Establishing Pair-wise Keys For Secure Communication in Ad Hoc Networks: A Probabilistic Approach

Sencun Zhu[1⋆], Shouhuai Xu[2], Sanjeev Setia[1⋆⋆], and Sushil Jajodia[1]
[1]Center for Secure Information Systems, George Mason University, Fairfax, VA 22030
[2]Dept. of Information and Computer Science, Univ. of California at Irvine, Irvine, CA 92697
Email: {szhu1,setia,jajodia}@gmu.edu, shxu@ics.uci.edu

Abstract. We present a scalable distributed protocol that enables two mobile nodes in an ad hoc network to establish a pairwise shared key on the fly, without requiring the use of a 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 operations. We show that a pairwise key established between two nodes using our protocol is secure against a collusive attack by a certain number of compromised nodes, and that our protocol can be parameterized to meet the appropriate 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 network cannot afford the deployment of public key cryptosystems due to their high computational and communication overheads and storage constraints. For instance, Brown *et al* [1] have reported that a 512-bit RSA signature generation takes 2-6 seconds on a RIM Pager and a Palm Pilot, and Perrig *et al* [12] report that a current generation sensor node has just 4500 bytes for security and the application. Consequently, it is necessary to explore approaches that are based on the use of symmetric key cryptography.

The problem of facilitating secure communication based only on symmetric key cryptography has been investigated extensively. The seminal idea due to Needham and Schroeder [9] is to deploy a server that acts as a key distribution center(KDC) and pre-distributes a single key to an individual participant. To communicate securely, a pair of participants have to obtain fresh session keys from the on-line server. Although this scheme and other KDC based schemes such as Kerberos [7] and Otway-Rees [10] have been widely deployed in wired networks, the requirement of an on-line key server poses a problem for ad hoc networks that are characterized by dynamic topology changes and node failures, e.g., due to battery exhaustion.

To avoid the use of such an on-line server, Mitchell and Piper [8] proposed a solution based on probabilistic key sharing that ensures that the keys known to a pair of participants (i.e., the intersection of their own keys) is not covered by a certain threshold number of other participants. (Clearly, the trivial solution that lets each pair of participants hold a different key is sufficient for this purpose but not scalable.) Although this approach directly enables secure communication between any two participants, the storage complexity imposed on each participant is unaffordable in the context of the present work (cf. [13] for the lower bounds). A similar scenario has happened in the context of multicast authentication: a natural application of [3] would ensure multicast authentication against a threshold number of colluding participants, but a much more light-weight solution was obtained in [2] which provides some practically affordable security guarantee.

⋆ contact author
⋆⋆ also with Dept. of Computer Science, George Mason University

In this paper, we present a scalable and distributed protocol that enables two mobile nodes in an ad hoc network to establish a pairwise shared key on the fly, without requiring the use of a 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 operations and is thus computationally efficient.

A pairwise key established between two nodes using our protocol is *exclusively known* to the peers with overwhelming probability, and it is secure against a collusive attack by a certain number of compromised mobile nodes. We study the performance and the security aspects of our protocols through both analysis and detailed simulation, and show that the protocol can be parameterized to meet the appropriate levels of security, performance and storage for the application under consideration.

The rest of this paper is organized as follows. We first present our pairwise key establishment protocol in Section 2. In Section 3, we analyze the performance and security of the protocol. Finally, we conclude this work in Section 4.

2 The Pairwise Key Establishment Protocol

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

2.1 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. Second, the resources of a node, such as power, computation and communication capacity, are relatively constrained. 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 assume all the nodes in a network are equally trusted. No central key server exists in the formed network, whereas it may exist off-line to initiate the mobile 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. Moreover, all the compromised nodes may collude to launch attacks. Finally, we assume all the nodes are equally likely to be compromised.

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

Before the deployment of the 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 from a key server. Note that this phase occurs before the deployment of the network, and the key server stays off-line after this phase is complete. Keys are allocated to each node using a probabilistic scheme that enables every pair of nodes to share one or more keys with a certain probability. The keys directly shared between any two nodes can thus be used to encrypt messages exchanged by the nodes. Even if two nodes do not share any keys directly, our probabilistic key sharing scheme enables them to communicate securely using logical secure channels that can be established using the keys pre-allocated to nodes in the key pre-distribution phase (the logical path establishment process is discussed in more detail in Section 2.2).

Now consider two nodes, u and v that wish to communicate privately. Note that 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, sk, that is *known exclusively* to u and v.

