Establishing Pairwise Keys for Secure Communication in Ad Hoc Networks: A Probabilistic Approach

Sencun Zhu[1] Shouhuai Xu[2*] Sanjeev Setia[1†] Sushil Jajodia[1]

[1]Center for Secure Information Systems, George Mason University, Fairfax, VA 22030
[2]Department of Computer Science, University of Texas at San Antonio, San Antonio, TX 78249
Email:{szhu1,setia,jajodia}@gmu.edu,shxu@cs.utsa.edu

Abstract

A prerequisite for secure communication between two nodes in an ad hoc network is that the nodes share a key to bootstrap their trust relationship. In this paper, we present a scalable and distributed protocol that enables two nodes to establish a pairwise shared key on the fly, without requiring the use of any on-line key distribution center. The design of our protocol is based on a novel combination of two techniques – probabilistic key sharing *and* threshold secret sharing. *Our protocol is scalable since every node only needs to possess a small number of keys, independent of the network size, and it is computationally efficient because it only relies on symmetric key cryptography based operations. We show that a pairwise key established between two nodes using our protocol is secure against a collusion attack by up to a certain number of compromised nodes. We also show through a set of simulations that our protocol can be parameterized to meet the desired levels of performance, security and storage for the application under consideration.*

1 Introduction

For secure communication between two mobile nodes in an ad hoc network, i.e., secure peer-to-peer communication, it is necessary for the two nodes to share a secret key. This can be easily achieved if we assume the existence of a public key infrastructure. However, many mobile ad hoc networks cannot afford to deploy public key cryptosystems due to their high computational overheads and storage constraints. For instance, Brown *et al* [3] reported that a 512-bit RSA signature generation takes $2-6$ seconds on a RIM Pager and on a Palm Pilot, and Perrig *et al* [23] reported that a current generation sensor node has just 4500 bytes for security and applications. Consequently, it is necessary to explore approaches that are based on symmetric key cryptography.

A fundamental issue that must be addressed if symmetric keys are used for secure communication is key distribution. The simplest strategy is to use a pairwise key predeployment scheme, in which every node is pre-loaded with $N-1$ keys each of which is shared with another node, where N is the number of nodes in the network. Clearly, this scheme is not suitable for large networks since the storage required per node increases linearly with network size.

In a seminal work, Needham and Schroeder [20] proposed an approach in which an on-line server acts as a key distribution center (KDC) for establishing a pairwise shared key between two nodes. Each node is pre-loaded with a unique key that it shares with the KDC. To communicate securely, a pair of participants obtain fresh session keys from the on-line server. For example, secret key protocols such as Kerberos [15] and Otway-Rees [21] require an interactive trusted third party, a KDC, or a Key Translation Center (KTC) in order to establish a shared key between any two nodes. While these schemes have been widely deployed in wired networks, this approach is unsuitable for ad hoc networks that are characterized by dynamic topology changes and node failures (e.g., due to battery exhaustion), and by the fact that there is typically no on-line server available.

In this paper, we present a scalable distributed protocol that enables two nodes in an ad hoc network to establish a pairwise shared key on the fly, without requiring the use of an on-line key distribution center. The design of our protocol is based on a novel combination of two techniques – *probabilistic key sharing* and *threshold secret sharing*. In our protocol, the storage requirements per node depend only on the level of security desired and are independent of the size of the network. Our protocol relies only on symmetric key cryptography operations and is thus computationally efficient.

Unlike a traditional pairwise key that is shared *only* between two nodes, a pairwise key established using our pro-

*Work mostly done while affiliated with University of California at Irvine.

†Also with Dept. of Computer Science, George Mason University.

tocol is *exclusively known* to the peers with *overwhelming probability*, and it is secure against a collusive attack by up to a certain number of compromised nodes. We study the performance and security aspects of our protocol through both analysis and detailed simulation, and show that the protocol can be parameterized to meet the desired levels of security, performance and storage for the application under consideration.

The rest of this paper is organized as follows. We discuss related work in Section 2 and present the details of our pairwise key establishment protocol in Section 3. We analyze its performance and security in Section 4, and discuss its availability and robustness in Section 5. Finally, Section 6 concludes this work.

2 Related Work

Probabilistic Key Sharing As discussed in Section 1, most of the proposed symmetric key cryptography protocols for establishing a pairwise shared key between two nodes make use of an on-line key server. Mitchell and Piper [19] proposed a solution based on probabilistic key sharing that does not depend on such an on-line server. However, the storage complexity imposed on each participant in their scheme seems to be unaffordable in the context of ad hoc networks (cf. [25] for lower bounds).

The probabilistic keying scheme in our protocol is similar to schemes that have been used by other researchers [10, 5]. Eschenauer and Gligor [10] introduced a key management scheme based on probabilistic key sharing for distributed sensor networks (DSN) with central key servers (e.g., base stations). Chan et al. extended this scheme by presenting three new mechanisms for key establishment in sensor networks based on the framework of probabilistic key predeployment, including a mechanism for pairwise shared key establishment called *multipath key reinforcement*.

