

# Path Key Establishment using Multiple Secured Paths in Wireless Sensor Networks \*

Guanfeng Li, Hui Ling and Taieb Znati  
Department of Computer Science  
University of Pittsburgh  
Pittsburgh, PA 15260, U.S.A  
Email: {ligf, hling, znati}@cs.pitt.edu

## Abstract

*Random key Pre-distribution scheme has been proposed to overcome the memory, computation and energy limits of individual sensor in wireless sensor networks. In this scheme, a ring of keys is randomly drawn from a large key pool and assigned to a sensor. Nodes sharing common keys can communicate securely, path-key needs to be established for those nodes who do not share any common keys. However, there is no hard security guarantee for path key set up in existing protocols. In this paper, we propose two new methods to discover multiple secure proxies and a scheme to secure path key establishment using secure proxies discovered. We show both from analysis and simulation that our scheme can achieve a high level of security against node capture based on the network parameters the user has chosen, meanwhile only incurring a small amount of overhead. All path keys established in our scheme is exclusively known by two end nodes with high probability.*

## 1. Introduction

Recent advances in wireless technologies have led to a new generation of inexpensive sensors and actuators. Individually, these devices are resource-constrained and, as such, are only capable of a limited amount of processing and communication. When deployed in a large number, however, the coordinated effort of these networked devices

\*This work is supported by NSF awards ANI-0325353, ANI-0073972, NSF-0524634 and NSF-0524429.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CoNEXT'05, October 24–27, 2005, Toulouse, France.  
Copyright 2005 ACM 1-59593-097-X/05/0010 ...\$5.00.

bears promises for a significant impact, not only on science and engineering, but equally importantly on a broad range of civil and military applications, including health care, critical infrastructure protection, environmental and wildlife monitoring, crisis management, and military reconnaissance.

Harnessing the potential of wireless sensor networks, however, brings about a number of fundamental challenges, the most critical of which is security. It is frequently the case that sensors are deeply embedded into the environment or deployed in open areas, making them vulnerable to physical attacks and potentially compromising sensor nodes' security. Secure communication among sensors, during the response phase to an attack on a critical infrastructure, for example, is crucial for emergency responders to successfully coordinate their activities. Malicious information, injected by attackers during the response phase may hamper greatly the ability of first responders to communicate and share data. Cryptology methods are, therefore, needed to achieve secure communication among sensor nodes.

Since sensors will either have to be powered by small non-renewable batteries, or by a modest amount of energy that can be harvested from the environment, developing energy-efficient cryptographic algorithms and methods is a critical issue in designing security protocols for wireless sensor networks. The sensors' resource constraints, coupled with their limited knowledge of the topology within which they are deployed, render Public Key Infrastructure (PKI) based schemes inappropriate for wireless sensor networks. Carman, Kruus and Matt pointed out that asymmetric cryptography algorithms, like 1024-bit RSA, consume at least two orders of magnitude more energy than symmetric cryptography algorithms, such as 1024-bit AES in[1]. Furthermore, symmetric-key cipher and hash functions execute between two to four orders of magnitude faster than their asymmetric counterparts. Similarly, trusted server based cryptography systems, such as Kerberos, do not apply in WSNs, as these schemes require a trusted third party which is not always available in WSNs. Consequently these schemes may not be scalable when a WSN involves thousand of sensor nodes. These constraints leave designers of security protocols for WSNs with no choice but to use symmetric-key cryptographic systems.

In symmetric-key cryptographic systems, keys have to be installed onto sensors before deployment. Nodes then use shared keys to conduct secure communication. Two strategies can be used to distribute shared keys between sensors in WSNs. In the first strategy, all sensor nodes share the same session key, while in the second case each sensor node shares a unique key with the remaining  $n - 1$  sensors, where  $n$  is the total number of sensors in the WSN. The advantage of the first strategy stems from its low maintenance cost. In this strategy, however, the compromise of one single node may jeopardize the security of the entire network. The second strategy has potential to achieve perfect security even when a number of nodes are captured. In large WSNs, however, this approach requires installing  $n - 1$  keys in each sensor and, as such, may be prohibitive, given the limited memory size of a sensor node. Furthermore, sensors are likely to fail due to hardware faults or energy depletion caused by excessive communication. Consequently, in order to maintain the level of node density required to meet the quality of service requirement of the applications, new sensors may have to be injected into the existing network. The addition of these nodes further limits the applicability of the second approach, as it requires installing new keys into the existing sensors in order to facilitate communication between these sensors and the newly injected ones.

