GKMPAN: An Efficient Group Rekeying Scheme for Secure Multicast in Ad-Hoc Networks *

Sencun Zhu† Sanjeev Setia‡ Shouhuai Xu§ Sushil Jajodia¶
Email:szhu@cse.psu.edu,{setia,jajodia}@gmu.edu,shxu@cs.utsa.edu

Abstract

We present GKMPAN, an efficient and scalable group rekeying protocol for secure multicast in ad hoc networks. Our protocol exploits the property of ad hoc networks that each member of a group is both a host and a router, and distributes the group key to member nodes via a secure hop-by-hop propagation scheme. A probabilistic scheme based on pre-deployed symmetric keys is used for implementing secure channels between members for group key distribution. GKMPAN also includes a novel distributed scheme for efficiently updating the pre-deployed keys. GKMPAN has three attractive properties. First, it is significantly more efficient than group rekeying schemes that were adapted from those proposed for wired networks. Second, GKMPAN has the property of *partial statelessness*; that is, a node can decode the current group key even if it has missed a certain number of previous group rekeying operations. This makes it very attractive for ad hoc networks where nodes may lose packets due to transmission link errors or temporary network partitions. Third, in GKMPAN the key server does not need any information about the topology of the ad hoc network or the geographic location of the members of the group. We study the security and performance of GKMPAN through detailed analysis and simulation; we have also implemented GKMPAN in a sensor network testbed.

1 Introduction

Many applications of ad hoc networks involve collaborative computing among a large number of nodes and are thus group-oriented in nature. Examples of such applications include coordination of fire fighters in a rescue task or coordination of soldiers during a battle. For deploying such applications in an adversarial environment such as a battlefield or even in many civilian commercial scenarios, it is necessary to provide support for secure group communication. In this paper, we address the issue of providing confidentiality for group communication in ad hoc networks.

The most efficient approach for achieving confidential group communication is to use a symmetric group key that is shared by all the nodes for data encryption. This approach however introduces the problem of *group rekeying*, i.e., the group key must be updated and redistributed to all the remaining nodes in a secure, reliable, and timely fashion when group membership changes. This problem has been studied extensively in the context of secure multicast in wired networks and several scalable key management protocols have been proposed, e.g., OFT [3], Subset-Difference [18], LKH [27], and ELK [22]. However, these approaches are

*A preliminary version of this work appeared in Proceedings of the 1st International Conference on Mobile and Ubiquitous Systems (Mobiquitous'04), 2004.

†Sencun Zhu is with both the Department of Computer Science and Engineering and the School of Information Science and Technology, The Pennsylvania State University.

‡Sanjeev Setia is with the Department of Computer Science, George Mason University.

§Shouhuai Xu is with Department of Computer Science, University of Texas at San Antonio

¶Sushil Jajodia is with the Center for Secure Information Systems, George Mason University.

not directly applicable to ad hoc networks, because the communication cost per node can become very high for a large ad hoc network with very dynamic group membership. So far very few group rekeying schemes have been proposed for ad hoc networks. They either use public-key techniques [11], or adapt the LKH scheme for ad hoc networks [16]. Public-key based schemes [11] are more expensive than symmetric-key based schemes in both communication and computation. The adapted LKH scheme [16] incurs the computational and communication cost that is of the same order as the LKH scheme [27]. Moreover, the LKH-based schemes [3, 16, 27] have the disadvantage of *stateful* group rekeying schemes that a node that has missed a group rekeying operation will need to obtain previously transmitted key encryption keys in order to be able to decrypt the current group key. This may involve requesting the key server for retransmission of any missing key encryption keys, which is highly undesirable in a multi-hop wireless network.

In this paper, we present a scalable and efficient group rekeying protocol (GKMPAN) for ad hoc networks based solely on symmetric key techniques. GKMPAN exploits the property of an ad-hoc network that member nodes are both hosts and routers. In IP Multicast, all group members are end hosts, and they have no responsibility for forwarding keying materials to other group members. In contrast, for group communication in an ad hoc network, the members of the group also act as routers. As such, in GKMPAN the key server only has to deliver the new group key securely to the group members that are its immediate neighbors, and these neighbors then forward the new group key securely to their own neighboring members. In this way, a group key can be propagated to all the members. Because every node only needs to receive one encryption of the group key, the average transmission cost per node is one key independent of the group size.

For the above scheme to work, a fundamental requirement is the existence of a secure channel between every pair of neighboring nodes. GKMPAN provides secure channels through probabilistic key predeployment. The technique of probabilistic key pre-deployment has been applied in several studies [6, 8, 33]; however, to the best of our knowledge, none of these studies address the issue of updating the pre-deployed keys. Updating the predeployed keys is critical in order to prevent the compromised and revoked nodes from launching a collusive attack in which they pool together their keys with the goal of jeopardizing the secure channels between other nodes. Without key updating, both the performance and security of the system will degrade greatly with the number of compromised nodes. To address this issue, we present an efficient distributed key updating scheme for updating any compromised channels.

The contributions of this paper are two-fold. First, GKMPAN, which based on symmetric key techniques, is significantly more communication-efficient than previous approaches [16, 27] for group rekeying when used for ad hoc networks. It also has the property of *partial statelessness*; that is, a node can decode the current group key even if it has missed a certain number of previous group rekeying operations. This makes it very attractive for ad hoc networks where nodes may lose packets due to transmission link errors or temporary network partitions. Moreover, unlike the LKH-based scheme in [16], GKMPAN does not require any information about the topology of the ad hoc network or the geographic location of the members of the group.

Second, the key update scheme of GKMPAN can also be used to increase the robustness of other probabilistic key pre-deployment based schemes [6, 8, 33] to more node compromises when compromised nodes can be detected.

The rest of this paper is organized as follows. An overview and the protocol details of GKMPAN as well as security analysis are presented in Section 2. We analyze and evaluate its performance in Section 3. The related work are described in Section 4. Finally, Section 5 concludes this paper.

2 The Group Rekeying Protocol

In this section, we describe three different schemes for group rekeying – a basic scheme, and two extensions of the basic scheme that result in improved security and performance. We first discuss our assumptions and present a brief overview of our protocol, then describe its operation in greater details.

2.1 Network and Security Assumptions

Network Assumptions The communication model we consider is group-oriented communication; that is, messages are addressed to all the members. For the ease of presentation, in this section, we assume that all nodes in an ad hoc network are members of a group. In Section 2.5 we discuss how this scheme can be extended for networks where not all nodes are members of a group. For secure group communication, a group-wide symmetric key is used to encrypt group broadcast messages. Note that using pairwise shared keys for securing group communication does not improve security in comparison to a scheme based on group keys. This is because under both schemes an adversary only needs to compromise one node to obtain the group data; moreover, if pairwise keys are used for securing group data, a node will have to perform decryptions and re-encryptions for the data packets it is forwarding. Nevertheless, if the network needs to provide pairwise keys for private communication between pairs of nodes, we can directly employ the probabilistic pairwise key establishment scheme in [33] without making any additional security and network assumptions.

We assume that the resources of a node, such as power, computational and communication capacity, and storage are relatively constrained; thus a node neither can afford public-key operations nor has space for storing pre-deployed pairwise shared keys for all the nodes in the network. However, we assume that every node has space for storing hundreds of bytes or a few kilobytes of keying materials, depending on the security requirements. One type of such nodes is the current generation of sensor nodes (e.g., Berkeley Mica2 motes [29] with 8MHZ CPU and 4K RAM).

Security Assumptions and Attack Models We assume that there is a group manager (or multiple collaborative managers for fault tolerance and compromise resilience [12, 31]) managing the group membership. Under our protocol, a group rekeying is initiated by the group manager (or called key server hereafter) to revoke one or multiple nodes. We do not specify the cause (e.g., policy change, owner compromises) for node revocation. In particular, we do not assume that the reason for node revocation must be node misbehavior (e.g, injecting spurious packets). Unlike in sensor networks where sensor nodes are often unattended, in a mobile ad hoc network, it is more common that nodes are carried by other entities (e.g., soldiers, vehicles). Therefore, a node revocation is often the result of revoking the carrier of a node. For example, if a solider is captured by the adversary or is missing in a battlefield, other soldiers can report the event to the group manager, which initiates a group rekeying operation to revoke the node carried by this solider.