Our work differs from the previous ones in several aspects. First, in our scheme, a node can deduce the set of keys it shares with any other node (which may be an empty set) only based on the latter's identity. In contrast, the approaches in [10, 5] require each node to exchange the ids of the keys it possesses with its neighbors. Thus, our approach trades computation for communication, which is desirable in ad hoc networks. Second, Eschenauer and Gligor proposed using the predeployed keys for encrypting all communication between nodes. A session key between two nodes can also be established using a logical path secured by the predeployed keys. However, it seems that the established session key might not be exclusively known to the two nodes involved, because each predeployed key is known to several nodes. In contrast, we propose using the predeployed keys for establishing a shared pairwise key that is exclusively known to two nodes with overwhelming probability.

We note that the *multipath key reinforcement* scheme in [5] shares some similarities with our scheme in that both the schemes use the idea of secret sharing; however, our scheme differs from theirs in the following aspects. First, their scheme uses multiple *physically* disjoint paths between two nodes in establishing a pairwise key, while our scheme can use a single physical path as long as the shares are transmitted over multiple *logically* disjoint paths. Second, we present a detailed security and performance analysis of the idea of combining probabilistic key sharing and secret sharing, and also present an algorithm for deciding the number of secret shares to be used for establishing a pairwise key based on the desired level of security.

Threshold Secret Sharing There has been a great deal of research on threshold secret sharing Shamir [26] and its applications. In one direction, Gong [13] proposed an approach in which threshold secret sharing is used for increasing the availability of authentication services. Our work bears the similarity that we also utilize secret sharing techniques to establish pairwise keys. Unlike Gong's scheme, however, our scheme does not use any single on-line key server. In another relevant direction, researchers have extensively investigated the interplay of network connectivity and secure and reliable communication (e.g., Dolev [6], Delev et al. [8], Franklin and Wright [11], Desmedt and Wang [7]). We refer the reader to Bagchi et al. [2] for an overview and recent result in this regard.

3 The Pairwise Key Establishment Protocol

In this section, we first describe our assumptions and present the basic ideas underlying our scheme, and then present our scheme in detail.

3.1 Overview

Network, Node and Security Assumptions First, we assume network links are bidirectional, i.e., if node A can hear node B, B can also hear A. This is true when all the nodes use omnidirectional antennas and have equal power levels. Second, we aim to provide solutions for low-end devices. The resources of a node such as power, storage, computation and communication capacity, are relatively constrained, making public key techniques impractical. We assume that every node has space for storing hundreds of bytes or a few kilobytes of keying materials, depending on the security requirements. Third, we do not assume a central key server exists in the formed network, whereas it may exist off-line to initiate the nodes prior to the formation of the network. Fourth, we assume that if a node is compromised, all the

information it holds will also be compromised. We do not distinguish between a compromised node and an attacker. Moreover, all the compromised nodes may try to eavesdrop on other nodes' communications and collude to launch attacks by sharing their keying materials.

Protocol Operation Our pairwise key establishment protocol is based on two techniques – probabilistic key sharing and threshold secret sharing.

Before the deployment of a network, i.e., during a *key pre-distribution* phase, every node is loaded with a (small) fraction of keys out of a large pool of keys by a key server. Note that this phase occurs before the deployment of the network, and the key server stays off-line after finishing this phase. Keys are allocated to each node using a probabilistic scheme that enables every pair of nodes to share one or more keys with certain probability. The keys directly shared between any two nodes can thus be used to encrypt messages exchanged between them. Even if two nodes do not share any keys directly, our probabilistic key sharing scheme enables them to communicate securely using logical paths obtained via a *logical path discovery* process that will be presented in Section 3.2.

To be concrete, consider two nodes u and v that wish to communicate privately. u and v may already share one or more keys from the pool of keys after the *key pre-distribution* phase. However, these keys are *not* known exclusively to u and v because every key in our key pool may be allocated to multiple nodes; hence, they cannot be used for encrypting any message that is private to u and v. Thus the goal of our algorithm is to establish a key, S, that is known exclusively to u and v. The basic idea underlying the establishment of such a key S is as follows: The sender node splits S into multiple shares using an appropriate secret sharing scheme. The sender then transmits to the recipient node all these shares, using a different logical path for each share. The recipient node then reconstructs S after it receives all (or a certain number of) the shares.

3.2 Detailed Protocol Description

We now discuss *key pre-distribution*, *logical path discovery*, and *pairwise key establishment* in detail.

Notations

- u, v (in lower case) are principals such as nodes.
- R_u is the set of keys that node u possesses.
- I_u is the set of key ids corresponding to the keys in R_u.
- $|I_u|$ is the size of the set I_u.
- R_{uv} is the intersection of R_u and R_v, i.e., $R_{uv} \stackrel{def}{=} R_u \cap R_v$.

3.2.1 Key Pre-distribution

In the *key pre-distribution* phase, the off-line key server loads each node u with m distinct keys from the key pool P of l keys $\{k_1, k_2, \ldots, k_l\}$ prior to the formation of the ad hoc network. A deterministic algorithm is used to decide the subset of keys R_u allocated to node u. Specifically, for each node with a unique node id, the key server generates m distinct integers between 1 and l using a pseudo-random number generator upon the input of a node id. These m integers are the ids of the keys for this node. As a result, each key in the key pool has a probability of m/l to be assigned to each node.

In [10, 5], the key server *randomly* chooses m keys out of l keys; hence, a node u does not know what keys another node v possesses unless node v sends its key id set I_v to it. In contrast, our id-based key assignment scheme allows any node that knows another node's id v to determine I_v independently. Thus, our scheme is more communication-efficient.