To overcome the shortcomings of the above strategies, a random key pre-distribution scheme has been proposed [5]. This scheme only requires a relatively small number of keys, in the order of ten to hundred, to be installed onto each node, to achieve connectivity between pair of nodes with high probability. The link with two end nodes sharing keys is called secure link. Nodes, that do not share a key, set up a path key, through negotiation, using paths formed by secure links. The major shortcoming of this scheme is during path key establishment, communication between the end nodes is exposed to intermediate nodes along the path.

This path key establishment problem has been introduced in [8], furthermore it is shown that the risk of path key being revealed can be significantly decreased by using multiple node-disjoint secure paths to establish the path key. However, the proposed scheme may incur too much extra overhead due to the necessity of discovering multiple node-disjoint paths between sources and destinations.

In this paper, we propose to use multiple one-hop paths instead of node-disjoint paths to enhance the security of path-key establishment. We present two efficient algorithms for discovering these intermediate hops (referred as proxy). It is shown both through analysis and simulation that our scheme can achieve a very high level of security, while reducing the overhead.

The rest of this paper is organized as follows: section 2 introduces related works. In section 3, we present our key establishment scheme to secure the path key using multiple proxies. Furthermore, we show how to discover such proxies in two algorithms. The security analysis and simulation results show that our scheme can achieve a high level of security. Section 4 concludes this paper and presents future work.

## 2. Related Work

The Random Key Pre-distribution scheme was first introduced by Eschenauer and Gligor [5]. Using this framework, different methods of key generation and distribution have been proposed to improve energy efficiency and security [2][4][3][6]. A  $q$ -composite random key pre-distribution scheme has been proposed which increases the security of key set-up such that an attacker has to capture a large number of nodes to compromise a communication, with high probability [2]. The authors also propose a Multipath Key Reinforcement scheme to update an existing link key to a unique key, thereby ensuring that the key is not used by any other sensor node. Although the scheme proposed in this paper uses multiple paths, it differs from the one proposed in [2] in that the proposed scheme achieves the same level of security, without the need to use node-disjoint paths. Computing node disjoint paths is known to be NP-hard, and, therefore, may result in considerable communication overhead.

In the random key pre-distribution scheme proposed in [3], each node only needs to carry a fraction of the keys required by [5], while achieving the same level of security. This scheme has the potential to reduce memory usage and improve the network's resilience against node compromise. To achieve this goal, however, the scheme requires prior knowledge of sensor deployment within the WSN, which may not be readily available at any time. Furthermore, if the sensor nodes are moving, the network topology changes, thereby making prior knowledge deployment obsolete.

In [7] a seed-based key deployment strategy to discover shared keys, in a more energy-efficient manner, is described. All the keys in the key pool are indexed. Each node uses its ID as the seed and uses a pseudo-random function to generate the key indexes. It then loads the corresponding keys onto itself. This scheme requires more memory space as nodes have to store the associated indexes along with the keys. The scheme may also require additional computation, but no communication is required for two nodes to discover if they share a key between them.

The schemes described in [6] uses a similar technique to discover shared keys. Although these new schemes save communication, they pose a security threat. After capturing a node, an attacker can gain additional advantage by selectively eavesdropping on nodes that are known to share keys with the captured one.

To prevent this attack, the key distribution strategy proposed in this paper adopts the original scheme described in [5]. Notice that the scheme described in [7] also sets up path keys using different logic paths. However, the scheme cannot use the original shared key mechanism to discover these logic paths.

