An Efficient Scheme for Establishing Pairwise Keys for Wireless Sensor Networks

Abhishek Gupta, Pavan Nugehalli and Joy Kuri
Centre for Electronics Design and Technology
Indian Institute of Science, Bangalore
{agupta, pavan, kuri}@cedt.iisc.ernet.in

Abstract—This paper addresses the problem of secure path key establishment in wireless sensor networks that uses the random key pre-distribution technique. Inspired by the recent proxy-based scheme in [1] and [2], we introduce a friend-based scheme for establishing pairwise keys securely. We show that the chances of finding friends in a neighbourhood are considerably more than that of finding proxies, leading to lower communication overhead. Further, we prove that the friend-based scheme performs better than the proxy-based scheme both in terms of resilience against node capture as well as in energy consumption for pairwise key establishment, making our scheme more feasible.

I. INTRODUCTION

In the last few years, wireless sensor networks (WSNs) have become a very actively researched area. The impetus for this spurt of interest were developments in wireless technologies and low-cost VLSI, that made it possible to build inexpensive sensors and actuators. Each such device has limited computational power, memory and energy supply. Nevertheless, because of the low cost, such devices can be deployed in large numbers, and can thereafter form a sensor network [3], [4].

A typical sensor node contains some sensors (light, temperature, acceleration etc.), a radio chipset for wireless communication, an EEPROM chip for logging sensor data, a node-to-host communication interface (typically a serial port), and a microcontroller which contains some amount of flash memory for program storage and RAM for program execution. Power is provided by batteries. Typical choices for the microcontroller are the 8 bit Atmel ATmega 128 or the 16 bit Texas Instruments MSP430 family, with the amount of RAM varying between 2 kB and 10kB and flash memory ranging from 48kB to 128 kB. The speed of radio communications is of the order of 100 kbit/s [5].

A number of applications of sensor networks has been suggested in diverse areas, including border area surveillance, environmental monitoring, health care and crisis management systems [3], [4], [6]. In some application areas, security is a major concern. When sensor networks carry sensitive information, it is important to ensure privacy. For example, in a surveillance application [7], it would be very undesirable if intruders can access the information being carried by the network. To provide security, the well-developed public key cryptographic methods have been considered, but these generally demand excessive computation and storage from the resource-poor sensors [8]. This has led researchers to conclude that symmetric key cryptography, in which nodes share a secret key, is the only viable solution.

While cryptographically strong algorithms are available, the issue of key distribution and management is critical to the level of security actually achieved. At one end of the spectrum, we have a system in which all the sensors share a single secret key. But this makes the network very vulnerable; an adversary needs to capture just a single sensor node to access any information that the network carries. At the opposite end, we have a system where each node has a distinct shared key for every other node. But for large sensor networks, such a scheme demands an excessive amount of on-board memory, which is again undesirable [5]. It is also possible for nodes to securely generate keys on the fly using key exchange algorithms, such as the well-known Diffie-Hellman scheme. However, the computational and storage requirements for such schemes have also been deemed unacceptable for sensor networks [8].

In [9], Eschenauer and Gligor suggested a probabilistic solution to the problem of efficient key distribution. In this scheme, each sensor node is assigned a key-ring consisting of \( k \) keys chosen at random (without replacement) from a pool of \( P \) keys. If the key-rings of two nodes have one or more keys in common, then one of these common keys can be used as the shared symmetric key and the link between these nodes is said to be secure. A problem arises when two nodes wanting to communicate do not share a common key. In this case, [9] proposes that nodes execute a path key establishment procedure, in which the source node \( S \) transfers a secret key to the destination node \( D \) via a path made up of secure links. A drawback of this scheme is that the secret key is known to all the nodes on the path from the source to the destination node. If any of these nodes is compromised, then the communication between the source and destination becomes insecure.

This paper is concerned with the problem of securely establishing a secret key between the source and the destination. Several researchers have considered this problem. In [1], the authors proposed an elegant solution of using multiple node-disjoint paths between \( S \) and \( D \) for secure path key establishment. But the problem of discovering multiple node-disjoint paths is computationally hard, and too much overhead may be incurred in this process. In a later work [2], the authors relax the requirement of node-disjoint paths, and utilize multiple proxies for path key establishment. A proxy \( P \)
is a node that shares one or more keys with the source node $S$ and one or more keys with the destination node $D$.

In this paper, we propose a novel scheme based on nodes that are referred to as friends of the destination. A friend of the destination is simply a node that shares one or more keys with the destination. Each friend $F$ in a neighborhood of $S$ sends part-keys back to the source, where a part-key is obtained by applying a hash function to all the keys shared between $F$ and $D$. The source then chooses a number of these part-keys, say $i$, and uses a publicly known function to generate the shared key $K_{SD}$ from them. $S$ informs $D$ about which $i$ friends’ part-keys were used, and this information is sufficient for $D$ to generate $K_{SD}$ using the publicly known function.