We do not distinguish between an attacker and a compromised node, because we assume that an attacker can obtain all the information stored in a compromised node. We assume, however, that a non-compromised node can be trusted; that is, a node executes the protocol correctly unless it has been compromised. Since wireless communication is broadcast-based, we assume that an adversary can eavesdrop on all traffic, inject packets, and replay older packets. Since we assume that an adversary can take full control of compromised nodes, an adversary may command compromised nodes to drop or alter messages they are forwarding.

2.2 Design Goal

Given the threat model described above, in this paper, we focus on preventing a *group key recovery attack* in which an attacker's goal is to learn the group key through eavesdropping on key distribution messages

exchanged by group members. Several nodes whose group membership has been revoked can collude in this attack by pooling together their keys. Our goal is to design a bandwidth-efficient group rekeying scheme that updates the compromised keys with little performance overhead once the compromised nodes are detected. The scheme should enable the non-compromised nodes to reject spurious group keys injected by compromised nodes. The scheme should also be robust to refusal-of-service attacks in which compromised nodes prevent other nodes from receiving group keys by dropping packets going through them.

2.3 Protocol Overview

Our group rekeying protocol involves a *key pre-distribution* phase and a *rekeying* phase.

Key Pre-distribution Prior to the deployment of the ad hoc network, all nodes obtain a distinct subset of keys out of a large key pool from the key server and these keys are used as key encryption keys (KEKs) for delivering group keys.

A rekeying operation itself involves three steps: authenticated revocation notification, secure group key distribution and key updating.

Authenticated Node Revocation When the key server decides to revoke a node, it broadcasts a revocation notice to the network in an authenticated way.

Secure Key Distribution The key server generates and distributes a new group key K. The key K is propagated to all the remaining nodes in a hop-by-hop fashion, secured with the non-compromised predeployed keys as KEKs.

Key Updating After a node receives and verifies the group key K, it updates its own KEKs based on K.

2.4 Schemes For Group Rekeying

Notation Below are the notations that appear in the rest of this discussion.

- u, v (in lower case) are principals such as communicating nodes.
- R_u is a set of keys that u possesses, and I_u is the set of key ids corresponding to the keys in R_u.
- I_C is the set of ids of the compromised keys known to the revoked nodes.
- I_P is the set of ids of all the keys in the key pool P.
- $\{f_k\}$ is a family of pseudo-random functions [9].
- $\{s\}_k$ means encrypting message s with key k.
- $MAC(k, s)$ is the MAC of message s using a symmetric key k.

2.4.1 Scheme I: The Basic Scheme

Key Pre-distribution Each node is loaded with the following information:

1. Each node u is loaded with m distinct keys from the key pool P of l keys $\{k_1, k_2, \ldots, k_l\}$, and these keys are used as KEKs. A deterministic algorithm is used to decide the subset of keys R_u allocated to node u. Specifically, for each node, the algorithm generates m distinct integers between 1 and l using a uniform pseudo-random number generator upon the input of a node id. These integers are the ids of the keys for the node and the node is loaded with the keys indexed by these ids. As a result, each key in the key pool has a probability of m/l to be chosen by each node. Note that this construction allows any node that knows another node's id u to determine I_u, the ids of the keys held by u.
2. Each node is loaded with the initial group key k_g that is used for securing group-wide communications, and an individual key that is only shared between the node and the key server.

3. Finally, each node is loaded with the commitment (i.e., the first key) of the key chain of the key server because we are employing TESLA [20] for broadcast authentication.

The above key predistribution technique is different from that in the previous work [8, 6]. In our scheme, a node only needs to know another node's id to be able to determine the keys it shares (if any) with that node. Since the id of a neighbor is freely available from either the medium access control layer [10] or the network layer [23], no additional cost is needed for discovering the shared keys. In contrast, the approach in [8, 6] requires each node to exchange the ids of the keys it possesses with its neighbors. This is not scalable because in a mobile ad hoc network the neighbors of a node may change frequently. To this end, our approach is much more bandwidth-efficient and scalable.

Note that this key pre-distribution phase is equivalent to the member joining phase in traditional secure IP multicast. We will discuss node additions in more detail in Section 2.5. In this paper, we are mainly concerned about group rekeying due to node revocations.

Authenticated Node Revocation To revoke a node, the key server broadcasts a notification to the network to initiate a group rekeying. The notification must be authenticated so that compromised nodes cannot revoke a legitimate node or spread malicious packets that could lead to inconsistency in our schemes.

We employ TESLA [20] for broadcast authentication in our protocol due to its efficiency and tolerance to packet loss. TESLA is based on the use of a one-way key chain along with delayed key disclosure. To use TESLA, we assume that all the nodes and the key server are loosely time synchronized, i.e., a node knows the upper bound on the time synchronization error with the key server.

Let u be the node being revoked and k_i^T be the to-be-disclosed TESLA key. The key server updates $I_C = I_C \cup I_u$ (originally $I_C = \emptyset$). Let M be the id of the non-compromised key (i.e. keys in $I_P - I_C$) that is possessed by the maximum number of remaining nodes in the network. The key server generates the new group key as follows: $k'_g = f_{k_M}(k_g)$, then broadcasts the following message

$$KeyServer \longrightarrow * : u, M, f_{k'_g}(0), MAC(k_i^T, u|M|f_{k'_g}(0)).$$

We refer to $f_{k'_g}(0)$ as the *verification* key because it enables a node to verify the authenticity of the group key k'_g that it will receive later. The key server distributes the MAC key k_i^T after one TESLA interval[1]. A node receiving the above node revocation message and the MAC key K_i^T verifies the message using TESLA. It will store the verification key $f_{k'_g}(0)$ temporarily if the verification is successful.

Secure Key Distribution When a node u is revoked, all the keys it possesses, including k_g and the keys in R_u, must either be changed or discarded to prevent it from accessing future group communications. In our basic scheme, all the nodes but u update k_g but discard all the keys in R_u.

When a node v receives the node revocation notice and the MAC key that arrives one TESLA interval later, it verifies the authenticity of the notice based on TESLA. If the verification is successful, node v deletes its keys which have ids in I_u. If node v possesses k_M, the key possessed by the maximum number of remaining nodes, it will generate the new group key $k'_g = f_{k_M}(k_g)$ on its own as the key server does. Otherwise, it expects to receive k'_g from other nodes through a secure logical path.

To explain the key distribution process, we assume a multicast delivery tree rooted at the key server. In practice, the delivery tree should be already in place for the distribution of group data, and it can be constructed by any appropriate multicast/broadcast routing protocol. The key server initiates the process

[1]We can deploy the variant of TESLA proposed in [19] instead because it allows a receiver to verify the received messages immediately, thus preventing possible resource consumption attacks. Here we employ the original TESLA scheme for the ease of presentation

by sending k'_g to each of its children in the tree which does not have k_M through a *secure logical path* (discussed in detail below). A receiving node v can verify the key by computing and checking if $f_{k'_g}(0)$ is the same as the verification key it received earlier in the node revocation notice. The algorithm continues recursively down the delivery tree, i.e., each node v that has received k'_g transmits k'_g to its own children that have no k_M via secure logical paths.

The probability that a particular key K is allocated to a node in the key predistribution phase is equal to m/l. Therefore, on average, the number of nodes that possess K is given by Nm/l for a group size of N. Recall that k_M is selected on the basis of being the key possessed by the maximum number of nodes. Thus, the number of nodes in the network that possess k_M is generally larger than Nm/l. Therefore, a fraction of nodes can compute the new group key independently without waiting for it to be delivered to them. To reduce the rekeying latency and increase the reliability of delivering the new group key, these nodes can also independently start propagating the new group key to their downstream neighbors in the multicast tree. Furthermore, the parents of these nodes do not need to transmit k'_g to these nodes, thus saving the energy involved in transmitting and receiving k'_g.