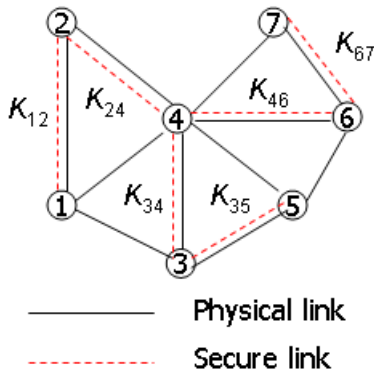
Both [4] and [2] propose a scheme to support key authentication by generating unique pairwise keys. In [2], a node loads a set of node IDs and a unique pairwise key  $k$  is generated for each pair of nodes. Hence, if  $k$  is used to secure communication, both nodes are certain of their respective identities, since no other node pair can hold  $k$ . In [4] the random key pre-distribution is combined with Blom's key pre-distribution scheme [2] to achieve " $\lambda$ -security". This level of security is achieved only if an adversary cannot compromise more than  $\lambda$  nodes; uncompromised nodes remain

perfectly secure. When more than  $\lambda$  nodes are captured, the entire network may be compromised.

While the security of random key pre-distribution scheme have made significant improvement, the path-key establishment problem has not yet been fully addressed [8]. In this paper, we propose to improve the path-key security using multiple secure proxies.

### 3. Path Key Establishment

We first review Gligor’s work upon which our work is based and give an example to highlight the main idea. In his scheme, each node is installed with a key ring of  $m$  keys randomly drawn from a large key pool,  $P$ . This scheme requires moderate memory space for storing a key ring, and therefore can be used in a very large network. For example, if a key ring consists of 20 keys and is drawn from 1000 keys, theoretically it can support up to  $\binom{1000}{20} = 2.4 \times 10^{19}$  nodes, and only requires 160 bytes assuming 64-bit key cryptography system. After being deployed, two nodes within transmission range exchange either key identifiers or challenges to discover common keys in their key rings. Then a common key is selected for secure communication between these two nodes. Node pairs without a common key establish a path key through a secure path.



**Figure 1. An example sensor network after shared key discovery.**

In the network depicted in Figure. 1, it is assumed that shared keys have been discovered as illustrated by dashed links. According to this example,  $N1$  shares a key with  $N2$  but not with  $N3$  or  $N4$ . When  $N1$  wants to communicate with  $N3$ , it finds a secure path  $N1 \rightarrow N2 \rightarrow N4 \rightarrow N3$  and sends a key  $K$  to  $N3$  through the established path.  $K$  is encrypted with  $K_{12}$ ,  $K_{24}$  and  $K_{34}$ , respectively as it travels from  $N1$  to  $N3$ . Notice, however, that while the pair-wise key  $K$  is supposed to be exclusively shared between  $N1$  and  $N3$ , the need for successive decryptions and encryptions along the path causes the key to be exposed to the intermediate nodes  $N2$  and  $N4$ . This may lead to potential security compromise if a node along the path is captured.

We refer to this problem as the “path-key establishment problem” [8].

Another security concern about this framework is that the probability of any two nodes sharing keys is high. In the last example, the probability is 33.5%. If the key ring size is increased to 30, the probability will be over 60.5%. Any key in the key pool has a probability equal to  $\frac{\text{Key ring size}}{\text{Key pool size}}$  to be installed on one node. In a large WSN, if two nodes are using a shared key to talk to each other, chances are, there will be some nodes in the neighborhood that hold this shared key they are using. This situation demands that two nodes set up a path key for private communication even they share keys on their key rings.

Using multiple node disjoint paths to secure the path key establishment has been proposed to cope with the risk that compromising one node along the path leads to revealing the path key [8]. A path key  $K$  is broken down into  $k$  nuggets and sent along  $k$  node-disjoint paths. All nuggets are required to reconstruct  $K$ . Therefore, an attacker would have to capture at least one node along each path to obtain the key. However, as pointed out above, these key nuggets are exposed to each intermediate node along the routing path. In summary, this scheme has the following undesirable features:

- It involves a high level of overhead to find nodes disjoint paths, furthermore, in some case, it may not be physically feasible to construct  $k$  node disjoint paths.
- Contrary to intuition, increasing the number of node disjoint paths does not necessarily improve the level of security of the underlying path key establishment scheme. This is because as the number of node disjoint paths increases, so does the number of intermediate nodes. This in turn increases the vulnerability of the path key.