Note that in our scheme all the nodes do not have to be initialized and join the network at the same time. Indeed, new nodes can be initialized in the same way as described above and join the network at any time. Moreover, our scheme does not require a key predistribution phase for every instance of network formation.

3.2.2 Logical Path Discovery

The *logical path discovery* process is necessary when a node wants to exchange messages securely with other nodes in the network.

We say there are *logical* paths between two nodes when (i) the two nodes share one or more keys in their key sets. We call such paths *direct* paths. (ii) the two nodes do not share any keys, but through other intermediate nodes they can exchange messages securely. We call such paths *indirect* paths and call the involved intermediate nodes *proxies*.

In our scheme, it is straightforward to find logical paths between two nodes. Since the key pre-distribution algorithm is public and deterministic, without proactively 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 node v, because node u can use a neighbor (say x) which shares keys with both of them 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 key exists between node u and node v. To transmit a message M to node v securely, node u executes the following steps.

$$u \rightarrow x : \{M\}_{k_{ux}}, x \rightarrow u : \{M\}_{k_{xv}}, u \rightarrow v : \{M\}_{k_{xv}}.$$

From this example, we can see that a *proxy* node acts as a translator between nodes. For convenience, we say node x *translates* the message M.

We call node x in the above example node u's *one-hop proxy* to v because x is one hop away from u. 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. Let z_1, s_1, s_2 be the number of keys in R_{uv}, R_{ux}, R_{xv} respectively. We say that node u and v have z_1 *direct* paths and $z_x = \min(s_1, s_2)$ *indirect* paths via x. There could be zero, one or many logical paths between two nodes. 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.

3.2.3 Pairwise Key Establishment

We now describe an approach whereby two nodes can establish a pairwise key that is *exclusively* known to the two nodes with overwhelming probability. We note the common keys, if any, between any two nodes after the *key predistribution* phase, are not *exclusively* held by them, because every key in the key pool is statistically allocated to $\frac{m \cdot N}{l}$ nodes, given l, m, and the network size N. Therefore, the keys in the key pool cannot be used directly for *private* communications between two nodes; otherwise, a compromised node or the coalition of compromised nodes may be able to compromise the communications of many pairs of other nodes.

The main observation underlying our key establishment scheme is that a sender node can split the to-be-established pairwise secret key into multiple shares, and then send them securely over multiple *logical* paths between itself and a recipient node. More specifically, the basic scheme involves five steps:

1. The sender node u first randomly generates the secret key S, then derives n shares $sk_1, sk_2, \cdots, sk_n$ from S using the following simple algorithm: it generates $n-1$ random strings $sk_1, sk_2, \cdots, sk_{n-1}$, $|S| = |sk_1| = \cdots = |sk_{n-1}|$, and then computes $sk_n = S \oplus sk_1 \oplus \cdots \oplus sk_{n-1}$, where $\oplus$ is the bitwise XOR operation. This scheme requires a recipient node to receive all these n shares to recover S using simple XOR operations, while no information about S can be determined with less than n shares. We shall discuss the choice of n in Section 3.2.4 and some alternative schemes that increase the availability of this basic scheme by using threshold secret sharing in Section 5.

2. Node u transmits all the shares to the recipient node v, using a different logical path for each share.

3. Node v computes the secret key S from the n received shares.

4. Node v sends back to node u a HELLO message, authenticated with S as the MAC key[1].

5. Node u verifies the HELLO message. The key establishment process is done if the HELLO message is correct; otherwise, node u aborts the process or tries again with a different set of logical paths after a certain time period.

Two types of logical paths, i.e., *direct* paths and *indirect* paths, are potentially used in the above process. The proxy nodes involved in the *indirect* logical paths in step 2 act as on-the-fly KDC servers (in parallel to [13]).

3.2.4 Determining The Number of Secret Shares

Generally, the greater the number of secret shares used in the pairwise key establishment process, the better security will the pairwise key achieve. However, using a greater number of secret shares requires a greater number of logical paths to be involved between two nodes, which leads to higher bandwidth and computational overheads. This is because any logical path may be used for securing at most one secret share for two nodes; otherwise, compromising one path would compromise multiple secret shares.

In general, between two nodes there are three classes of logical paths that can be used in a pairwise key establishment, although which classes are available depends on the application under consideration.

Class C_1 The first class, denoted by C_1, includes the *direct* paths based on the keys common to the two nodes. In our scheme, a sender node u knowing the id of the recipient node v can determine their common key set by itself. Let z_1 be the number of keys in R_{uv}, and sk_1 be the share generated by node u for class C_1. To deliver sk_1, node u computes the XORed key $k_{enc} = \text{XOR } \delta_i, \forall \delta_i \in R_{uv}$, then encrypts (with appropriate authentication) sk_1 with k_{enc}. (One may argue that the sender can generate z_1 shares and encrypt each share with a distinct key in R_{uv}. However, this approach incurs a higher communication cost without increasing security.)

Class C_2 The second class, denoted by C_2, includes *indirect* logical paths that use an intermediate node on the physical path between the two nodes as a proxy. Class C_2 logical paths can be easily found in ad hoc networks where the routing protocol in use facilitates the discovery of intermediate nodes between the two participants. For example, in the

[1]More precisely, node v uses $k_m = f_S(0)$ as the MAC key, and $k_p = f_S(1)$ as the pairwise key, where f is a pseudo random function [12]

Dynamic Source Routing (DSR) [14] protocol, the ids of all the intermediate nodes between the source and the destination are returned to the source in the ROUTING REPLY message. The source node can therefore choose the intermediate nodes that might act as proxies to securely forward some secret shares.