Logical Path Discovery The *logical path discovery* process is necessary when a node wants to forward a new group key to its neighbors securely. We say there are logical paths between two nodes when (i) the two nodes share one or more keys. We call such paths *direct* paths. (ii) the two nodes do not share any keys, but through other nodes they can transmit messages securely to each other. We call such paths *indirect* paths and call the involved nodes *proxies*.

In our design, it is very easy to find logical paths between two nodes. Since the key pre-distribution algorithm is public and deterministic, a node u can independently compute I_v, the set of key ids corresponding to a node v's key set. Therefore, without pro-actively exchanging the set of its key ids with others, a node knowing the ids of its neighbors can determine not only which neighbors share or do not share keys with it, but also which two neighbors share which keys. The latter knowledge is very valuable when node u does not share any keys with a neighbor node v, because node u can ask a neighbor (say x) which shares keys with each of them to act as a *proxy*. For example, suppose node u shares a key k_{ux} with node x, node v shares a key k_{vx} with node x, but no shared keys exist between node u and node v. To forward a new group key k'_g to node v, the following steps are taken.

$$u \longrightarrow x : \{k'_g\}_{k_{ux}},\ x \longrightarrow u : \{k'_g\}_{k_{xv}},\ u \longrightarrow v : \{k'_g\}_{k_{xv}}$$

From this example, we can see that a *proxy* node acts as a translator between nodes.

We call node x in the above example node u's *one-hop proxy* to v. More generally, node x is said to be node u's *i-hop proxy* if x is i hops away from u and x shares a key with both u and v. If u and v do not have any *direct* paths and indirect paths via a one-hop proxy to each other, they can resort to using a *multi-hop proxy*. Our protocol always uses any direct paths that exist between nodes in preference to indirect paths, since the use of an indirect path incurs additional computational and communication overhead.

Security Analysis The security and correctness of the group rekeying scheme described above derive from the fact that (i) none of the revoked nodes can generate the new group key k'_g since they do not know k_M, (ii) none of the revoked nodes can obtain the new group k'_g since it is transmitted via secure logical paths established with keys not known to any of the revoked members, and (iii) every node can verify the new group key independently using the verification key it received in the authenticated revocation notice.

However, as the number of node revocations increases, the size of the set I_C also increases until ultimately $I_C = I_P$. In other words, all the keys in P are known to the coalition of all the revoked nodes. Thus, there is no k_M that can be used to update k_g. Let $Pr(w)$ denote the covering probability that the collusion

of w revoked nodes renders $I_C = I_P$, which is given by

$$Pr(w) = \begin{cases} 0 & \text{if } w < l/m \\ (1 - (1 - \frac{m}{l})^w)^l & \text{if } w \geq l/m \end{cases} \quad (1)$$

For given l, m and a covering probability p_0, we can calculate the value of w such that $Pr(w) = p_0$. For instance, for $l = 2000$, $m = 100$, and $p_0 = 90\%$, we have $w = 192$. That is, 192 nodes have a probability of 90% to cover the entire key pool.

Another security problem that arises as the number of revoked nodes increases is that the coalition of the revoked nodes may have keys that completely cover the key set of a legitimate node. Note that it is much easier for the compromised nodes to have keys to cover the m keys of a legitimate node instead of all the l keys in the key pool. Since compromised keys are discarded after node revocation, a legitimate node will no longer have any keys left that it can use to establish any logical paths to other nodes for obtaining the new group key. This node is therefore excluded from the network innocently, even though it is still a legitimate node.

Let $p_c(w)$ be the probability that the m keys of a legitimate node are completely covered by that of w colluding revoked nodes. We have

$$p_c(w) = (1 - (1 - \frac{m}{l})^w)^m. \quad (2)$$

From this equation, we know that by varying m and l we can obtain a desired security level. In Figure 1, we plot the number of colluding nodes (denoted as w_0) that the scheme can tolerate for a desired $p_c(w) = 10^{-6}$ for different m and l pairs. We observe that w_0 decreases with m but increases with l. Although this indicates that a smaller m and a larger l is more desirable from the security point of view, in Section 3.1 we shall show that a larger m and a smaller l is more desirable from the performance standpoint. Thus, the scheme needs to make a tradeoff between security and performance.

Define $N_r(w)$ as the number of nodes that are excluded from the system innocently after w nodes have been revoked. For a network of N nodes, $N_r(w) = (N - w) \cdot p_c(w)$. (At present, we do not consider the case where new nodes are added to the network.) In Figure 2, we plot $N_r(w)$ as a function of w for a network with $m = 100$ and $l = 2000$. We observe that $N_r(w)$ increases quickly with w. For example, when $w = 100$, more than 55% of nodes are excluded.

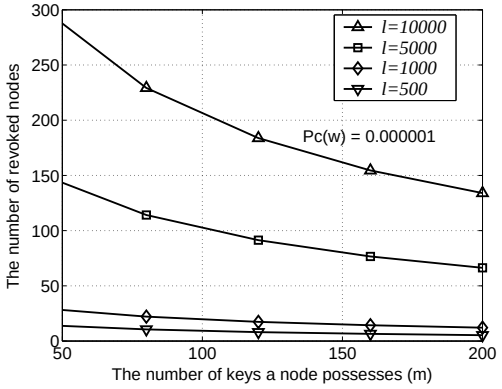

Figure 1: The number of revoked colluding nodes (i.e., w) the scheme can tolerate under different m and l pairs, given $p_c(w) = 10^{-6}$

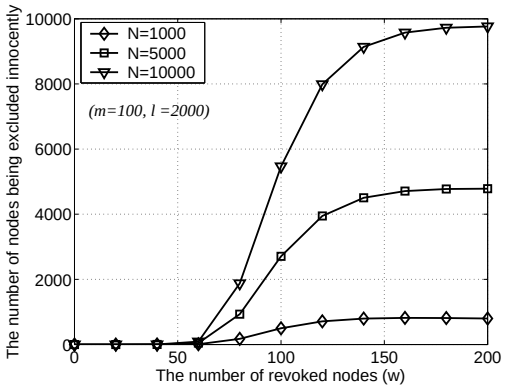

Figure 2: The number of nodes being excluded from the system innocently, $N_r(w)$, as the function of the number of revoked nodes w.

In summary, we can conclude that while our basic scheme works correctly, it does not scale well with the number of revoked nodes, because the nodes revoked during different rekeying events may collude. This

conclusion also applies to the previously probabilistic key pre-distribution schemes [8, 6, 33] because these schemes lack a key updating mechanism that prevents nodes compromised at different times from colluding.

2.4.2 Scheme II: Updating The Compromised Keys

The main reason that our basic scheme does not scale well is that compromised keys are discarded on every node revocation. We now present a second group rekeying scheme that addresses this issue. In this scheme, we modify the secure key update phase of our group rekeying algorithm so that the compromised keys are updated instead of being discarded. In our discussion below, we assume that a compromised node is revoked immediately and no new nodes are compromised before the current group rekeying is completed. We relax this assumption in scheme III discussed in Section 2.4.3.

The group rekeying operation involves the following steps.

1. The key server determines M, the id of the non-compromised key that is possessed by the maximum number of remaining nodes in the network. The key server then generates an intermediate key $k_{im} = f_{k_M}(k_g)$ and the new group key $k'_g = f_{k_{im}}(0)$. Then it broadcasts a node revocation message that contains M, the id of the revoked node (i.e., u), and $f_{k'_g}(0)$. The message is authenticated by TESLA as in the basic scheme. The key server further distributes k_{im} (not k'_g) to all the nodes through a secure key distribution process as in the basic scheme.
2. The nodes that possess k_M can compute the intermediate key k_{im} independently after it verifies the revocation message. The nodes that do not possess k_M receives k_{im} from their neighbors via logical paths secured by non-compromised keys, i.e., keys in $P - R_u$. Node u may impersonate a non-revoked node v by claiming node v's id, but this does not enable it to receive k_{im} because all the keys in R_u are precluded from establishing logical paths.
3. Every node computes the new group key $k'_g = f_{k_{im}}(0)$. It verifies the correctness of k'_g by computing and checking if $f_{k'_g}(0)$ equals to that in the node revocation message. Node u cannot compute k'_g because neither it has K_M nor it receives K_{im} from others.
4. Every node, say v, updates every key k_i in R_v as $k'_i = f_{k_i}(0)$. We denote the updated set of keys, k'_i's, as R'_v.
5. If k'_i is computed from k_i that was held by node u, i.e., $k_i \in R_u$, the nodes that hold k'_i further update k'_i as $k'_i = f_{k_{im}}(k'_i)$.
6. Finally, every node erases k_{im} and the original k_i's.