We compare our friend-based scheme with the proxy-based scheme reported in [2], and find several advantages. First, for a source-destination pair, the requirement for a node to be a friend is less stringent than for it to be a proxy. This implies that the computation and communication effort in finding a friend is less than in finding a proxy, making the friend-based scheme reported in [2], and find several advantages. First, for a source-destination pair, the requirement for a node to be a friend is less stringent than for it to be a proxy. This implies that the computation and communication effort in finding a friend is less than in finding a proxy, making the friend-based approach more viable. Second, our friend-based scheme is able to achieve a level of security at least as good as the one based on proxies. Last but not the least, for a given security, the scheme consumes less energy in establishing pairwise key as compared to proxy-based scheme.

II. RELATED WORK

The random key pre-distribution scheme was first proposed by [9]. We discuss this proposal in some detail in the next section. Based on this, several schemes with enhanced security features have been suggested. A $q$-composite-random key pre-distribution scheme is proposed by [10] which achieves strengthened security under small-scale attack while trading off increased vulnerability in the face of a large-scale physical attack on network nodes. It then uses multi-path key reinforcement scheme to update the communication key to a random value after key set-up phase.

In [11], a seed-based approach is used for assigning keys to each node. Each key is associated with a unique key identifier or key-id. Moreover, the key itself cannot be deduced from the knowledge of the key-id. For any node, a set of key-ids is generated from a common pseudo-random generator with the node identity acting as the seed. The corresponding keys are then stored in the node. This makes it possible for each node to identify the key-ids that another node has, and thereby find if they share any common keys. The seed-based approach reduces the communication burden in sharing key identifiers. For example, the source node can find out if it shares a common key with the destination node without having to communicate with the destination node. Ref. [12] uses a similar technique for shared key discovery phase.

Ref. [13] gives a scheme where memory requirements can be reduced at the expense of pre-deployment knowledge. In this scheme, knowledge about which nodes are likely to be the neighbors of each sensor node is exploited such that the probability of any two neighboring nodes sharing a common key is maximized without degrading the other performance metrics, such as security and memory usage.

The scheme in [14] exhibits a nice threshold property: When the number of compromised nodes is less than the threshold, the probability that any node other than these compromised nodes are affected is close to zero. Their scheme builds on Blom’s key pre-distribution scheme [15] and combines the random key pre-distribution method with it.

Our work is closest to that reported in [1] and [2]. Both these papers consider the path key establishment problem. Let $S$ and $D$ be the source and destination nodes between which a shared key is required. $D$ generates a secret key and this key is securely passed to $S$.

Ref. [1] proposes a scheme in which the key is broken up into $l$ nuggets, and the nuggets are passed to $D$ along node-disjoint paths. All nuggets are required to reconstruct the key. Therefore, an attacker has to capture at least one node along each of the node-disjoint paths to recover the key. In [2], the authors note the following shortcomings of this scheme: (a) Finding $l$ node-disjoint paths is a NP-hard problem, and too much overhead is required; further, in some cases it may not be possible to find $l$ such paths. (b) A nugget is exposed to each intermediate node along its path to $S$; thus, contrary to intuition, increasing $l$ does not necessarily improve the level of security, because the nuggets are exposed to more nodes, increasing the vulnerability of the scheme.

To address this drawback, [2] proposes a scheme in which no more than one node along a path knows the key nugget. This node is referred to as a proxy. A proxy shares one or more keys with $S$ and also one or more keys with $D$. Thus, $D$ can securely pass a nugget to the proxy on the path, and the proxy can securely relay the nugget to $S$. Moreover, the paths used need not be node-disjoint any longer, and, in fact, need not be composed of secure links either, because security is achieved by the shared keys between $S$ and the proxy, and the proxy and $D$.

III. PATH KEY ESTABLISHMENT

We begin this section by reviewing briefly the basic scheme in [9]. We also recall the method of efficiently generating the key-ids belonging to a node given in [11]. We then discuss the path key establishment scheme given in [2]. This is followed by our friend-based scheme.

In [9], Eschenauer and Gligor suggested a probabilistic solution to the problem of efficient key distribution. In this scheme, each sensor node is assigned a key-ring consisting of $k$ keys chosen at random (without replacement) from a pool of $P$ keys. After deployment, two nodes within communication range exchange key-identifiers or challenges to discover common keys. Then, a common key is selected for secure communication. Node pairs without a common key establish a path key through a secure path.