We employ the following *forwarding* algorithm. Let node x be a proxy node for node u and node v, as long as $R_{ux} \neq \emptyset$ and $R_{xv} \neq \emptyset$. Suppose there are s_1 and s_2 keys in R_{ux} and R_{xv} respectively and $z_s = \min(s_1, s_2)$, then the number of keys in R_{ux} and R_{xv} used to encrypt a share is z_s. More specifically, (i) node u generates a new secret sk_x, (ii) it then (randomly) selects z_s keys in R_{ux} to compute the XORed keys k^1_{ux}, (iii) it encrypts (with appropriate authentication) sk_x with k^1_{ux}, (iv) it sends the encrypted share to node x. Node x decrypts sk_x, re-encrypts it with the XORed key k^2_{xv} computed using any z_s keys in R_{xv} and sends the result to node v. Since node x is on the physical path from u to v, no extra message overhead is incurred in the use of such proxies. The number of such proxies is mainly determined by the topological distance between u and v. We denote the set of such proxies by Px_2.

Class C_3 The third class, denoted by C_3, includes the *indirect* logical paths that use nodes that do not belong to class C_2, i.e., nodes that are not on the path from u to v. Both the neighbors of the source node and the neighbors of the destination node are potential proxy nodes for logical paths in class C_3. The source node can discover the neighbors of the destination node via an explicit message exchange with the destination. Alternatively, it may be possible to extend the underlying routing protocol to provide this information to the source at the time of route formation. For example, in DSR, when the destination node receives the ROUTING REQUEST message from the source node, it can piggyback the ids of its neighbors that can act as proxies in the ROUTING REPLY message. Additionally, the intermediate nodes may also add their own neighbors to the ROUTING REPLY message if their neighbors can be proxies for the source and the destination. With this information, the source node can determine how many proxies and how many secret shares it can deliver using some or all of them. Basically, the source node can run a *forwarding* algorithm similar to that used for class C_2. The main difference is that it incurs some additional communication overhead because the proxy nodes are not on the physical path from the source to the destination. The number of such proxies is mainly determined by node density of the network and the distance between the source and the destination. We denote by Px_3 the set of proxy nodes used in class C_3 logical paths.

For a source node u and a destination node v, every key in their key sets may be used at most once for delivering one secret share. Therefore, node u selects only one of the proxies that contribute the same keys. The source should select proxies with the goal of minimizing the performance overhead. In Fig. 1 we show the algorithm used by the source node to determine the total number of secret shares n, given all the candidate proxy sets Px_2, Px_3 and the desired security level p^0_w. The algorithm evaluates the security level p_w in each iteration based on the security analysis in Section 4.1. If $p_w \leq p^0_w$, the algorithm terminates and returns n. Otherwise, if the desired security level cannot be achieved with all the available proxies, the algorithm reports n and the finally achieved security level p_w.

In the latter case, the (sender) peer may abort the process until a later time when network conditions change, e.g., when it has more neighbors. Under certain circumstances, it may be better for the peers to establish a *temporary* pairwise key based on the currently available secure paths, even though this temporary key does not satisfy the required security level. At a later time, the two peers may discover additional proxies due to their movements. Therefore, the sender can use these proxies to deliver some new secret shares to the destination. The final pairwise key is a combination (XORing) of the temporary key and these new shares. In other words, a pairwise key can be established and enhanced *incrementally* due to node mobility.

4 Security and Performance Analysis

4.1 Security Analysis

For the sake of simplicity and clarifying the presentation, we assume that the underlying encryption scheme is secure and define the security[2] of our scheme as the probability p_w that a coalition of up to w nodes can compromise the established pairwise key. Suppose there are w compromised nodes, $r_1, \cdots, r_w$, that collude by sharing their key sets. Therefore, they have the set of keys $\Sigma = \cup_{i=1}^{w} R_{r_i}$, which allows them to obtain the secret shares via the logical paths secured by $K \subseteq \Sigma$.

Let us assume node u and node v have z_1 direct paths (i.e., z_1 keys in R_{uv}) that are used for securing the class C_1 share. Recall in our key predistribution scheme each key in the key pool has a probability of m/l to be chosen by each node. Consider a key K in the key pool and any coalition of w nodes, the probability p_c that key K is contained in the union of the key sets of the w nodes is $p_c = (1 - (1 - \frac{m}{l})^w)$. Therefore, the probability p_{w1} that the coalition of w nodes cover all the z_1 keys in R_{uv} is

$$p_{w1} = p_c^{z_1} = (1 - (1 - \frac{m}{l})^w)^{z_1}. \quad (1)$$

Let z_2 be the number of *indirect* paths which we compute in Step 4 of the algorithm in Fig 1. A secret share

[2]More precisely, here security refers to privacy or secrecy.

Algorithm
Input: u, v, Px_2, Px_3, p_w^0
Output: n – the number of secret shares
Method:
// z_1 – the number of *direct* paths, z_2 – the number of *indirect* one-proxy paths.