To understand this scheme, let us consider the *first* revocation in the network when node u is revoked. After step 2, every node except u has k_{im}. In step 3 and step 5, k_{im} is used to update the group key and the compromised keys (i.e., the keys in R_u). Since we use a pseudo-random function to update these keys, knowing the updated keys does not help an attacker compute k_{im}. Moreover, since the distribution of k_{im} is secured by only the non-compromised keys in P, updating all these non-compromised keys in step 4 and then erasing the original keys in step 6 prevents a node that is compromised in the future from providing any keys that enable the recovery of k_{im}.

Note that in step 6, every node deletes k_{im}; therefore, in the entire network, no node holds k_{im} and no node can recover it when the rekeying process is completed. Thus, all the keys in R_u are updated securely. In fact, the status of the system is reinstated to its original setting after every rekeying. Even if all the previously revoked nodes collude, they cannot compute the updated keys.

Security Analysis When w nodes are revoked *simultaneously* as a batch, their coalition might have keys to cover that of a legitimate node and thus exclude this node from the network. The expected number of

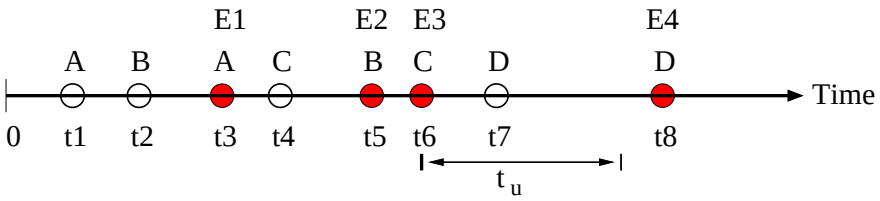

Figure 3: An attack on Scheme II. An empty circle stands for an event that a node is compromised, and a solid circle stands for an event that a compromised node is detected and hence revoked

excluded legitimate nodes, N_{excl}, is a function of w, the number of nodes being revoked simultaneously as a batch. We can compute N_{excl} using the same analysis that we used for computing the expected number of excluded legitimate nodes in the basic scheme, except that in the analysis of the basic scheme w stands for the cumulative number of revoked nodes. We find that N_{excl} is very small for reasonable values of w. For example, it is reasonable to assume that in most application and network scenarios the number of nodes that will be revoked simultaneously is at most 100. We can choose $m = 100$ and $l = 5000$ according to Figure 1. That is, for a group size of $1,000,000$, only one innocent node will be excluded from the network when 100 nodes are compromised simultaneously.

In Appendix A, we give a more formal analysis of the correctness and security of this scheme.

2.4.3 Scheme III: Allowing Detection Latency

Scheme II works correctly under the assumptions that a compromised node is revoked immediately and no new nodes are compromised before the current group rekeying is completed. Now we relax this assumption by allowing new compromises to happen, for example, due to the delay for node compromise detection.

Let u be a node that has been compromised but whose compromised status has not yet been detected. This delay in detecting a node compromise leads to a potential security loophole in Scheme II that can be exploited by a coalition of previously revoked nodes and node u, such that even after node u is revoked, the attackers can learn the new group key. This exploit depends upon the compromised node u *not erasing* the intermediate key, k_{im}, and the original keys in R_u, as it should have in Step 6 of Scheme II.

Figure 3 shows a possible attack[2]. Node A is compromised at t_1, but a group rekeying to revoke it occurs at time t_3 due to the detection latency. Node B is compromised at t_2, but this has not been detected yet. When the group is rekeyed at time t_3 using scheme II to revoke node A, an intermediate key k_{im} is distributed to the entire network, using keys that are not contained in R_A for establishing logical paths. In step 6, every node should erase k_{im} and all the k_i's in its own key set. However, since node B has been compromised, it keeps all these keys. If node B divulges k_{im} to node A, node A will be able to update its keys using the procedure in Step 3 and Step 4 of Scheme II, so that it has the current version of the keys in R_A, i.e., R'_A. At t_5, another group rekeying occurs for revoking node B. The distribution of k'_{im} is secured by keys except those in R'_B. If any of the keys in R'_A are used for encrypting k'_{im}, node A will be able to obtain k'_{im}. As a result, node A and B will be able to discover the new group key.

Our protocol will be secure if we assume that there is a reasonable upper bound on detection delay, denoted as t_u. We argue that this is a reasonable assumption for secure group communication. If a compromised node stays in a group without being detected, all group communication is insecure because this node knows all the group keys, and it does not matter what rekeying protocol is deployed.

Given an upper bound t_u on detection delay, the security loophole in Scheme II discussed above can be fixed by performing *batched rekeying*. Consider two consecutive rekeying events E_i and E_{i+1}. Scheme

[2] A similar attack was described in [5] in the context of Secure IP multicast.

III is based on the observation that if the time interval between E_i and E_{i+1} is larger than t_u, the node being revoked at E_{i+1} was not compromised at the time of the previous rekey event E_i. On the other hand, if the time interval between two consecutive rekeys is smaller than t_u, then during the rekey event E_{i+1}, we must assume that the node being revoked was already compromised at the time of E_i. Thus the nodes revoked at E_i and E_{i+1} could collude to obtain the new group key as discussed above. To prevent this from happening, in Scheme III, we treat E_{i+1} as a batched rekeying event if the time interval between E_i and E_{i+1} is smaller than t_u. Let S_i be the set of nodes that were revoked in the rekey event E_i, and let S_{i+1} be the set of compromised nodes whose detection led to rekey event E_{i+1}. Then, under scheme III, a batched group rekey is performed at E_{i+1} to revoke the nodes in $S_i \cup S_{i+1}$. Specifically, in the secure key distribution phase, we avoid using the keys of any of the nodes in $S_i \cup S_{i+1}$ for encrypting the new group key. In summary, if the time interval between E_{i+1} and E_i is larger than t_u, the key server performs an individual group rekey to revoke the nodes in S_{i+1}; otherwise, a batched group rekey is taken to revoke the nodes in $S_i \cup S_{i+1}$.

To illustrate the idea, let us assume the following relationship for the timing of the rekeying events in Figure 3: $t_5 - t_3 < t_u$, $t_6 - t_5 < t_u$, but $t_8 - t_6 > t_u$. We observe that the time (t_7) when node D is compromised cannot be earlier than the rekeying event at t_6; otherwise we must have $t_8 - t_6 \leq t_u$. Thus, at the rekeying time t_8, all the nodes participating in the rekey operation can conclude that at the rekeying time t_6, there was no compromised node in the system that had not yet been detected. Thus, during the rekeying at time t_8, the only node being revoked is node D.

For the rekeying event E_1 at t_3, no keys in R_A will be used for the distribution of k_{im}. In the rekeying event E_2 at t_5, no keys in $R'_A \cup R'_B$ will be used for key distribution. Similarly, at the rekeying event E_3 at t_6, no keys in $R''_A \cup R''_B \cup R''_C$ are used for key distribution. Hence, each of these rekeyings is a batched rekeying. In the rekeying event E_4 at t_8, however, the keys in $R'''_A \cup R'''_B \cup R'''_C$ can be used for key distribution, because node D was not compromised at the time of the previous rekey event.

Security Analysis The security of Scheme III lies between that of Scheme I and Scheme II. If no two consecutive rekeying events have a time interval greater than t_u, which means all the rekeyings are batched rekeyings, Scheme III is equivalent to Scheme I. On the other hand, if the time intervals between all consecutive rekeying events are greater than t_u, this scheme is identical to Scheme II. As such, the security of this scheme also lies between that of Scheme I and Scheme II, depending on t_u and the rekeying frequency.