Fig. 1 shows a WSN with 7 nodes. If two nodes are within radio range, they are joined by a solid line representing a link. The collection of nodes and links constitutes the “network graph”. Further, a dashed line connecting two nodes indicates
that they share one or more keys, i.e., the link between them is secure. The sub-graph consisting of the nodes and the secure links is referred to as the “key graph”.

A natural question that arises in this context is whether the key graph is connected. Ref. [9] makes use of fundamental results in the theory of random graphs, due to Erdős and Rényi [16], to determine the pool size $P$ and the key-ring size $k$ such that the key graph is connected with arbitrarily high probability.

Thus, two neighbours on the network graph can find, with very high probability, a secure path between them. This secure path is constituted by a sequence of links from the key graph. Ref. [9] suggests that this secure path be used to establish a key between the neighbours. In this approach, the key to be shared is successively encrypted and decrypted by the nodes along the secure path. The drawback is that the key to be shared is exposed to each node on the path, and if one of these nodes is compromised, the key is available to the adversary.

The seed-based approach of [11] reduces the communication burden in sharing key identifiers, required during shared key set-up phase. For example, the source node can find out if it shares a common key with the destination node without having to communicate with the destination node, by simply using a pseudo-random generator with node-ids as seed.

A. Proxy-based Scheme

The solution to secure path key establishment given in [2] utilizes the notion of proxies. The idea is that $D$ breaks the secret key $K_{SD}$ into key nuggets. Each nugget is then send securely to a proxy by using one of their common keys. Next, each proxy $P$, then forwards the key nugget securely to $S$ using one of their common keys. Any path between $D$ and $P$, and between $P$ and $S$ can be used for this. The authors give two algorithms that $D$ can use to discover proxies.

B. Friend-based Scheme

The friend-based scheme we propose is similar in spirit to the scheme in [2], but has some further advantages. We will utilise the seed-based approach [11] for the shared key discovery phase, which reduces the communication burden of sharing key identifiers. We now outline the friend-based scheme.

$S$ broadcasts a request packet containing the identifier of $D$ and with the Time-To-Live (TTL) field set to $H$ hops. The request is sent to the nodes by using the broadcast support in the underlying routing protocol. Intermediate nodes receiving the request packet check their key-rings to see if they have any keys in common with $D$. If a node does not share a key with $D$, and if TTL is not zero, it simply forwards the request packet to others and decrements the TTL value by one. If it does share a key, then it is a friend.

A friend responds to a received request packet by sending a part-key back to $S$. A part-key is simply an $l$-bit substring of a key shared between $F$ and $D$. It is obtained by using a pre-specified hash function $h(\cdot)$ on the key. In case $F$ shares multiple keys with $D$, $F$ performs an XOR operation on all the common keys and finally applies the function $h(\cdot)$ to the result. This part-key is sent back to $S$.

A friend does not forwards the request packet after responding to it. We note that some of the friends who respond to a request may actually be proxies, because they share keys with both the destination $D$ and the source $S$. All those friends which are proxies, send part-key back to $S$ in encrypted form using the underlying encryption algorithm $E_{K}(\cdot)$, with a bit in the header (hereby will be called as HEB, the Header Encryption Bit) set to 1. Other friends send part-key in clear text with HEB set to 0. Shortest path routing algorithm is used to route packets from $F$ to $S$.

On receiving part-keys from possibly multiple friends, source $S$ randomly selects $i$ of them. At least one of the selected part-keys should be in encrypted form. These $i$ part-keys are then combined using a publicly known function $g(\cdot)$ to obtain a full-length key $K_{SD}$ to be used between $S$ and $D$. Note that all those part-keys which were in encrypted form (say, $n_e$ in number) are first decrypted using the underlying decryption algorithm $E_{K}^{-1}(\cdot)$. They are then used, along with other part-keys, as arguments for the function $g(\cdot)$.

After $S$ selects $i$ friends at random, the last step of the algorithm consists of securely conveying the identities of the selected friends to $D$. Once $D$ becomes aware of the identities of the $i$ selected friends, $D$ can find out all the keys that it shares with a particular friend - this occurs because knowledge of the node-id of the friend enables $D$ to find out the key-ids that the friend possesses. Hence, $D$ can compute the part-keys using the function $h(\cdot)$, exactly those part-keys that the friends had transmitted to $S$.

We note that only the identities of the friends are being sent to the destination $D$, not the part-keys. Moreover, these identities are sent securely to the destination by choosing one of the proxies, the information about which was already obtained during the broadcast-reply phase. Even if an adversary somehow manages to extract this information, he will not be able to generate the secret key. He will merely know the key-ids of the friends involved and thus the common keys shared between every friend and $D$. Since it is computationally impossible to obtain keys from key-ids, the adversary will not have access to the keys themselves. Now, in addition to having node-ids’ information, if the adversary also captures
the communication between all the friends and $S$, he will not be able to generate the key $K_{SD}$ as some of the part-keys involved were received in encrypted form. We note also that the method outlined here is independent of the specific functions $h(.)$ and $g(.)$ that are used. Pseudo-code for the friend-based scheme is supplied in the Appendix.