1. $n \leftarrow 0, z_1 \leftarrow 0, z_2 \leftarrow 0, I'_u \leftarrow I_u, I'_v \leftarrow I_v$.
2. Node u computes $I_{uv} = I_u \cap I_v$, and updates $I'_u \leftarrow I'_u - I_{uv}, I'_v \leftarrow I'_v - I_{uv}, z_1 \leftarrow |I_{uv}|$. If $z_1 \geq 1$, then $n \leftarrow 1$.
3. $P \leftarrow Px_2$.
4. From all the candidate proxy nodes in P, node u randomly chooses a node, say x.
   Let $I_1 \stackrel{def}{=} I_{ux} \cap I'_u$ and $I_2 \stackrel{def}{=} I_{xv} \cap I'_v$.
   If $I_1 \neq \emptyset$ and $I_2 \neq \emptyset$, then
   Select x as a proxy node. Let $z_s \leftarrow \min(|I_1|, |I_2|)$,
   then node u deletes z_s ids in I_1 from I'_u and deletes z_s ids in I_2 from I'_v.
   $n \leftarrow n + 1, z_2 \leftarrow z_2 + z_s$
   $P \leftarrow P - \{x\}$.
   Evaluate the security level p_w based on z_1 and z_2 using the formulae in Section 4.1. If $p_w \leq p_w^0$, **return** n.
5. Repeat Step 4 until P becomes empty.
6. $P \leftarrow Px_3$, repeat Step 4 and Step 5.
7. **return** n.

Figure 1. An algorithm for determining the number of secret shares in a pairwise key establishment.

through an indirect logical path involving a single proxy node is compromised when the compromised nodes have keys to decode either the transmission between the source node and the proxy node, or the transmission between the proxy node and the destination node. Therefore, the probability p_{w2} that the coalition of w nodes are able to decode all these z_2 secret shares is

$$p_{w2} = (1 - (1 - \frac{m}{l})^{2w})^{z_2}. \quad (2)$$

Thus, the security of the pairwise key is

$$p_w = p_{w1} \cdot p_{w2} \quad (3)$$

In addition, there is a lower bound on the security of the pairwise key, which occurs when a peer uses all the keys in its key set for securing secret shares. Let $p_c(w)$ be the probability that the key set of a legitimate node is completely covered by that of w colluding nodes. We have

$$p_c(w) = (1 - (1 - \frac{m}{l})^{w})^{m}. \quad (4)$$

Given the desired security level p_w^0 and w, we should select m and l so that $p_c(w) \leq p_w^0$.

4.2 Performance Analysis

We use the communication cost involved in a pairwise key establishment process as the metric to evaluate the performance of the basic scheme. We do not consider the computational cost because this process only involves a few inexpensive symmetric key cryptography operations. We also do not consider the energy consumption of our protocol because it largely depends on the communication cost. Note that the cost is *one-time* (i.e., once and for all) because two nodes usually only have to establish their pairwise key once.

4.2.1 Communication Cost

The communication cost C_s of our protocol is the total number of hops traversed by the n secret shares in a pairwise key establishment operation. Clearly, C_s increases with n and d the topological distance in hops between two peers, because all the secret shares are forwarded hop-by-hop along the route between the two peers. Let n_1 be the number of secret shares corresponding to C_1, n_2 corresponding to C_2, and n_3 corresponding to C_3 (when only considering one-hop proxies), then $C_s = d \cdot (n_1 + n_2 + n_3) + 2n_3 = dn + 2n_3$. Namely, those proxies on the route (corresponding to C_2) will automatically forward, after decryption and re-encryption, a message to the recipient.

Below, we use n as the indication of the communication cost, and study the factors that affect the communication through detailed simulations. Note we only consider the *one-hop* proxies for C_3 in this performance study.

In our simulations, we consider a network space of

$2000m \times 2000m$, and a default network size of 200 nodes that are randomly distributed in the space[3]. The transmission range of a node is $250m$. Based on these settings every node has 8.7 immediate neighbors on average. We generate a route for each pair of connected peers based on the shortest path routing algorithm. All the results have 95% confidence intervals that are within 5% of the reported values.

Number of Secret Shares and its Composition To evaluate the communication costs of our protocol, we ran the algorithm in Fig. 1 to determine the number of each type of proxies used for achieving the target security level $p_w = 10^{-6}$ and $w = 10$ (Canetti *et al* [4] suggest that a unforgeability probability of 10^{-6} is suitable for many applications.). Fig. 2 shows the average number of secret shares n used in a pairwise key establishment for any two peers at a distance of d hops. The figure also shows the composition of n, i.e., the number of shares that are attributed to the different classes of logical paths (C_1, C_2, and C_3) used by our algorithm.

From Fig. 2, we can make the following observations. First, the average number of shares corresponding to C_1 is around one. As long as two nodes have at least a common key in their pre-deployed key sets, they will be able to deliver one secret share through this direct path. Consider a key K in the key pool. The probability that K is allocated to both the peer nodes is $(m/l)^2$. Then the probability that none of the keys in the key pool is allocated to both the nodes is $(1-(m/l)^2)^l$. Thus, the probability p_s that two nodes have at least one key in common is

$$p_s = 1 - (1 - (m/l)^2)^l. \quad (5)$$

Let $m = 200$ and $l = 10000$, then $p_s = 0.98$. p_s is independent of the distance between nodes. This explains why in Fig. 2 the number of shares corresponding to C_1 is constant. Note that the expected value of z_1 (the number of common keys between two nodes) also depends on m and l and is given by $E(z_1) = l * (m/l)^2 = m^2/l$.