2.5 Additional Issues

We now discuss some additional issues that arise in the deployment of our key management protocol.

2.5.1 Node Additions

Additional nodes may be added into the system even after the *key pre-distribution* phase discussed in Section 2.4.1 is complete. For example, the key server may introduce new nodes into the system to compensate for revoked nodes. To add a node u into the system, the key server first determines its key set R_u based on its node id. Then it loads node u with the current version of R_u and the current group key that allows u to communicate with other nodes in the network. Depending on the application under consideration, if a group rekeying is needed to prevent a new node to understand the earlier communication, the key server can simply broadcast a message instructing every node to update the group key k_g to $k'_g = f_{k_g}(0)$.

2.5.2 Secure Key Distribution

Since the main function of a network is distributing data, not distributing keys, it is reasonable to assume that an infrastructure, such as a multicast delivery tree [23] or mesh [17], or a broadcast tree [13], exists for data communication. None of these protocols are based on flooding for data distribution. Indeed, these protocols provide loop-free routing paths and ensure that every node receives every data packet only once, although they do involve bandwidth overhead in constructing and maintaining the delivery infrastructure. GKMPAN does not require to construct its own delivery infrastructure, but uses the underlying infrastructure directly. As such, GKMPAN neither involves the overhead of constructing and maintaining its own delivery infrastructure, nor introduces redundant messages in key distribution.

2.5.3 Networks with Non-member Nodes

In the previous discussion, we assumed that all nodes in the network were members of the group under consideration. We now discuss how GKMPAN can be extended to networks where only a fraction of nodes are part of the multicast group.

In multicast routing protocols for mobile ad hoc networks such as ODMRP [17] and the multicast extension of AODV [23], one or several non-member nodes may be involved in forwarding data packets for the group members that are not directly neighboring. In the case of group key management, the group members and some non-members form a multicast delivery tree [23] or mesh [17] with the key server of the group as the source. The member nodes can be considered to form an overlay network on top of the underlying multicast delivery tree that includes both members and non-members.

Similarly, GKMPAN can use an overlay network formed on top of a multicast delivery tree for the purposes of secure key distribution. In GKMPAN, a node will need to know the ids of its logical neighbors in the overlay network in order to establish secure logical paths with them. Currently in ODMRP and AODV a member node does not necessarily know the ids of its neighbors in the overlay network; however, node id information will be available when piggy-backed in a JOIN REQUEST, ROUTE REQUEST, or other membership control messages.

In GKMPAN, once a member node knows the ids of its neighbor nodes in the overlay network, they can communicate securely through their logical paths. Although non-member nodes are involved in forwarding the messages for member nodes, they cannot decrypt the messages. We note, however, that a selfish non-member node (just like a compromised member node) could launch refusal-of-service attacks by dropping the keying packets it is supposed to forward. This attack can be mitigated by the partial statelessness property of GKMPAN, as introduced below.

2.5.4 Packet Loss and Network Partitions

Consider the scenario where a node misses a group rekeying event because it does not receive one or more packets that are broadcast during the rekey event. This scenario might occur in ad hoc networks due to the intermittent connectivity within the ad hoc network, the inherent unreliability of the key delivery protocol, or the active refusal-of-service attacks discussed previously .

A straightforward solution for key recovery involves the node sending a retransmission request to the key server. The key server responds by sending a message that includes the ids of the revoked nodes and the intermediate key k_{im} that are used in this rekey operation. This message is encrypted with the individual secret key of this node, which is loaded into the node in the *key pre-distribution* phase. The node can therefore update its keys and compute the current group key correspondingly. However, it is very desirable to avoid this type of NACK-based solution, originally designed for wired networks, in the relatively higher packet loss environment of an ad hoc network because of the high communication and energy costs, as well

as the potential for NACK-implosion. Thus, we discuss solutions that attempt to recover the missing keys from the local neighborhood of the node, while only using the above solution as a last resort. We consider two scenarios below based on Scheme II.

Scenario I In the first scenario, a node receives the node revocation notice but does not receive the intermediate key k_{im}. This is not a problem if the node already possesses k_M. However, a situation may arise when the neighbors of a node that does not possess k_M have already erased k_{im}. Consequently, it cannot obtain k_{im} to correctly update any compromised keys that are known to the revoked node (see step 5 in scheme II). However, this node will still be able to update any keys that are not compromised (see step 4) and obtain the authenticated current group key from a neighbor through a logical path secured by a non-compromised KEK, if it has only experienced a relative small number of such packet loss events. By choosing appropriate m and l, we can design the system so that the probability that a node has a direct path to a neighbor is high even after a certain number of such losses.

We note that our rekeying scheme can be considered to be partially stateless since it falls between the stateful protocols such as LKH [27] and stateless protocols such as SDR [18]. A node can recover the current group key even if it has missed a certain number of rekey events – as long as it possesses some keys that are not known to the set of nodes that have been revoked.

Scenario II The second scenario arises if a node does not receive the node revocation notice, the notice arrives too late violating the security condition in TESLA, or the notice is verified to be incorrect. However, the node knows that it has missed a group rekeying event because it receives data encrypted with a later version of the group key (a group key should have a key version field) or it actually receives the intermediate key k_{im} from another node. In this case, the most important issue is to verify the authenticity of the rekey event. It can do this by communicating with the base station. Below we sketch an alternative solution that involves only local communication, although it provides weaker security.

Let the node missing a notice S be u, and its current neighbors be $v_1, v_2, ..., v_s$. Let $R_{uv} \stackrel{def}{=} R_u \cap R_v$. Node u broadcasts a query with $TTL = 1$. A neighbor, say v_j, computes $k_{enc} = \text{XOR } \delta_i, \forall \delta_i \in R_{uv_j}$ if $R_{uv_j} \neq \emptyset$, and sends the following reply:

$$v_j \longrightarrow u : S, MAC(k_{enc}, S).$$

The more such notices node u receives, the higher the confidence it has in the authenticity of the notice. Let z be the number of distinct keys in $\cup_{j=1}^{s} Ruv_j$. To forge a notice to node u, the attacker must have all these z keys. The probability $p_f(w)$ that the attacker possesses these z keys after it compromised w nodes is

$$p_f(w) = (1 - (1 - m/l)^w)^z. \tag{3}$$

From this equation, a node can determine the threshold z_0 such that $p_f(w)$ is smaller than a desired level (say 10^{-6}) for a given w. If z is larger than z_0, it accepts the notice; otherwise, it may ignore the notice, or find additional nodes to provide enough MACs of the notice. Note that even the attacker succeeds in forging a notice to node u, it does not compromise the security of the system. Indeed, in the worst case node u can contact the base station to recover its keys. Therefore, in practice, the value of $p_f(w)$ could be a little larger, say 10^{-3}, and a node may only poll a small number of neighbors. After a node obtains an authentic notice via this approach, it then proceeds as in the first scenario.

Note that the above technique is different with the q-composite scheme in [6] where the goal is to provide a certain degree of network connectivity when two nodes establish a pairwise key when they share at least

q common keys. This is achieved by selecting appropriate m and l. While in our case, we do not put any requirement on the number of common keys between a node and one of its neighbors because we use multiple neighbors to provide enough common keys for authentication.

2.5.5 Other Security Attacks

We assumed that an attacker may eavesdrop on all traffic, inject packets or replay older packets. Because the key server authenticates all the rekeying messages by TESLA [19], no nodes can inject any fake rekeying messages into the network or modify any rekeying messages they are forwarding while impersonating the key server. That is, malicious nodes cannot cause other nodes to accept forged group keys. Because time-stamp information is also embedded in every TESLA key, the attacker cannot replay older rekeying packets.

Possessing the keys of revoked nodes normally does not help an attacker launch refusal-of-service attacks, because the revoked nodes are not part of the multicast delivery tree any longer – they do not have any valid keys to establish secure logical paths to other nodes. Moreover, the worst situation caused by refusal-of-service attacks is equivalent to that due to packet losses or network partitions, which our protocol can handle well due to its partial statelessness property. We note that the mobility of nodes helps relieve this attack as well.