We emphasize here that our scheme depends on the presence of at least one proxy. If none of the part keys are received in encrypted form, then the source rejects all of them and starts a fresh round of discovery by sending the request packet again. We will show later that the probability of finding at least one proxy is very close to 1.

IV. COMPARATIVE ANALYSIS

In this section we first establish the viability of our friend-based approach as compared to the proxy based approach. We show that in any collection of $m$ nodes, the number of friends is statistically higher than the number of proxies. We then evaluate the security of our scheme in terms of the resilience of our scheme to node capture. We also show that the friend-based scheme requires adversaries to expend much more computational effort than the proxy-based scheme. Finally, we show that, for a given level of security, the friend-based scheme consumes less energy as compared to proxy-based scheme, making our scheme more viable.

A. Feasibility analysis

Suppose that there are $m$ nodes in the network excluding the source and destination nodes. Let $v$ be a node other than the source and destination nodes. Let $A$ be the event that $v$ shares at least one key with the source node $S$, $B$ be the event that $v$ shares at least one key with the destination node $D$ and $C$ be the event that the source and destination nodes do not share a key. ¹We denote the complement of an event $E$ by $E^c$.

Let $p'$ be the probability that a node other than the source and destination nodes is not a friend, given that the source and destination do not share a key. If $P$ is the key pool size and $k$ is the ring size, then the probability $p$ of a node being a friend is given by

$$p = 1 - p' = 1 - P(B^c|C) = 1 - \left(\frac{P-k}{k}\right)$$

Let $q$ be the probability that a node other than the source and destination nodes is a proxy, given that the source and destination do not share a key. We have

$$q = P(A \cap B|C) = 1 - P(A^c \cup B^c|C) = 1 - P(A^c|C) - P(B^c|C) + P(A^c \cap B^c|C) = 1 - 2\left(\frac{P-k}{k}\right) + \frac{P-2k}{k}$$

¹This is the event when the source and destination need to establish a secret key.

By comparing (1) with (2), we see that $p > q$. Therefore, in any collection of nodes, it is easier to find friends compared to proxies. The complementary cumulative distribution function of the number of friends and proxies in a collection of $m$ nodes is plotted in Fig. 2.

![Fig. 2. The complimentary distribution function of the number of friends and proxies.](image)

Since the proposed scheme depends on the presence of at least one proxy, it is worthwhile to look at the probability of finding at least one proxy. This probability is given by

$$Pr(\text{at least one proxy}) = 1 - (1 - q)^N$$

where $N$ is the total number of nodes till $H$ hops. If $r$ is the radio range of each node, $n$ is the total number of nodes in the network and $A = \pi R^2$ is the total area then $N$ is given as

$$N = \sum_{h=1}^{H} [h^2 - (h - 1)^2] \frac{\pi r^2 n}{A} = \sum_{h=1}^{H} [h(2h - 1)] \frac{\pi r^2 n}{A}$$

It can be seen that for some given values of $r$, $q$, $n$ and $R$ (say, $r = 50$ m, $q = 0.216$, $n = 10000$, and $R = 790.57$ m), the above probability will be close to 1, even for low values of $H$.

B. Security Analysis

In analysing the security properties of the friend-based and proxy-based schemes, we assume that $S$ and $D$ are never captured. In our threat model, we assume that once a node is captured, all of its keys become available to the adversary. Consequently, if $S$ or $D$ is captured then the secret key is revealed straight away. The more interesting case is when neither is captured.

Suppose that apart from $S$ and $D$, there are $m$ nodes in all. Let us now assume that $x$ nodes (distinct from $S$ and $D$) have been captured. It is possible for the adversary to find
out of which these \( x \) nodes are friends or proxies. For this, knowing the node-id of \( D \) is sufficient. Then, the adversary can generate the key-ids in \( D \)'s key-ring, as well as those in a captured node’s key-ring and find out if there is a non-empty intersection. In this way, the number of friends or proxies in the \( x \) captured nodes can be obtained.

We now consider the \( i \) friends or proxies that were actually used to generate the key \( K_{SD} \). Clearly, the \( x \) captured nodes can either include these \( i \) friends/proxies (let us call this event \( T \)) or not. If \( T \) occurs, then the adversary cannot obtain the secret key. If \( T \) occurs, the adversary has to recover the key after some more effort, as we show below.