Second, the total number of secret shares required to achieve the desired security level is independent of the physical distance d between two peers. This is because once m and l are chosen, the security of a pairwise key depends on z_1 and z_2 according to Eqn.1, 2 and 3. Since $E(z_1)$ is independent of d, $E(z_2)$ is also independent of d when achieving a desired level of security, p_w^0. Thus, the total number of secret shares required is also independent of d. In addition, we note our protocol usually incurs a very small communication cost. In Fig. 2 we see only 3.8 shares (totally about 30 bytes if each share is 8 bytes) are necessary on average to establish a pairwise key that has the security level $p_w = 10^{-6}$ when $w = 10$.

[3]In our simulations, we used a static network because a pairwise key establishment usually takes a very short time relative to the velocities of mobile nodes

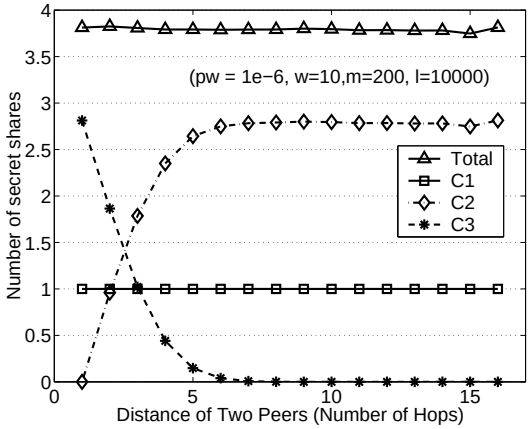

Figure 2. The number of secret shares and its composition as a function of distance (in hops) between two peers

Third, the number of secret shares corresponding to C_2 increases with d while the number of shares corresponding to C_3 decreases with d. As d increases, more intermediate nodes can act as proxies and hence more secret shares are delivered via proxies in Px_2 instead of via proxies in Px_3. This is because Px_2 proxies are selected in preference to Px_3 proxies in the algorithm in Fig. 1 and the algorithm terminates once it reaches the desired security. Fig. 2 shows that almost no indirect paths in C_3 are used in a pairwise key establishment for two nodes at a distance of more than 5 hops.

Impact of Node Density We evaluated the impact of node density on n by varying the number of nodes in the space $2000m \times 2000m$. The simulation results show that the impact of node density is very small. However, we found that some short-distance pairs in a very sparse network cannot establish a pairwise key that satisfies the required security level. This is because the peers cannot find enough proxies in the network. In this case, they may establish their pairwise key *incrementally* through movement as discussed in Section 3.2 or resort to other means (e.g., through physical contact [24]).

We note that in our scheme it is the node density rather than the absolute network size that matters. In a fixed space, the larger the network size, the more the number of proxies available. Hence, two nodes can establish a pairwise key more easily or establish a more secure pairwise key. In this sense, we consider our scheme to be scalable with the network size.

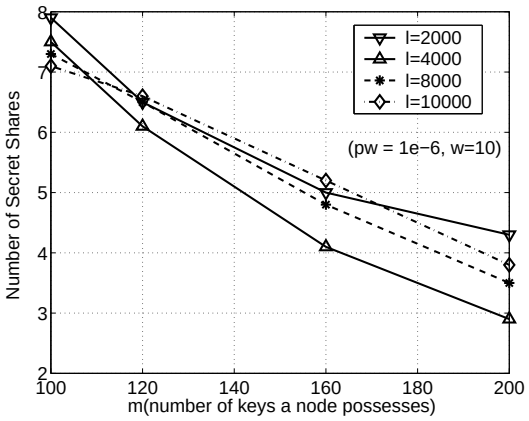

Figure 3. The impact of varying m and l on the number of secret shares required to satisfy the desired security ($p_w = 10^{-6}$).

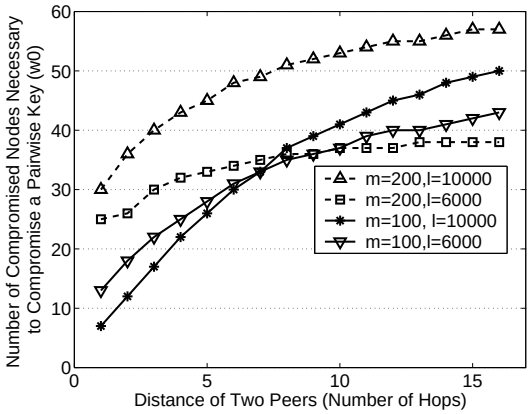

Figure 4. The number of colluding nodes necessary to compromise a pairwise key ($p_w = 10^{-6}$) when two nodes utilize all available C_1, C_2, C_3 logical paths between them.

Impact of Probabilistic Key Sharing Parameters Fig. 3 shows the impact of l and m on the necessary number (n) of secret shares used in a pairwise key establishment to achieve the security level $p_w = 10^{-6}$. We observe that for a fixed l, n generally decreases with m, while the impact of l seems to depend on m. In practice, this means a larger m should be chosen if the memory space of a node allows.

4.2.2 Storage Requirements

The storage requirements of our approach are determined by the number of keys held by a node, i.e., m. Clearly, the storage requirements are independent of the size of the network.