3 Performance Analysis

We now study the performance of our rekeying protocol GKMPAN. We analyze the communication cost of GKMPAN and discuss the tradeoff between performance, security, and storage cost. We do not consider the computational cost because GKMPAN only involves a few inexpensive symmetric key operations and/or pseudo random function evaluations during every group rekeying.

3.1 Communication Cost

The communication cost of a group rekeying operation is composed of the cost for broadcasting a node revocation notice and the cost for secure group key distribution. The revocation notice includes the id(s) of the node(s) being revoked, a verification key and a MAC of the notice, and the message is usually very small unless the number of nodes being revoked is very large. Note that a revocation notice is necessary no matter what rekeying scheme is being used.

In the secure key distribution phase of the rekey operation, a node transmits one encrypted key (i.e., the new group key) to each of its children in the delivery tree if they share a direct path. Every node receives one encrypted key; therefore, on average every node transmits one encrypted key although an intermediate node may transmit more than one keys whereas a leaf node does not transmit at all. However, if a node does not have a direct logical path to a child node, it will need to communicate with a proxy node to form an indirect logical path. For a (round-trip) communication with a proxy node that is i hops away, the total number of keys transmitted is $2i$. Therefore, the communication cost for secure key distribution is determined by the numbers and types of logical paths existing between two neighboring nodes in the network, which are in turned determined by (i) the parameters l and m (ii) the network node density (iii) the number of nodes being revoked at a batch.

Impact of Probabilistic Key Sharing Parameters For a given l and m, p_s, the probability of directly sharing at least one key between any two nodes initially (i.e.,prior to any node revocations) is

$$p_s = 1 - \frac{\binom{l}{m} \cdot \binom{l-m}{m}}{\binom{l}{m}^2} = 1 - \frac{\binom{l-m}{m}}{\binom{l}{m}}. \quad (4)$$

Because the probability that a key is picked by both the nodes is $(\frac{m}{l})^2$, z_s, the expected number of keys shared between any two nodes initially (i.e., the number of *direct* paths between them) is $z_s = l \cdot (\frac{m}{l})^2 = \frac{m^2}{l}$.

From the above analysis, we see that for a given l, both p_s and z_s increase with m, while both p_s and z_s decrease with l for a given m. Also, we can see that m has a larger effect on z_s than l has. Thus, we can use the equations above to select l and m so that the probability of existing *direct* paths between two nodes is large enough. For example, to obtain $p_s = 99.5\%$, we can choose $l = 2000$ and $m = 100$ ($z_s \approx 5$). In this case, only 0.5% nodes have to receive the group key through an indirect path. Thus, the average transmission cost per node is close to one key.

Impact of Node Density The average number of immediate neighbors of a node depends upon the density of the network as well as the transmission range of a node. The number of neighbors of a node does not affect the probability p_s that two nodes share direct paths, but affects the number of indirect paths between them. If a node does not have a direct path to another node, the probability that one of its immediate neighbors can act as one-hop proxy between the two nodes will increase with the number of neighbors. However, irrespective of the node density, only $1 - p_s$ of nodes have to resort to indirect logical paths. Thus, node density has only negligible impact on the performance of GKMPAN when p_s is large enough. For example, when $m = 100$ and $l = 2000$, only 0.5% nodes have to receive the group key through an indirect path. Moreover, for these values of m and l, as long as two nodes have one common neighbor, they can establish a direct path or a one-hop indirect path with the probability of 99.995%. That is, the situation that two nodes have to resort to a multi-hop proxy node is extremely rare. This indicates that the transmission cost of a group rekeying is close to one key per node. Our simulation result shown later confirms this analysis.

Impact of Number of Nodes Being Revoked The above analysis shows how the parameters m and l affect the *initial* number of logical paths between any two nodes. However, the keys known to the nodes being revoked as a batch cannot be used during the current rekeying event. As a result, both p_s and z_s decrease with the number of revoked nodes.

Let $p_s(w)$ be the probability that two nodes share at least one key when w nodes are revoked simultaneously. We use the following analysis to compute $p_s(w)$. Consider a key k. The probability p_1 that k is chosen by both nodes is $(m/l)^2$, and the probability p_2 that k is not chosen by any of the w compromised nodes is $(1 - m/l)^w$. Therefore, the probability p_0 that k is "safe" is $p_0 = p_1 p_2$. The probability $p_s(w)$ is the same as the probability that at least one of the l keys in the key pool is "safe" to both nodes when w nodes are revoked. Therefore, we have

$$p_s(w) = 1 - (1 - (m/l)^2 (1 - m/l)^w)^l. \tag{5}$$

Similarly, $z_s(w)$, the expected number of shared keys between two nodes when w nodes are revoked is

$$z_s(w) = l(m/l)^2 (1 - m/l)^w. \tag{6}$$

In Figure 4 we plot $p_s(w)$ as a function of w. The figure clearly indicates that $p_s(w)$ decreases with w. Hence, in order to deliver the group key to a downstream node, a node might have to resort to using a one-hop proxy or even a multi-hop proxy. As a result, the cost of group rekeying increases with w. The figures also shows that it is possible to achieve a desirable $p_s(w)$ by choosing appropriate m and l. For example, when $m = 40, l = 500$, $p_s(10) = 75\%$; whereas $p_s(10) = 96\%$ when $m = 200$ and $l = 10000$. Thus, in the latter case the communication cost of a small batch rekeying is almost as low as that of an individual rekeying.

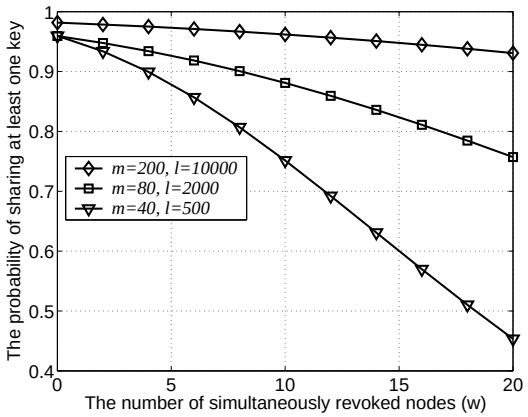

Figure 4: The probability of sharing at least one key between two nodes when w nodes are revoked simultaneously

A Rough Comparison with LKH We now compare the communication cost of GKMPAN with that of LKH [27]. Since LKH was proposed for wired networks, not for ad hoc networks, this comparison should not be interpreted as a direct comparison against LKH. Rather, our purpose is to illustrate the differences in communication cost between a protocol that was designed for a wired environment as opposed to a protocol that is geared towards wireless ad hoc networks. To avoid misunderstanding, we denote the LKH for ad hoc networks as LKHA. We also note that this comparison is based mainly on bandwidth not security. Indeed, we should mention that LKHA provides stronger collusion resistance than GKMPAN because in LKHA no coalition of compromised nodes can cover the keys of other nodes though the covering probability in GKMPAN is already very small.

The group rekeying scheme for ad hoc networks in [16] shows that it is possible to reduce the cost of the original LKH scheme by $15\% \sim 37\%$ (although it may incur a larger overhead in some scenarios) by mapping the physical locations of the members to the logical key tree in LKH for a static network. Because the performance overhead in [16] is of the same order as that of the original LKH scheme, we can also see the comparative performance of GKMPAN with respect to the protocol described in [16]. Note that we do not consider reliable key delivery in the comparison, which actually biases the comparison in favor of LKHA since LKHA is a stateful protocol.

As discussed in Section 2.5, GKMPAN uses the underlying delivery infrastructure for key distribution. In the simulation study, we use an underlying spanning tree for delivering keys in both the protocols. Both the key server and the nodes are randomly distributed in a fixed space of 1000×1000 square units and the transmission range of a node is 75 units. We consider a static network, which corresponds to a snapshot of the ad hoc network at the time of a rekeying event, instead of a mobile network. This is because group rekeying only takes a very short time relative to mobile node velocities. Moreover, since a node only has to forward the group key to its direct neighbors (and the knowledge of neighbor ids usually comes for free from the Media Access Control protocol [10] or a routing protocol [23]), GKMPAN does not introduce the bandwidth overhead for maintaining multiple-hop routes. Also, node mobility does not reduce the probability that two neighboring nodes establish a logical path.