If \( T \) occurs, we know that the \( x \) captured nodes contain at least \( i \) friends/proxies, because the \( i \) friends/proxies actually used for \( K_{SD} \) are already present. Let the total number of friends/proxies present among the captured nodes be the random variable \( Z \), taking values in \( \{i, (i+1), \ldots, x\} \). In this situation, the adversary has to guess the right collection of \( i \) friends to obtain \( K_{SD} \). Let us suppose that the adversary can make \( L \) attempts to get the key. Then, our basic measure of resiliency against node capture is the probability of key recovery by the adversary in \( L \) attempts.

By an attempt, we mean a guess where an adversary picks a particular set of \( i \) nodes from the \( x \) captured nodes and assumes that these \( i \) nodes were used to obtain the secret key. If the guess is wrong, then the adversary has spent time and computation effort, but has not been able to recover the key \( K_{SD} \). Thus, the cost corresponding to a wrong guess is wasted time and computation. Clearly, as \( L \) increases, we expect the probability of key recovery in \( L \) attempts to increase.

Let \( H_L \) denote the event that \( K_{SD} \) is recovered by the adversary in \( L \) attempts. Then

\[
P(H_L) = P(T^c)P(H_L|T^c) + P(T)P(H_L|T)
\]

Now, as noted before, \( P(H_L|T^c) = 0 \). So we need to obtain \( P(T) \) and \( P(H_L|T) \). \( P(T) \) is given by

\[
P(T) = \sum_{x=1}^{x} \binom{x}{i}
\]

Also

\[
P(H_L|T) = \sum_{k=1}^{x} P(H_L, Z = k|T)
\]

\[
P(H_L|Z = k, T) \text{ can be found as follows. Given that there are } k \text{ friends/proxies, the probability that the adversary finds the "correct" combination of friends/proxies in the first attempt is } \frac{1}{\binom{x}{i}}.
\]

Similarly, using a standard combinatorial argument, the conditional probability that the correct combination of friends is found in the 2\textsuperscript{nd} attempt is also \( \frac{1}{\binom{x}{i}} \). In fact, this argument applies equally well to cases where the correct combination is found in the 3\textsuperscript{rd}, 4\textsuperscript{th}, \ldots, \min(L, \binom{x}{i})\textsuperscript{th} attempt. In all cases, the conditional probability is \( \frac{1}{\binom{x}{i}} \). Thus, we have

\[
P(H_L|Z = k, T) = \min\left(\frac{L}{\binom{x}{i}}, \binom{x}{i}\right)_{\binom{x}{i}}
\]

The difference between the friend-based and proxy-based schemes appears when computing \( P(Z = k|T) \). Recalling that \( p \) denotes the probability that a node is a friend and \( q \) the probability that a node is a proxy, we have from the binomial distribution:

\[
P(Z = k|T) = \binom{x}{i} p^{k-i} (1-p)^{x-k}
\]

for the friend-based scheme and

\[
P(Z = k|T) = \binom{x}{i} q^{k-i} (1-q)^{x-k}
\]

for the proxy-based scheme. Hence, from (3), (4), (5), (6), (7) and (8) we have

\textbf{Theorem 4.1:} When \( x \) nodes are captured by the adversary, the probability of key recovery in \( L \) attempts in the friend-based scheme is

\[
\frac{\binom{m-i}{x-i} \sum_{k=1}^{x} \min(L, \binom{k}{i}) \binom{x-i}{k-i} p^{k-i} (1-p)^{x-k}}{(x-i)!(x-k)!}
\]

and in the proxy-based scheme is

\[
\frac{\binom{m-i}{x-i} \sum_{k=1}^{x} \min(L, \binom{k}{i}) \binom{x-i}{k-i} q^{k-i} (1-q)^{x-k}}{(x-i)!(x-k)!}
\]

In Fig. 3 we show how the probability of key recovery varies with \( x \) for the two schemes, for some values of \( m, i, p \) and \( q \). It can be seen that the probability of key recovery is distinctly lower for the friend-based scheme.

We also note that as \( L \) tends to infinity, we obtain the probability that key recovery occurs at all. When \( L \to \infty \), the expressions suggest that the probability of key recovery in either scheme approaches \( \binom{m-i}{x-i} \). This is intuitively meaningful, because (i) when many attempts are allowed, the difference


Fig. 4. Probability of key capture with increasing $L$.