To establish sk, a sender node (say u) splits sk into multiple shares using a *threshold secret sharing* scheme. It then transmits to the recipient (v) all these shares, using *a separate secure logical channel* (established using the logical path establishment process) for each share. The recipient node then reconstructs sk after it receives all (or a certain number of) the shares.

2.2 Detailed Protocol Description

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

Notation In this discussion, we use the following notations.

- u, v are principals such as communicating 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 and $|sk|$ is the size of the key sk.

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. 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 node u.

Logical Path Establishment The *path establishment* procedure is executed when a node wants to securely exchange messages 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. 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 design, it is straightforward 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 proactively exchanging the set of its key ids with others, a node knowing the ids of its neighbors[1] 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 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.

$$u \longrightarrow x : \{M\}_{k_{ux}}$$
$$x \longrightarrow u : \{M\}_{k_{xv}}$$
$$u \longrightarrow v : \{M\}_{k_{xv}}$$

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

[1] A node can obtain its neighborhood information from lower layers. Many ad hoc routing protocols [11] provide neighborhood knowledge.

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 respectively. If u and v do not have any *direct* paths or *one-hop* proxies to each other, they can resort to a proxy node of multiple-hop away. Note that it is also possible to establish a logical path with multiple proxies involved. For example, if there is a shared key between u and x, between x and y, between y and v respectively, u and v can establish a logical path as well.

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. Note that we do not consider using multi-proxy paths in this work because a multi-proxy path incurs much higher performance overhead but weaker security than a direct path or a one-proxy path (more keys are involved in the path and hence compromising any one of them will compromise the delivered message).

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 pre-distribution* 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.

Our scheme is adapted from the distributed authentication scheme by Gong [5], which utilizes a set of key distribution centers (KDCs) [9] to achieve better security and availability. However, the setting we have in this paper is very different from that in [5]. We do assume the existence of a key server, but this server is not involved after the key pre-distribution phase. In contrast, the set of servers in [5] are involved whenever a pair of principals need to establish a session key.

The key 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 secure them with the multiple *logical* paths between itself and a recipient node. More specifically, the scheme involves five steps:

1. The sender node u first randomly generates the secret key ks, then derives n shares $sk_1, sk_2, ..., sk_n$ from ks using the following simple algorithm: it generates $n-1$ random strings $sk_1, sk_2, ..., sk_{n-1}$, $|sk| = |sk_1| = ... = |sk_{n-1}|$, and then computes $sk_n = ks \oplus sk_1 \oplus ... \oplus sk_{n-1}$, where $\oplus$ is an XOR operation. This scheme requires a recipient node to receive all these n shares to recover ks using simple XOR operations, while no information about ks can be determined with less than n shares. We shall discuss the choice of n in Section 2.2 and the alternative schemes that can increase the availability of this scheme in Section 2.3.
2. Node u secures each share with a separate logical path (or multiple logical paths) and sends all the shares to the recipient node v.
3. Node v computes the secret key ks from the n received shares.
4. Node v sends back to node u a HELLO message, authenticated with ks as the MAC key[2].
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 in a manner similar to the on-the-fly (KDC) servers in [5].

[2] More precisely, node v uses $k_m = F_{ks}(0)$ as the MAC key, and $k_p = F_{ks}(1)$ as the pairwise key, where F is a pseudo random function [4]

Determining The Number of Secret Shares Clearly, the more the secret shares used in the key establishment scheme, the more secure the secret key will be. However, using a larger number of secret shares requires a larger number of logical paths to be established between two peers, leading to higher bandwidth and computational overhead. This is because any key that is used in establishing logical paths may be used for securing at most one secret share for the two peers; otherwise, compromising one key would compromise multiple secret shares.

We consider the use of three classes of logical paths between two peers. The first class, denoted as C_1, includes the *direct* paths based on the common keys between the two peers. In our scheme, a sender node u knowing the id of the recipient node v can independently determine their common key set. Let $R_{uv} \stackrel{def}{=} R_u \cap R_v$, and let there be z_1 keys in R_{uv}. Let the share generated by node u be sk_1. To securely deliver sk_1, node u computes the XORed key $k_{enc} = \text{XOR } \delta_i, \forall \delta_i \in R_{uv}$, then encrypts sk_1 with k_{enc}.

The second class, denoted as C_2, includes the *indirect* logical paths based on the intermediate nodes on the path between the two peers. In an ad hoc network, before a node u sends a message to another node v, it establishes a route using an algorithm that typically involves network-wide flooding [6, 11]. For example, in the dynamic source routing (DSR) [6] 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 can act as proxies to forward some secret shares.