4.3 Security, Performance and Storage Tradeoff

In Fig. 4, we show the security strength of an established pairwise key when two nodes use *all* the available logical paths in classes C_1, C_2, and C_3 to establish their pairwise key. In the Y-axis, $w0$ denotes the number of colluding nodes that are needed to achieve a probability of $p_w = 10^{-6}$ for compromising a pairwise key established by two other nodes. From the figure we make the following observations. First, the security of the pairwise key increases with the distance d between the peers because more proxy nodes become available as d increases, leading to a larger number of logical paths. Second, increasing both m and l can improve the security of the pairwise key significantly, although increasing m results in larger storage requirements per node. For example, when $m = 200$ and $l = 10000$, $w0 = 45$ for two nodes at a distance $d = 5$. That is, 45 compromised nodes have to collude to be able to compromise a pairwise key established by two nodes at a distance of 5 hops, with a probability of 10^{-6}. This choice of m and l thus provides strong enough security for most applications.

In Fig. 5, we further show the impact of security level on the number of secret shares used for key establishment. We observe that achieving a higher security level requires a larger number of secret shares to be used and hence will incur a higher performance overhead. The figure shows that our protocol can be parameterized to trade performance for security, and vice versa, as is appropriate for the application under consideration.

5 Increasing The Availability and Robustness

In the basic scheme presented above, a recipient node needs to receive all the secret shares to recover the pairwise key. To increase the availability of this scheme, we can instead deploy a (k, n) ($k < n$) threshold secret sharing scheme [26, 16]. Such a threshold secret sharing scheme, based on polynomial interpolation, allows a node receiving any k (out of the n) shares to recover a pairwise key S, while no information about S can be determined with less than k shares. Since the underlying operations are typically very cheap, it is computationally affordable even for low-end devices.

There are two scenarios that may benefit from the use of the above threshold scheme. In the first case, a class C_3 proxy fails when it is involved in a pairwise key establishment process. For example, if a C_3 proxy node that is a neighbor of the destination node becomes unavailable due to node mobility or failure when a secret share that requires its translation arrives at the destination node, the pairwise

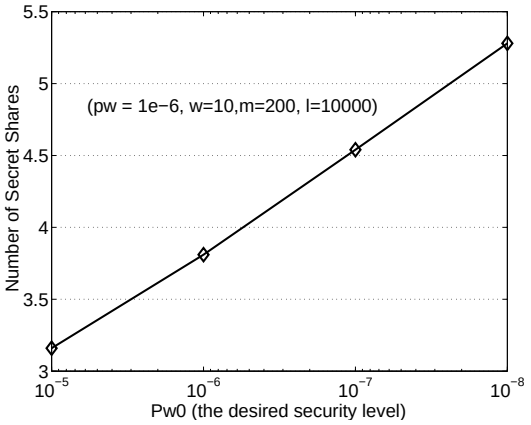

Figure 5. The number of secret shares used as a function of desired security level

key will not be established in our basic scheme. Using a threshold scheme addresses this issue because the peers can establish the pairwise key even without this missing share. However, since all the secret shares are potentially transmitted over the same physical path between the peers, if one of the intermediate nodes becomes unavailable, none of the shares will arrive at the destination. The threshold scheme does not help in this case. Actually, this means the route between two nodes is broken and needs to be re-established. Consequently, the pairwise key establishment process also needs to be restarted.

In the second case, a *compromised* node on a logical path intentionally drops or alters the share(s) going through it. Consider the scenario where nodes u wants to establish a pairwise key with node v. Let a compromised node x be one of their class C_3 proxies. Node x only receives the share that needs its translation because it is not on the physical path between u and v. There are two attacks node x could launch:

- Node x drops its share intentionally. This situation is the same as in the first case where the proxy node is unavailable due to mobility or failure. Therefore, using a threshold scheme will help to defend against this attack.

- Node x does not drop its share but alters the share to a random string. As a result, in the basic scheme the pairwise key node v recovers will be different from the one node u generates. Unless an appropriate (k, n) threshold scheme is deployed, node v may recover different pairwise keys. Moreover, in order to achieve better robustness in the case that an appropriate (k, n) threshold secret sharing scheme is adopted, it may be helpful to accompany with each share the hashes of all the shares.

We note that, under certain circumstances, a threshold scheme does not guarantee the establishment of a pair-wise key. For example, a compromised intermediate node drops all the shares or replaces all the shares with random strings. This is particularly true in our scheme because typically the same physical path is used to forward all the shares (i.e., a quite unique scenario). Actually, these are general attacks in ad hoc networks and there is no way to prevent a compromised node from launching such attacks. The only solution is to identify the malicious node and re-establish a route avoiding the node. In [18], Marti *et al* propose using a *watchdog* and a *pathrater* to mitigate this attack. In [1], Awerbuch *et al* propose an adaptive probing technique to detect Byzantine failure. In addition to these techniques, we may apply the following *multi-path* scheme to further mitigate this attack.

The multi-path scheme is applicable when there are multiple physically disjoint paths between two nodes. Basically, each path runs one instance of our scheme. As long as one of the paths work correctly, two nodes will be able to establish a pairwise key. Moreover, if multiple paths work correctly, the security of the established pairwise key could be further enhanced by combining the pairwise keys from multiple paths. We note many routing protocols propose to use multiple paths between nodes. For example, for nodes that are one hop or two hops away, they can find multiple disjoint paths by simply exchanging their neighbor sets. In routing protocols such as DSR [14], multiple paths may be available after the *Route Discovery* phase is done. Other examples of multi-path protocols include MP-DSR [17] and TORA [22].