**TABLE I**

| $x$ | $P(T)$ | $E[N_F|T]$ | $E[N_P|T]$ |
|-----|--------|------------|------------|
| 29  | 0.1    | 256        | 46         |
| 35  | 0.2    | 355        | 72         |
| 41  | 0.3    | 682        | 106        |
| 45  | 0.4    | 891        | 133        |
| 48  | 0.5    | 1071       | 157        |
| 51  | 0.6    | 1274       | 184        |
| 53  | 0.7    | 1425       | 203        |
| 56  | 0.8    | 1669       | 234        |
| 58  | 0.9    | 1846       | 256        |
| 60  | 1.0    | 2037       | 280        |

between the two schemes is due only to the probabilities of finding a friend and a proxy ($p$ and $q$), and (ii) when a large number of nodes is captured, practically the entire mass of the binomial distribution (in the expressions above) are being considered and the summation is very close to 1. Hence, the difference in performance between the two schemes vanishes as $L$ becomes large, with the probability of key recovery being given by $\binom{m}{i} x^i (1-x)^{m-i}$. This can be seen in Fig. 4.

The analysis above was carried out for a chosen number of attempts $L$. Alternatively, one may ask the question: Given that the $i$ friends/proxies used to generate $K_{SD}$ are included in the $x$ captured nodes (i.e., the event $T$), what is the average number of attempts required by an adversary to obtain $K_{SD}$? Letting $N_F$ and $N_P$ denote the random number of attempts required in the friend-based and proxy-based schemes respectively, we seek to obtain $E[N_F|T]$ and $E[N_P|T]$. This requires that the conditional probabilities $P(N_F = l|T)$ and $P(N_P = l|T)$, $l = 1, 2, \ldots$, be obtained first; this can be done using the same approach employed above. We omit the details for brevity. Some representative values are given in Table I. It can be seen that as $x$ increases, the difference in computation effort increases dramatically. To get some intuitive understanding of this phenomenon, we consider the friend-based scheme with large $x$ as is done in [14], [17], [18]. In this scenario, the number of captured friends will be very close to the expected number, namely $xp$, with high probability. Similarly, for the proxy-based scheme, the number of expected proxies is very likely to be around $xq$. Thus, we can make a first-cut comparison between the two schemes by considering the expected value of $N_F$ and $N_P$, given that $xp$ and $xq$ nodes have been captured, respectively.

We plot these values in Fig. 5 and note that $N_F$ is almost an order of magnitude greater than $N_P$.

**C. Energy Analysis**

Energy is one of the most important constraints in sensor networks and hence any algorithm which can provide enhanced security with minimal usage of energy should be considered as a viable option for securing sensor networks. Therefore, in this section, we compare our scheme with the proxy-based scheme in terms of energy and show that for a given security measure, as discussed in the previous section, the energy consumption for pairwise key set-up is lower in the friend-based scheme as compared to the proxy-based scheme.

Throughout our discussion, we assume that the nodes are distributed uniformly over an area $A_R$ and that each node transmits at constant energy (per bit) to its neighbours irrespective of their distance from the node. For the sake of comparison, let us assume that the part-keys and the key-nuggets are of the same length $L_{pk}$. Also, since the energy spent during broadcast will be same in both the cases, we will omit it from the energy equations. Energy for encryption/decryption is being neglected for the simple reason that it is about three orders ($10^3$) of magnitude less than the energy for transmission [8].

In the friend-based scheme, all the friends within $H$ hops ($TTL = H$) reply back to the source $S$ with the part-keys.
$S$ randomly picks $i$ part-keys and sends the corresponding nodes id information (of length $L_{id} = \lceil \log(n) \rceil$) through a secure path to the destination which is $d$ hops away from the source. If $E_{F-S}$ is the average energy spent in getting all the friends’ information at the source and $E_{S-D}$ is the average energy spent in sending node-id information to the destination, then the total average energy spent in establishing a pair-wise key is given by

$$E_F = E_{F-S} + E_{S-D} \quad (9)$$

If $e_t$ is the transmission energy per bit, $e_r$ is the energy spent per bit in receiving the data, and $L_{hdr}$ is the packet header length, then $E_{F-S}$ is simply the average over each of the annular ring between $h$ and $h-1$. We, therefore, have

$$E_{F-S} = \sum_{h=1}^{H} \sum_{k=0}^{N_h} h(h^2 - (h-1)^2) \pi r^2 n_k p^k (1-p)^{N_h-k} \frac{H^2}{A} \times (e_t + e_r)(L_{hdr} + L_{pk})$$

where $N_h$ is the number of nodes at the $h^{th}$ hop annular ring and $p$ is the probability of finding a friend. Simplifying the above expression, we have

$$E_{F-S} = \sum_{h=1}^{H} [h(2h-1)] \pi r^2 n_k p^k (e_t + e_r)(L_{hdr} + L_{pk})$$

Now, the $i$ randomly chosen part-keys’ node-id information which is send to the destination, will consume energy given by

$$E_{S-D} = d(e_t + e_r)(L_{hdr} + iL_{id})$$

Thus, $E_F$ will be

$$E_F = \sum_{h=1}^{H} [h(2h-1)] \pi r^2 n_k p^k (e_t + e_r)(L_{hdr} + L_{pk}) + d(e_t + e_r)(L_{hdr} + iL_{id})$$

\[ (10) \]