We employ the following *forwarding* algorithm. Let node x be a proxy node for node u and node v, and let $R_{ux} \stackrel{def}{=} R_u \cap R_x$ and $R_{xv} \stackrel{def}{=} R_x \cap R_v$. Since node x is a proxy for u and v, $R_{ux} \neq \emptyset$ 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} that are used to encrypt a share is z_s. More specifically, node u (i) 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 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 in the 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 as Px_2.

The third class, denoted as C_3, includes the *indirect* logical paths based on nodes that do not belong to the second class, i.e., nodes that are potentially not on the path from u to v. In a DSR-like routing protocol, 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. Similarly, the intermediate nodes can also add their neighbors to the ROUTING REPLY message if the neighbors can be proxies for the source and the destination. Of course, the source node can also utilize its own neighbors as proxies. With all this information, the source node can determine how many proxies and how many secret shares it can deliver through 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 in the 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. For a sparse network, a node may use its multiple-hop proxies. We denote the set of such proxies as Px_3.

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 complete algorithm that a source node runs 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 round of the loop. If $p_w \leq p^0_w$, the algorithm terminates and returns n. If the algorithm uses up all the proxies known to a node and p_w is larger than p^0_w, the algorithm reports a failure. In this case, the (sender) peer can either find more proxy nodes from nodes that are multiple hops away and re-run the algorithm, or it can re-run the algorithm at a later time when network conditions change, e.g., when it has

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. From all the candidate proxy nodes in Px_2, 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.
   $n \leftarrow n + 1$, $z_2 \leftarrow z_2 + z_s$
   $Px_2 \leftarrow Px_2 - x$.
   Evaluate the security level p_w based on z_1 and z_2 using the formulae in Section 3.1. If $p_w \leq p_w^0$, **return** n.
4. Repeat Step 3 until Px_2 becomes empty.
5. $Px_2 \leftarrow Px_3$, repeat Step 3 and Step 4.
6. **return** FAILURE

Fig. 1. The algorithm for determining the number of secret shares used in pairwise key establishment.

more neighbors. We describe the metrics for evaluating the security level of a to-be-established pairwise key in Section 3.1. Note that we can simply modify this algorithm to return the selected proxy sets and key sets as well.

2.3 Discussion

In the basic scheme presented above, a recipient node needs to receive all the secret shares to recover the secret key, which might be too strong a requirement. For example, if a proxy node that is a neighbor of the destination node becomes unavailable due to node mobility or power failure when a secret share that requires its translation arrives at the destination node, the source and the destination will fail in establishing the secret key, despite the efforts they have already made. To increase the availability of the basic scheme, a (k,n) ($k < n$) threshold secret sharing scheme [14] can be deployed. Moreover, we note that if a compromised node (that is yet undetected) is selected as a proxy, it might disrupt the pairwise key establishment by modifying the secret share(s) through it. To detect the compromised node and further increase the robustness of the basic scheme, we can apply the cross-checksum scheme proposed by Gong [5]. Using these alternative schemes instead of the basic scheme does not affect our design very much; indeed, these schemes only require more secret shares than the basic scheme does under the same security requirement.

3 Security and Performance Analysis

3.1 Security Analysis

For the sake of simplicity and clarifying the presentation, we assume that the underlying encryption scheme is secure and define the security 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 allow 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 that are used for forwarding one secret share. For every key in the key pool and any coalition of w nodes, the probability p_c that the key 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 these z_1 paths (i.e., keys) 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* one-proxy paths which we compute in the Step 3 of the algorithm in Fig 1. A secret share 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

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

We note there is an upper 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 revoked 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 $p_c(w) \leq p_w^0$.

3.2 Performance Analysis

Computational Costs The metrics used to evaluate the performance of our protocol are the computational and communication costs of key establishment. The main computational cost is the cost of encrypting and decrypting the n secret shares during the pairwise key establishment phase. For a secret share that is transmitted using a direct logical path between the two peers, there is a single encryption at the source and a single decryption at the destination. For a secret share that is delivered via a one-proxy (indirect) logical path, there are two encryptions (one at the source and one at the proxy) and two decryptions (one at the proxy and one at the destination). In the worst case, all the n secret shares are delivered through one-proxy paths. Consequently, the total number of encryptions and decryptions is $4n$. Note that the encryptions and decryptions involved are inexpensive symmetric key operations.