The metrics of interest are the average numbers of keys a node transmits and receives respectively in every rekeying. We use the method of independent replications for our simulation. All the results have 95% confidence intervals that are within 5% of the reported values. In Figure 5 we compare the communication cost of GKMPAN and LKHA by varying w, the number of nodes revoked as a batch. The communication cost does not include the cost of the node revocation notice because both LKHA and GKMPAN incur this

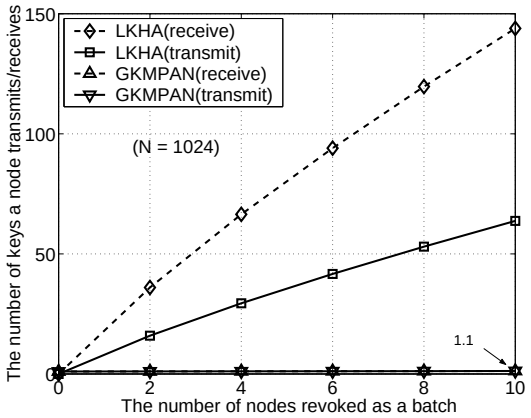

Figure 5: The impact of the number of nodes being revoked as a batch on the communication cost of LKHA and GKMPAN respectively.

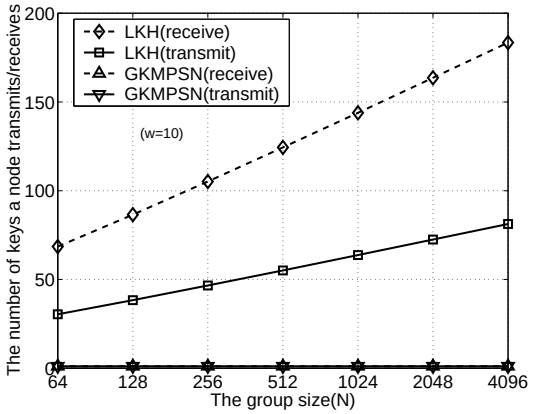

Figure 6: The impact of the group size on on the communication cost of LKHA and GKMPAN respectively

cost. The group size is $N = 1024$. In GKMPAN, $m = 40$ and $l = 500$. We can observe that the communication cost for LKHA is much higher than that of GKMPAN. In GKMPAN the average communication cost per node is approximately one key and is almost independent of w. In LKHA, both the number of keys a node transmits (T_n) and the number of keys a node receives (R_n) increase with w. Here $R_n > T_n$ because all the nodes receive the updated keys whereas only non-leaf nodes in the broadcast tree rebroadcast the keys. Therefore, GKMPAN greatly outperforms LKHA for individual or small batch rekeying in ad hoc networks. In Figure 6, we observe a similar result when varying the group size N.

3.2 Security, Performance and Storage Tradeoffs

As we observed above, to increase the number of *direct* logical paths between two nodes, it is necessary to increase m or decrease l. Further, increasing m has a larger impact on z_s. However, from the viewpoint of storage, a smaller m is more desirable. Moreover, as we showed in Figure 1, a smaller m and a larger l could increase the security of our schemes. Due to these conflicting requirements, the parameters m and l should be selected based on the application under consideration.

A Comprehensive Example Suppose that a node has space for 200 keys, i.e., $m = 200$. The key server chooses $l = 10000$. According to eqn. 4, we have $p_s = 98.3\%$, i.e., the probability that two nodes share *direct* paths is 98.3%. Let the desired security level be $p_c(w) = 10^{-6}$. According to eqn. 2, we have $w = 134$, i.e., the probability that the coalition of 134 nodes have keys to cover a legitimate node is about 10^{-6}. When 50 nodes are simultaneously revoked, the fraction of direct, indirect paths involving a one-hop proxy, and indirect paths involving a two-hop proxy are 76.7%, 22.7% and 0.6% respectively. Our simulation shows that on average a node needs to receive and transmit 1.2 keys.

4 Related Work

Group Rekeying Schemes Group rekeying has been extensively studied in the context of secure multicast in wired networks. The rekeying schemes can be categorized into stateful and stateless protocols. The stateful class of protocols includes several protocols based upon the use of logical key trees, e.g., LKH [27], OFT [3], ELK [22]. In these protocols, the key server uses key encryption keys that were transmitted to members during previous rekeying operations to encrypt the keys that are transmitted in the current rekeying operation. Thus, a member must have received all the key encryption keys of interest in all the previous rekey operations; otherwise, it will not be able to decode the new (group) key. Adding redundancy in key distribution [30, 25] will not solve the problem completely due to high packet loss rates or network partitions. *Stateless* group rekeying protocols [15, 18, 24] form the second class of rekey protocols. In these protocols, a legitimate user only needs to receive the keys of interest in the current rekey operation to decode the current group key. The stateless feature makes these protocol very attractive for ad hoc networks. However, these protocols have much higher communication overhead than the stateful protocols. GKMPAN differs with the above schemes mainly in two respects. First, it provides *partial* statelessness, that is, a node can miss a certain number of group rekeyings with the need of asking the key server for retransmission. Second, GKMPAN has much smaller per node transmission cost than the other schemes.

Other Key Management Schemes Basagni et al. [1] presented a rekeying scheme for periodically updating the group-wide data encryption key in a stationary sensor network. However, their scheme assumes sensor nodes are tamper-free, and it does not address the issue of rekeying on node compromise. Zhu et al. [32] proposed a group key management scheme for stationary sensor networks. Kaya et al. [11] presented a group key management scheme for ad hoc networks based on public-key techniques. GKMPAN, using symmetric-key techniques only, is designed for mobile ad hoc networks and makes no assumption about the strength of a node in resisting to compromise.

Eschenauer and Gligor [8] presented an efficient key management scheme for sensor networks based on probabilistic key predeployment. Chan et al. [6] and Zhu et al. [33] extended this scheme and present new mechanisms for pairwise key establishment. GKMPAN also uses the probabilistic key predeployment technique as the underlying means to establish secure channels between nodes. However, in these schemes, the predeployed keys are used for encrypting all communications between nodes; there is no group key. In contrast, we propose to use the predeployed keys only as KEKs for securely distributing a group key to the nodes in the network while using the group key for securing group data communications. Thus, GKMPAN incurs much smaller communication and computational overhead in group communication. Last but not least, those schemes do not update any compromised keys, while GKMPAN includes an efficient mechanism to update the keys known to revoked nodes.

Du et al. [7], Liu and Ning [14] combined the technique of probabilistic key predeployment with the Blom's [2] or the Blundo's [4] key management schemes for establishing pairwise keys in sensor networks. They have shown that their schemes are more robust to node collusion attacks than the previous schemes [6,

8]. Thus, GKMPAN can also base on these schemes for delivering group keys between nodes. However, our key updating scheme cannot apply directly to updating the predeployed polynomials or matrixes in these schemes when revoking a compromised node. We are investigating the secret updating issue in these schemes as our future work.

5 Conclusions

In this paper, we have presented GKMPAN, a scalable and efficient group key management protocol for ad hoc networks. Our protocol is based on a probabilistic key sharing scheme that can be parameterized to meet the appropriate levels of security and performance for the application under consideration. The main component of GKMPAN is a novel group rekeying protocol that has the following properties:

- It is efficient – it relies only on symmetric key cryptography and the computational and communication cost of group rekeying are distributed among the nodes in a network.
- It is scalable – the storage requirements per node are independent of the size of a network.
- It is partially stateless – a node can decode the current group key even if it has missed a limited number of previous group rekeying operations.

We have implemented GKMPAN on a sensor network testbed, as a building block of our security for Microsensor Networks project [21]. The source code is also available for download in the project website.

References

[1] S. Basagni, K. Herrin, E. Rosti, and D. Bruschi. Secure Pebblenets. In Proc. of the 2nd ACM International Symposium on Mobile Ad hoc Networking & Computing, 2001, pp.156 - 163.

[2] R. Blom. An Optimal Class of Symmetric Key Generation Systems. In Advances in Cryptology - EUROCRYPT'84, LNCS 209, 1984, pp. 335-338.