To summarize, we showed that various approaches can be used to increase the availability and the robustness of the basic scheme, at the price of increased computational and communication costs because more shares need to be generated and forwarded. In practice, the values of k and n in a (k, n) threshold scheme and the number of disjoint multipaths to use are both application dependent. We note that although the basic scheme is less robust to various attacks, it provides the necessary security guarantee. That is, the peers can determine if the pairwise key establishment process succeeds or not by exchanging the verifying HELLO message in the last step of the basic scheme.

6 Conclusions

In this paper, we have presented a scalable protocol for pairwise key establishment in ad hoc networks. The established pairwise key can be used to bootstrap trust relationship between nodes. The design of our protocol is based on

a novel combination of threshold secret sharing and probabilistic key sharing. Our protocol has the following properties:

- It is fully distributed – no on-line key server is required.
- It is computationally efficient – it relies only on symmetric cryptography.
- It is storage scalable – the storage requirements per node depend on the desired level of security and are independent of the size of the network.
- It is secure to a collusion attack by up to a certain number of compromised nodes.

Acknowledgements We thank Peng Ning, Yongge Wang and the anonymous reviewers for their valuable comments and suggestions.

References

[1] B. Awerbuch, D. Holmer, C. Nita-Rotaru and H. Rubens. An On-Demand Secure Routing Protocol Resilient to Byzantine Failures. In ACM Workshop on Wireless Security (WiSe) 2002.

[2] A. Bagchi, A. Chaudhary, M. Goodrich, and S. Xu. Constructing Disjoint Paths for Secure Communication. DISC'03, to appear.

[3] M. Brown, D. Cheung, D. Hankerson, J. Hernandez, M. Kirkup, and A. Menezes. PGP in Constrained Wireless Devices. In 9th USENIX Security Symposium, pages 247261, August 2000.

[4] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, B. Pinkas, Multicast Security: A Taxonomy and Some Efficient Constructions. IEEE Infocom'99. Long Beach, CA, USA, Oct. 2001.

[5] H. Chan, A. Perrig, D. Song. Random Key Predistribution Schemes for Sensor Networks. To appear in Proc. of the IEEE Security and Privacy Symposim 2003, May 2003.

[6] D. Dolev. The Byzantine generals strike again. In J. of Algorithms, 3:14-30, 1982.

[7] Y. Desmedt and Y. Wang. Perfectly secure message transmission revisited. In Advances in Cryptology EUROCRYPT '02, Lecture Notes in Computer Science (LNCS). Springer-Verlag, 2002.

[8] D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly Secure Message Transmission. JACM, (40)1, 1993, pp 17-47.

[9] P. Erdos, P. Frankl, and Z. Furedi. Families of Finite Sets in Which no Set Is Covered by the Union of r Others. Israel J. Math. 51(1985), 75-89.

[10] L. Eschenauer and V. Gligor. A Key-Management Scheme for Distributed Sensor Networks. In Proc. of ACM CCS 2002

[11] M. Franklin and R. Wright. Secure communincation in minimal connectivity models. In Journal of Cryptology, 13(1):9-30, 2000.

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

[13] L. Gong. Increasing Availability and Security of an Authentication Service. IEEE Journal on Selected Areas in Communications, 11(5):657–662, 1993.

[14] D. Johnson, D. Maltz, Y. Hu, J.Jetcheva. The Dynamic Souce Routing Protocol for Mobile Ad Hoc Networks. Internet-Draft, draft-ietf-manet-dsr-07.txt, February 2002.

[15] J. Kohl and B. Neuman. The Kerberos Network Authentication Service (V5). RFC 1510, September 1993.

[16] S. Kothari. Generalized Linear Threshold Scheme. Advances in Cryptology - CRYPTO'84, LNCS 196, pp 231-241, 1984.

[17] R. Leung, J. Liu, E. Poon, Ah-Lot. Chan, B. Li. MP-DSR: A QoS-Aware Multi-Path Dynamic Source Routing Protocol for Wireless Ad-Hoc Networks. In Proc. of 26th Annual IEEE Conference on Local Computer Networks (LCN 2001), 132-141.

[18] S. Marti, T. Giuli, K. Lai, M. Baker. Mitigating Routing Misbehavior in Mobile Ad Hoc Networks. In Proc. of ACM MOBICOM, 2000.

[19] C. Mitchell and F. Piper. Key Storage in Secure Networks. Discrete Applied Mathematics. 21(1988). pp 215-228.

[20] R. Needham and M. Schroeder. Using Encryption for Authentication in Large Networks of Computers. In Commuinications of the ACM 21(12): 993-999, 1978.

[21] D. Otway , O. Rees, Efficient and Timely Mutual Authentication, Operating Systems Review, 21 (1987), 8-10.

[22] V. Park and S. Corson. Temporaly-ordered routing algorithm. Internet Draft, August 1998.

[23] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. Tygar. SPINS: Security Protocols for Sensor Networks. In Seventh Annual ACM International Conference on Mobile Computing and Networks(Mobicom 2001), Rome Italy, July 2001.

[24] F. Stajano and R. Anderson. The Resurrecting Ducking: Security Issues for Ad-hoc Wireless Networks. In the 7th International Workshop on Security Protocols,1999.

[25] D. Stinson and R. Wei, Some New Bounds for Cover-Free Families. In J. Combin. Theory A, 90(2000), 224-234.

[26] A. Shamir. How to Share a Secret. Comm. ACM, 22(11):612–613, 1979.