Communication Costs 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 delivered 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 $C_s = d \cdot (n_1 + n_2 + n_3) + 2n_3 = dn + 2n_3$.

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 (unless otherwise mentioned) that are randomly distributed in the space[3]. The transmission range of

[3] In our simulations, we used a static network instead of a mobile network because a pairwise key establishment usually takes a very short time relative to mobile node velocities; therefore, we considered a static network corresponding to a snapshot of the ad hoc network at the time of the event.

a node is $250m$. 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.

The Composition of the Number of Secret Shares To evaluate the communication costs of our protocol, we ran the algorithm in Fig. 1 to determine the number of each type of proxy used for achieving the target security level $p_w = 10^{-6}$ and $w = 10$ (Canetti *et al* [2] suggest that a covering probability of $2^{-20}(\approx 10^{-6})$ is suitable for some applications.). Fig. 2 shows the number of secret shares n used for 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 can be attributed to the different classes of proxies (C_1, C_2, and C_3) used by our algorithm.

From Fig. 2, we can make the following observations. First, the number of shares corresponding to C_1 is one, since at most one secret share is delivered through C_1. Second, the communication cost corresponding to proxies in C_2 increases with d because more intermediate nodes can act as proxies and hence more secret shares are delivered via proxies in C_2. Third, the communication cost corresponding to C_3 first increases and then decreases with d. For two peers that are neighbors ($d = 1$), their one-hop neighbor sets overlap; therefore, the number of distinct proxy nodes available for them is smaller than in the case when the peers are two or three hops away from each other[4]. However, since logical paths in C_2 are selected in preference to those in C_3, using a larger number of logical paths from C_2 as d increases results in fewer logical paths from C_3 because the algorithm in Fig. 1 terminates once it reaches the desired security level. Fourth, we observe that the total number of secret shares required to achieve the security level is almost constant, independent of the physical distance between two peers.

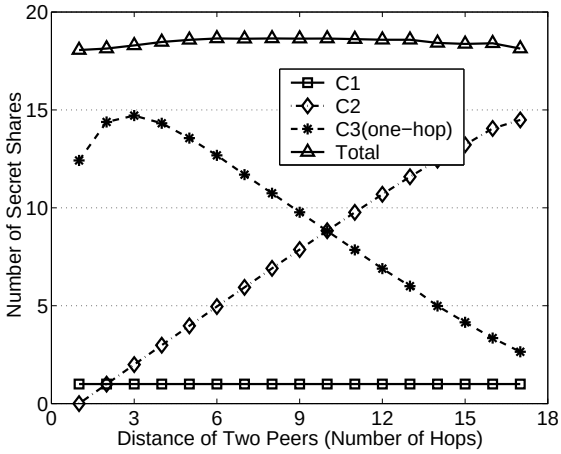

Fig. 2. The number of secret shares as a function of distance (in hops) between two peers and its composition

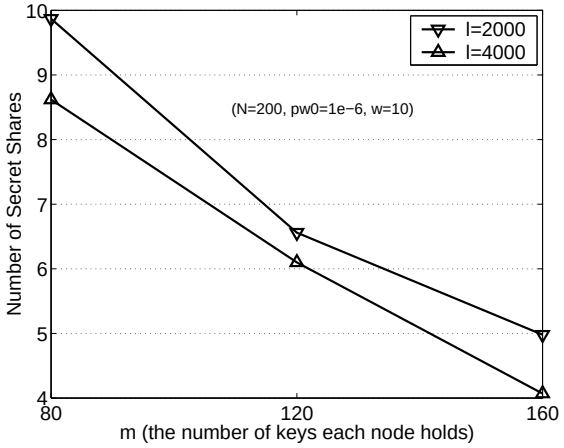

Fig. 3. The impact of varying m and l on the number of secret shares generated

Impact of Node Density In our simulations, we also evaluated the impact of node density on n by varying N, the number of nodes in our fixed space. Our results show the impact of changing the node density is very small. However, we noticed in our simulations that some pairs in a network with a very low node density cannot establish a pairwise key that satisfies the required security level. This is because the peers cannot find enough proxies in the network due to network partition.

[4] Actually in this figure when $d = 1$ or $d = 2$, a few 2-hop proxies of a source node serve one or more secret shares.

Impact of Probabilistic Key Sharing Parameters In Fig. 3 we show the impact of l and m on n, the number of secret shares used for key establishment. We observe that (1) for a fixed l, n decreases with m; (2) for a fixed m, n decreases with l; (3) m has a larger impact on n than l has.