To reduce the exposure of the key nugget along the path, the proposed scheme ensures that no more than one node along a path knows the key nugget. This node is referred to as a *proxy*. The proxy shares a key with each end node respectively. Now that the key nugget is secured by the proxy, it becomes feasible to relax the node disjoint requirement of the  $k$  paths without increasing the vulnerability of the path key. Furthermore, since these paths no longer require to be composed of secure links only, any physical path (e.g. the shortest path) between the proxy and the end nodes discovered by the underlying routing protocol can be used.

The fact that nodes share keys with high probability leads to the following two observations. On one hand, it imposes threat to reveal the key nuggets because of the exposure. On the other hand, it leaves a large number of nodes to act as proxies that can secure key nuggets exchanged between end nodes. Only the compromise of the proxy will cause the associated key nugget to be revealed. Consequently, the security level of establishing a path key will increase monotonically with the number of secured paths. Based on this observation, we propose Path Key Establishment Scheme which leverages multiple secure paths with only one proxy for key negotiation and establishment. We propose two simple algorithms to find these proxies and compare the response time and communication overhead in

terms of average number of hops and number of nodes involved to find a proxy. The following assumptions are made in our scheme and security analysis.

- Sensor nodes are not tamper resistant. Consequently, if a node is captured, the content in its memory is revealed to the attacker.
- An attacker can randomly compromise at most  $x$  out of  $n$  nodes.
- A routing structure has been established by a routing protocol.
- Attacker cannot get keys through traffic analysis.

The notations used throughout the paper are listed in Table 1.

**Table 1. Notation**

$P/R$	: A random key pre-distribution scheme with key pool size $P$ and key ring size $R$
$K$	: A path key to be established
$n$	: Total number of nodes in the network
$x$	: The maximum number of nodes an attacker can capture
$k$	: Number of secure paths to set up the path key
$p$	: The probability that two nodes share keys
$u v$	: Two nodes seeking privately communicate with each other

### 3.1 End-to-End Key Establishment Scheme

Consider a network with a total number of  $n$  nodes, where each node has been loaded with a key ring drawn from a large key pool. Furthermore, assume that node  $u$  wants to set up a path key with another node  $v$  to start a private communication. This can be achieved using the following steps:

- $u$  sends out its key ID list to invite  $v$  to set up a path key.
- $v$  randomly construct a key and breaks it down to  $k$  nuggets,  $K_1, K_2 \dots K_k$ , such as  $K = K_1 \cup K_2 \cup \dots \cup K_k$ , where  $K_i \cup K_{i+1}$  represents the concatenation of  $K_i$  and  $K_{i+1}$ . Each nugget contains a sequence number, and the last fragment contains a CRC (Cyclic Redundancy Checksum).
- $v$  then selects  $k$  proxies using one of the two approaches presented in the following section to transmit these  $k$  key nuggets to  $u$ .
- Upon receiving all  $k$  nuggets, node  $u$  reconstructs the key  $K$  based on the sequence number carried by each nugget, verifies its correctness using the CRC field and uses it to securely communicate with  $v$ .

Notice that the proposed scheme does not depend on the algorithm used to produce a key. Consequently, any pre-assigned algorithm to produce a secure key can be adopted. An issue, which is not addressed in above scheme, is how to select  $k$  proxies. We propose two simple methods to solve this issue. Notice that if  $u$  and  $v$  share a key,  $v$  can act as its own proxy.

The basic steps of first method to discover  $k$  proxies can be described as following:

- $v$  randomly selects  $k$  neighbors (or  $k - 1$  depending on whether or not  $v$  acts as its own proxy for one key nugget) and sends out request-for-proxy packets containing key IDs from both  $u$  and  $v$ .
- Each recipient examines the ID list to see if it shares keys with both  $u$  and  $v$ .
  - If it does, it responds to  $v$  with key ID that is chosen to communicate with  $v$ ,
  - If it does not, or it has received the same request from  $v$ , it forwards this request to a random neighbor other than the sender.