In the proxy-based scheme, proxies within $H$ hops from the destination $D$ reply back to $D$. Destination randomly picks $i$ of the proxies and sends the key-nuggets to the source via these proxies. If $E_{P-D}$ is the energy spent in getting all the proxies’ information at the destination and $E_{D-S}$ is the energy spent for sending key-nuggets to the source, we have

$$E_P = E_{P-D} + E_{D-S} \quad (11)$$

where

$$E_{D-S} = E_{D-P} + E_{P-S}$$

Again, $E_{P-D}$ is the average over each of the annular ring between $h$ and $h-1$ (Fig. 6). Therefore,

$$E_{P-D} = \sum_{h=1}^{H} [h(2h-1)] \pi r^2 n_k p^k \sum_{h'=1}^{H} (e_t + e_r)(L_{hdr})$$

where $q$ is the probability of finding a proxy. We have assumed here that the node id information of the proxy can be obtain from the packet header itself.

Finding $E_{D-P}$ and $E_{P-S}$ is analytically bit cumbersome and so we limit ourselves to the lower and upper bounds. The approximate lower bound occurs when the distance between the proxy and $D$ is taken as $d-h$ and upper bound when the distance is $d+h$.

Let, out of $i$ randomly chosen proxies, $i_1$ belong to $1^{st}$ hop annular ring, $i_2$ belong to $2^{nd}$ hop annular ring, ..., $i_H$ belong to $H^{th}$ hop annular ring such that

$$i = i_1 + i_2 + \ldots + i_H$$

Now, the probability of choosing $\alpha$ proxies from $1^{st}$ hop annular ring, $\beta$ proxies from $2^{nd}$ hop annular ring, ..., $\omega$ proxies from $H^{th}$ hop annular ring will be

$$Pr \{i_1 = \alpha, i_2 = \beta, \ldots, i_H = \omega\} = \frac{i!}{\alpha!\beta!\ldots\omega!} \prod_{h=1}^{H} \left( \frac{(2h-1)}{H^2} \right)^{i_h} \quad (12)$$

where $p_{ih}$ is the probability of choosing a proxy from $h^{th}$ hop annular ring. Assuming this probability to be constant for successive independent draws, we have

$$Pr \{i_1 = \alpha, \ldots, i_H = \omega\} = \frac{i!}{\alpha!\ldots\omega!} \prod_{h=1}^{H} \left( \frac{(2h-1)}{H^2} \right)^{i_h} \quad (13)$$

Thus, $E_{D-P}$ for both the upper and lower bounds is given by

$$E_{D-P} = \sum_{i_{h-1}=0}^{i-1} \sum_{i_h=0}^{i} \frac{i!}{\alpha!\ldots\omega!} \prod_{h=1}^{H} \left( \frac{(2h-1)}{H^2} \right)^{i_h} \times \sum_{h'=1}^{H} (i_h h' \times (e_t + e_r)(L_{hdr} + L_{pk})$$

\[ (13) \]
Also, the upper and lower bounds for $E_{P\rightarrow S}$ will be

$$E_{P\rightarrow S(UB)} = \sum_{i_1=0}^{i} \ldots \sum_{i_{H'-1}=0}^{i} \frac{i!}{i_1! \ldots i_{H'}!} \prod_{h=1}^{H} \left( \frac{(2h-1)}{H^2} \right)^{i_h} \times \sum_{h'=1}^{H} i_{h'}(d + h')(e_t + e_r)(L_{hdr} + L_{pk})$$

$$E_{P\rightarrow S(LB)} = \sum_{i_1=0}^{i} \ldots \sum_{i_{H'-1}=0}^{i} \frac{i!}{i_1! \ldots i_{H'}!} \prod_{h=1}^{H} \left( \frac{(2h-1)}{H^2} \right)^{i_h} \times \sum_{h'=1}^{H} i_{h'}(d - h')(e_t + e_r)(L_{hdr} + L_{pk})$$

Therefore, the upper and lower bounds of $E_{D\rightarrow S}$ for the proxy case are given as

$$E_{D\rightarrow S(UB)} = \sum_{i_1=0}^{i} \ldots \sum_{i_{H'-1}=0}^{i} \frac{i!}{i_1! \ldots i_{H'}!} \prod_{h=1}^{H} \left( \frac{(2h-1)}{H^2} \right)^{i_h} \times \sum_{h'=1}^{H} i_{h'}(d + h')(e_t + e_r)(L_{hdr} + L_{pk})$$

$$E_{D\rightarrow S(LB)} = i_d(e_t + e_r)(L_{hdr} + L_{pk})$$

Thus, the total average energy consumed in proxy-based is given by

$$E_{P(LB)} = \sum_{h=1}^{H} [h(2h-1)] \frac{\pi r_q^2}{A} (e_t + e_r)(L_{hdr} + L_{pk})$$

$$E_{P(UB)} = \sum_{h=1}^{H} [h(2h-1)] \frac{\pi r_q^2}{A} (e_t + e_r)(L_{hdr} + L_{pk})$$