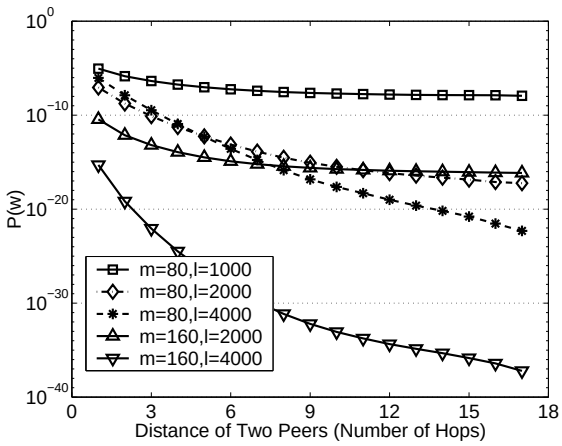

Fig. 4. The security of the pairwise key as a function of m and l ($w = 10$)

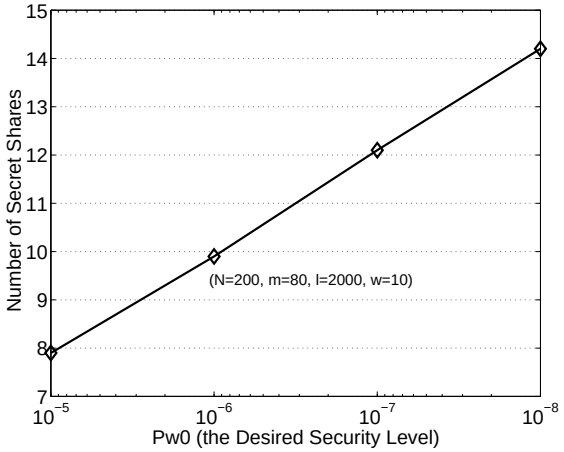

Fig. 5. The number of secret shares used as a function of desired security level

Storage Requirements The storage requirements of our approach are determined by the number of keys held by a node, m. Clearly, the storage requirements are independent of the size of the network. As discussed in this section, the parameters m and l impact both the security and performance of our protocol and should be selected based on the application under consideration.

3.3 Security and Performance Tradeoff

In Fig. 4, we plot p_w for $w = 10$ as a function of m and l when the peers use *all* the proxies in classes C_1, C_2, and C_3 for establishing the pairwise key. We make the following observations. First, the security of the pairwise key increases with the distance d between the peers because they can use a larger number of secret shares. Second, increasing both m and l can improve the security of the pairwise key significantly, while m has a larger impact than l has. For example, when $m = 160$ and $l = 4000$, the probability that 10 colluding nodes can compromise the pairwise key established by the peers at a distance of 5 hops is about 10^{-27}.

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, under the same simulation setting. 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.

4 Conclusions

In this paper, we have presented a scalable protocol for pairwise key establishment in ad hoc networks. The design of our protocol is base 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 are independent of the size of the network.
– It can be shown to be secure to a collusive attack by a certain number of compromised nodes.

As future work, we plan to further study the robustness and availability of our pairwise key establishment scheme.

References

1. 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.
2. 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.
3. 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.
4. O. Goldreich, S. Goldwasser, and S. Micali, How to Construct Random Functions, Journal of the ACM, vol. 33, no. 4, 1986, pp 210-217.
5. L. Gong. Increasing Availability and Security of an Authentication Service. IEEE Journal on Selected Areas in Communications, 11(5):657–662, 1993.
6. 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.
7. J. Kohl and B. Neuman, The Kerberos Network Authentication Service (V5). RFC 1510, September 1993.
8. C. Mitchell and F. Piper. Key Storage in Secure Networks. Discrete Applied Mathematics. 21(1988). pp 215-228.
9. R. Needham and M. Schroeder, Using Encryption for Authentication in Large Networks of Computers. In Commuinications of the ACM 21(12): 993-999, 1978.
10. D. Otway , O. Rees, Efficient and Timely Mutual Authentication, Operating Systems Review, 21 (1987), 8-10.
11. Charles Perkins. Ad hoc On Demand Distance Vector (AODV) Routing, Internet draft, draft-ietf-manet-aodv-00.txt.
12. 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.
13. D. Stinson and R. Wei, Some New Bounds for Cover-Free Families. In J. Combin. Theory A, 90(2000), 224-234.
14. A. Shamir. How to Share a Secret. Comm. ACM, 22(11):612–613, 1979.