunction with a distributed framework for key management such as IGKMP [10] that uses a single group-wide session key for encrypting communications between members of the group.

Using a detailed simulation, we examined the performance tradeoffs between Kronos, Iolus, and LKH. Our results indicate that the join and leave latencies obtained for Kronos are acceptable for most applications. Further, the average data latencies for the multicast application are lower than those obtained under Iolus. Finally, the network and processing overheads for group re-keying under Kronos are predictable and lower than those obtained under Iolus and LKH.

Acknowledgements

We would like to thank the anonymous referees for their comments and advice on improving this paper. Thanks also to Bob Briscoe for his comments on this work.

References

[1] Y. Amir, L. Moser,P. Melliar-Smith, D. Agrawal, and P. Ciarfella. The Totem single-ring ordering and membership protocol. *ACM Transaction on Computer Systems*, 13(1996), no. 4.

[2] D. Balenson, D. McGrew, and A. Sherman. Key Management for Large Dynamic Groups: One-way Function Trees and Amortized Initialization. IETF Draft, draft-balenson-groupkeymgmt-oft-00.txt, February 1999.

[3] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, and D. Saha. A Toolkit for Secure Internet Multicast. Manuscript, 1998.

[4] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor,and B. Pinkas. Multicast Security: A Taxonomy and Efficient Authentication. *Proc. of INFOCOM'99*, March 1999.

[5] R. Canetti and B. Pinkas. A Taxonomy of Multicast Security Issues. Internet Draft, draft-irtf-smug-taxonomy-01.txt, April 1999.

[6] M. Doar. A Better Model for Generating Test Networks. *Proc. of Global Internet*, IEEE, November 1996.

[7] S. Deering and D. Cheriton. Multicast Routing in Datagram Internetworks and Extended LANs. *ACM Transactions on Computer Systems*, May 1990.

[8] W. Fenner. Internet Group Management Protocol, Version 2. RFC 2236, Xerox PARC, November 1997.

[9] T. Hardjono, B. Cain, and N. Doraswamy. A Framework for Group Key Management for Multicast Security. Internet Draft, draft-ietf-ipsec-gkmframework-00.txt, July 1998.

[10] T. Hardjono, B. Cain, and I. Monga. Intra-Domain Group Key Management Protocol. Internet Draft, draft-ietf-ipsec-intragkm-00.txt, November 1998.

[11] H. Harney, and C. Muckenhirn. Group Key Management Protocol (GKMP) Architecture. RFC 2094, July 1997.

[12] H. Harney and E. Harder. Logical Key Hierarchy Protocol Internet Draft, draft-harney-sparta-lkhp-sec-00.txt, March 1999.

[13] S. Kent, and R. Atkinson. Security Architecture for the Internet Protocol. Internet Draft, draft-ietf-ipsec-arch-sec-07.txt, July 1998.

[14] D.L. Mills. Network Time Protocol (version 3) Specification and Implementation. RFC1305, March 1992.

[15] S. Mittra. Iolus: A framework Scalable Secure Multicast. *Proceedings of ACM SIGCOMM'97*, Cannes, France, September 1997.

[16] M.J. Moyer, J.R. Rao, and P. Rohtagi. Maintaining Balanced Key Trees for Secure Multicast. Internet Draft, draft-irtf-smug-key-tree-balance-00.txt, June 1999.

[17] V. Paxson and S. Floyd. Wide Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking*, June 1995.

[18] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, 2nd edition, December 1995.

[19] M. Steiner, G. Tsudik, M. Waidner. CLIQUES: A New Approach to Group Key Agreement. *Proceeding of ICDCS'98*, May 1998 Amsterdam, The Netherlands.

[20] Andrew S. Tanenbaum. *Computer Networks*. Prentice-Hall, March 1996.

[21] Kishore Trivedi. *Probabilty and Statistics with Reliability, Queueing, and Computer Science Applications.* Prentice-Hall, 1982.

[22] N. Waldvogel, G. Caronni, D. Sun, N. Weiler, B. Plattner. The VersaKey Framework: Versatile Group Key Management. Department of Computer Engineering, ETH Zurich. Technical Report TIK-57, 1998.

[23] D.M. Wallner, E.G. Harder and R.C. Agee. Key Management for Multicast: Issues and Architecture. Internet Draft, draft-wallner-key-arch-01.txt, September 1998.

[24] C.K. Wong, M. Gouda, S.S. Lam. Secure Group Communication Using Key Graphs. *Proc. of SIGCOMM '98*, 1998.

[25] UCB/LBNL/VINT Network Simulator, ns version 2. http://www-mash.CS.Berkeley.EDU/ns/ns.html.

A Average Time between Rekeys

We derive the expected time between re-keys for the Independent Subscriber Behaviour case discussed in Section 2.

Assume that the time for which a subscriber is tuned in to a multicast is exponentially distributed with mean $1/\mu$. Further, assume that the time during which a subscriber is *not tuned in* to the multicast is exponentially distributed with mean $1/\lambda = R/\mu$. Thus R is the ratio of the average time for which a subscriber is tuned out to the average time for which a subscriber is tuned in.

Under these assumptions, the number of subscribers that are tuned in to a multicast can be modeled by a birth-death Markov process [21]. Consider a state in which there are i subscribers tuned in to the multicast. Thus, there are $N - i$ subscribers that are tuned out. The transitions into state i can occur from state $i - 1$ at a rate $(N - i + 1)\lambda$ and from state $i+1$ at a rate $(N-i-1)\mu$. The transitions out of state i occur at a rate $i\mu+(N-i)\lambda$. Solving the balance equations for this Markov chain, we can obtain the probability of each state i.

Since the group is re-keyed each time there is a membership change, the rate at which transitions occur out of a given state i is the rate at which the group will be re-keyed. Thus, the expected time between re-keys is given by

$$T_{rekey} = \sum_{i=0}^{N} (i\mu + (N - i)\lambda)p_i$$

where p_i is the probability of being in state i. After some straightforward algebraic manipulation, we find that

$$T_{rekey} = (1 + R)/(2\mu N)$$