[3] D. Balenson, D. McGrew, and A. Sherman. Key Management for Large Dynamic Groups: One-Way Function Trees and Amortized Initialization. IETF Internet draft (work in progress), August 2000.

[4] C. Blundo, A. Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung. Perfectly-secure key distribution for dynamic conferences. In Advances in Cryptology - CRYPTO'92, LNCS 740, pp. 471-486, 1993.

[5] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, and D. Saha. Key Management for Secure Internet Multicast using Boolean Function Minimization Techniques. In Proc. of IEEE Infocomm'99, Mar. 1999, pp. 689–698.

[6] H. Chan, A. Perrig, and D. Song. Random Key Predistribution Schemes for Sensor Networks. In Proc. of IEEE Symposium on Security and Privacy, Oakland, May 2003.

[7] W. Du, J. Deng, Y. Han, and P. Varshney. A Pairwise Key Pre-distribution Scheme for Wireless Sensor Networks. In Proc. of 10th ACM Conference on Computer and Communications Security (CCS'03), Washington DC, October 27-31, 2003.

[8] L. Eschenauer and V. Gligor. A Key-Management Scheme for Distributed Sensor Networks. In Proc. of the 9th ACM Conference on Computer and Communications Security CCS, Washington D.C., 2002.

[9] O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. Journal of the ACM, Vol. 33, No. 4, 1986, pp. 210-217.

[10] IEEE Computer Society LAN MAN Standards Committee. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE Std 802.11, 1997.

[11] T. Kaya, G. Lin, G. Noubir, and A. Yilmaz. Secure Multicast Groups on Ad Hoc Networks. In Proc. of ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN'03), 2003.
[12] J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang. Providing Robust and Ubiquitous Security Support for Mobile Ad-Hoc Networks. In Proc. of the 9th Internation Conference on Network Protocols (ICNP'01), 2001.
[13] H. Lim and C. Kim. Multicast tree construction and flooding in wireless ad hoc networks. In Proc. of ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems, 2000.
[14] D. Liu and P. Ning. Establishing Pairwise Keys in Distributed Sensor Networks. In Proc. of the 10th ACM Conference on Computer and Communications Security (CCS'03), Washington D.C., October, 2003.
[15] D. Liu, P. Ning, and K. Sun. Efficient Self-Healing Group Key Distribution with Revocation Capability. In Proc. of the 10th ACM Conference on Computer and Communications Security (CCS'03), October, 2003.
[16] L. Lazos and R. Poovendran. Energy-Aware Secure Multicast Communication in Ad-hoc Networks Using Geographic Location Information. In Proc. of IEEE ICASSP'03, Hong Kong, China, April, 2003.
[17] S. Lee, W. Su, and M. Gerla. On-Demand Multicast Routing Protocol in Multihop Wireless Mobile Networks. ACM/Kluwer Mobile Networks and Applications, 2000.
[18] D. Naor, M. Naor, and J.Lotspiech. Revocation and tracing schemes for stateless receivers. In Advances in cryptology - CRYPTO 2001, Santa Barbara, CA, Aug. 2001.
[19] A. Perrig, R. Canetti, D. Song, and J. Tygar. Efficient and secuire source authentication for multicast. In Proc. of Network and Distributed System Security Symposium (NDSS'01), Feb. 2001.
[20] A. Perrig, R. Canetti, J. Tygar, and D. Song. Efficient authentication and signing of multicast streams over lossy channels. In Proc. of IEEE Symposium on Security and Privacy, Oakland, CA, May 2000.
[21] URL: http://rogue.cs.gmu.edu/sensor/.
[22] A. Perrig, D. Song, and D. Tygar. ELK, a new protocol for efficient large-group key distribution. In Proc. of IEEE Symposium on Security and Privacy, Oakland, CA, May 2001.
[23] E. Royer and C. Perkins. Multicast Operation of the Ad-Hoc On-Demand Distance Vector Routing Protocol. In Proc. of ACM MobiCom '99, Seattle, WA, Aug. 1999.
[24] J. Staddon, S. Miner, M. Franklin, D. Balfanz, M. Malkin and D. Dean. Self-Healing Key Distribution with Revocation. In Proc. of IEEE Symposium on Security and Privacy, Oakland, CA, 2002.
[25] S. Setia, S. Zhu and S. Jajodia. A Comparative Performance Analysis of Reliable Group Rekey Transport Protocols for Secure Multicast. In Proc. of Performance 2002, Rome, Italy, Sept 2002, pp. 21 - 41.
[26] URL: http://www.tinyos.net/.
[27] C. Wong, M. Gouda,S. Lam. Secure Group Communication Using Key Graphs. In Proc. Of ACM Special Interest Group on Data Communication (SIGCOMM'98), 1998.
[28] D. Wallner, E. Harder and R. Agee. Key Management for Multicast: Issues and Architecture. Internet Draft, draft-wallner-key-arch-01.txt, September 1998.
[29] URL: http://www.xbow.com.
[30] Y. Yang, X. Li, X. Zhang and S. Lam. Reliable group rekeying: Design and Performance Analysis. In Proc. of ACM Special Interest Group on Data Communication (SIGCOMM'01), San Diego, CA, USA, August 2001.
[31] L. Zhou and Z. Hass. Securing Ad Hoc Networks. IEEE Network Maganize, 13(6), November/December 1999.
[32] S. Zhu, S. Setia, and S. Jajodia. LEAP: Efficient Security Mechanisms for Large-scale Distributed Sensor Networks. In Proc. of the 10th ACM Conference on Computer and Communications Security (CCS '03), October, 2003.

[33] S. Zhu, S. Xu, S. Setia, and S. Jajodia. Establishing Pair-wise Keys for Secure Communication in Ad Hoc Networks: A Probabilistic Approach. In Proc. of IEEE ICNP'03, Atlanta, Georgia, November 4-7, 2003.

Appendix A

We require a key management scheme satisfy the following requirements.

1. Correctness. We require that the following two properties possessed by the system after the initialization be preserved after each rekeying process: (1) all the non-compromised nodes hold a unique group key k_g, and (2) for any two non-compromised nodes u, v that hold a same k_α, they should hold the same key k'_α after each rekeying process.

2. Security. We require that the adversary that has compromised w'' nodes (which may be much greater than w) knows no computational information about the non-compromised keys.

Theorem 1 *Scheme II satisfies the correctness and security requirements.*

First, it is easy to see that the scheme satisfies the *correctness* requirement. This is so since all the non-compromised nodes get the unique k_{im}, which enables the affected nodes to update the affected keys using the same transformation. The same argument applies to the update of k'_g.

Second, we claim that our scheme satisfies the aforementioned security requirement. This is supported by the following lemmas.

Lemma 1 *The adversary cannot distinguish the newly established k_{im} from a random string.*

proof (sketch) We have assumed that no new nodes are compromised before the rekeying process is done. We stress that the scheme ensures that after revoking w simultaneously compromised nodes, a non-compromised node, with a high probability, still holds at least a non-compromised key that enables the node to establish logical paths with others to update its compromised keys and k_g. (In the rare case that there is a cover, we simply ignore that node.)

Suppose the encryption functions for transmitting k_{im} are based on a pseudo random function family $\{f_k\}$. We consider an experiment $\mathcal{EXPR}$ where f_k for all the non-compromised keys k's (including k_M) that are shared among the non-compromised nodes are substituted with random functions; in other words, the encryption functions and the computation of k_{im} are based on the corresponding random functions. Then clearly k_{im} is a random string.

Suppose the adversary can distinguish k_{im}, which is established and held by the non-compromised nodes in the real-world system, from a random string of the same length. Then, a standard hybrid argument shows that there is a probabilistic polynomial-time algorithm that is able to distinguish a pseudo random function from a random function. Thus the lemma holds.

Given the above lemma, the following is almost immediate to see.

Lemma 2 *The adversary compromising simultaneously at most w nodes learns negligible information about (1) $k' = f_{k_{im}}(k)$ for any k, and (2) $k'_g = f_{k_{im}}(0)$.*