Fig. 7 shows a plot of the energy required to set up a pairwise key and the probability of obtaining the key in a given number of attempts. As the TTL value increases, the number of nodes involved increases drastically. Thus, for higher TTL values, the probabilities are very low and energy consumptions are high. To accommodate comparison over a large range of TTL values, we have chosen an logarithmic scale for both axes. Typical values for $e_t$, $e_r$, $L_{hdr}$, and $L_{pk}$ are 0.021 mJ/bit, 0.014 mJ/bit, 320 bits, and 96 bits respectively.

In Fig. 7, each marked point corresponds to a particular value of TTL (the TTL value is indicated in the plot). It can be seen that to achieve a given level of security (which corresponds to a given point on the x-axis), the required TTL value is greater for the proxy-based scheme. This already indicates two things: (a) the energy consumed in the proxy-based scheme will be greater, and (b) the time delay involved in key set-up for the proxy-based scheme will also be greater. Another way to interpret the graph is by fixing the maximum possible energy consumption acceptable in key set-up (that is, this corresponds to a chosen point on the y-axis). For a given energy budget, we see that the security provided by our scheme is better than that provided by proxy based scheme making our scheme more viable for sensor networks.

V. CONCLUSION

Our friend-based scheme for secure path key establishment is inspired by the proxy-based scheme in [1] and [2]. We showed that because a friend needs to share one or more keys with only the destination, the chances of finding a friend in a neighbourhood are considerably greater than that of finding a proxy. This means that friends can be found with less communication overhead, and therefore, appreciable savings in energy can result.

Further, we showed that the friend-based scheme exhibits clear advantages with respect to resilience against node capture. This was proved by obtaining analytical expressions for the conditional probability of key recovery within $L$ attempts by the adversary. For typical scenarios, the average computational effort for key recovery was also shown to be much larger for the friend-based scheme. Finally, the energy
consumption for pair-wise key set-up was shown to be less in case of friend-based scheme as compared to proxy-based scheme.

ACKNOWLEDGEMENTS

This work was supported by the Defence Research and Development Organization (DRDO), Ministry of Defence, Government of India, under a research grant on wireless sensor networks (DRDO 571, IISc).

REFERENCES


APPENDIX

Algorithm 1 The friend-based algorithm

Pseudo-code at $S$:

Input: Destination id $D$
Output: A pairwise key $K_{SD}$

Set TTL = $H$
Request packet (RP) contents: $D$
Receive part-keys
Randomly select: $n_e$ part-keys with $HEB = 1$ & $i - n_e$ with $HEB = 0$
for $j = 1$ to $n_e$ do
    Part-key $j = E_{K_{SD}}^{-1}$ (Selected part-key $j$)
end for
for $j = n_e + 1$ to $i$ do
    Part-key $j = $ Selected part-key $j$
end for
$K_{SD} = g$ (part-key $1$, part-key $2$, ..., part-key $i$)
Send $E_k$ (node-id $1$, node-id $2$, ..., node-id $i$) $\rightarrow D$

Pseudo-code at $F_j$ ($j^{th}$ friend of $S$):

Input: Destination id $D$
Output: Part key $F$

Given $D$, generate key-ids $K_{D_1}, K_{D_2}, \ldots, K_{D_k}$
Given $S$, generate key-ids $K_{S_1}, K_{S_2}, \ldots, K_{S_k}$
Part-key $F = 0$; flag$_{sa} = 0$; flag$_{sd} = 0$; $HEB = 0$
for $m = 1$ to $k$ do
    for $l = 1$ to $k$ do
        if ($K_{F_m} == K_{D_l}$) then
            Part-key $F = E_{K_{D_l}}$ (Part-key $F$)
            flag$_{sd} = 1$
        end if
        if ($K_{F_m} == K_{S_l}$) then
            flag$_{sa} = 1$
        end if
    end for
    if (flag$_{sa} = 1$) and (flag$_{sd} = 1$) then {Proxy node}
        Part-key $F = E_{K_{SD}}$ (Part-key $F$)
        $HEB = 1$
    end if
    if (Part-key $F \neq null$) then
        Send Part-key $F$ and identity $F$ to $S$
    else if (TTL $\neq 0$) then
        Broadcast request packet to neighbours
    else
        drop the packet
end if