The procedures used by node  $v$  and candidate node to select  $k$  proxies are outlined in Algorithm 1.

The second method is described as following:

- $v$  creates a request packet and set its Time-To-Leave (TTL) field to  $t$  before locally flooding it into the network. The value of  $t$  may be set to reflect the density of the node within the neighborhood. For dense network, the value of  $t$  should be small while large value of  $t$  may be required for sparse networks.
- Nodes which receive a request packet respond with positive acknowledgment only if they share a key with  $u$  and a key with  $v$  respectively.
- Upon receiving  $k$  positive acknowledgment,  $v$  selects the sender of these acknowledgments as  $k$  proxies.<sup>1</sup>

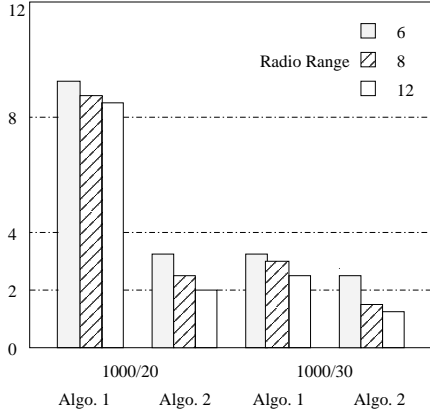
Based on Algorithm 1, node  $v$  selects  $k$  neighbors and sends each one of them a proxy request packet. Nodes can be repeatedly selected if  $v$  has less than  $k$  direct neighbors. Notice that a request copy ceases to travel when received by a proxy. Consequently, at most  $k$  copies of the original requests exist in the network at any time. However, depending on the key distribution, a request may incur a large delay before discovering a proxy. If the probability of two nodes sharing keys is  $p$ , then the probability that a node shares key with two other nodes is  $p^2$ . On average, one request will need to travel  $\frac{1}{p^2}$  nodes on average to find a proxy.

The Algorithm 2 discovers proxies faster than Algorithm 1. This is specially true in dense WSNs. This algorithm, however, requires more nodes than Algorithm 1 for local flooding.

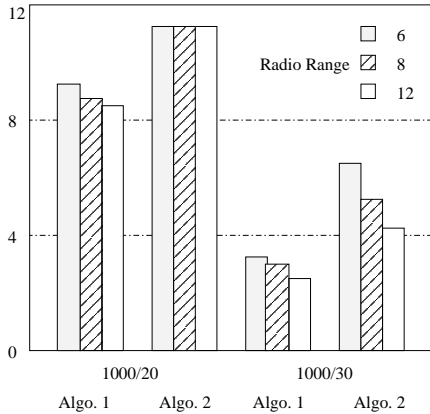
A simulation experiment was set up to compare the performance of these two algorithms. In this experiment, 1000

<sup>1</sup>Notice that other schemes to select  $k$  proxies can be used to satisfy specific requirements such as power awareness, shortest paths, etc.

nodes are randomly distributed over a  $100 \times 100$  square area. The radio range was varied to be 6, 8 and 12 to generate networks with different densities. In this experiment, a path key is fragmented into 5 nuggets, therefore, 5 proxies are necessary to communicate the path key between two end nodes. A 100 pairs of end nodes are randomly selected for 1000/20 and 1000/30 choices of pool size and ring size. Figure. 2 shows the average number of hops to find a proxy. Figure. 3 depicts the average number of nodes involved in discovering one proxy.



**Figure 2. Average number of hops to find a proxy**



**Figure 3. Average number of nodes involved to find a proxy**

The result shows if  $p$  is large, the first approach is preferred, while the second approach should be used if the network is dense. It is therefore important that the choice of parameter  $p$  and the proxy selection method be carefully decided prior to deployment. Alternatively, a node can dynamically adapt its proxy selection strategy to use the appropriate method based on current characteristics of the network.

In further considering the simulation result, it must be pointed out that the proxies discovered by the first ap-

proach may not be physically located many hops away from node  $v$  as Figure. 2 leads to believe. These proxies may actually reside within the vicinity of node  $v$ , but have been discovered through a lateral path connecting neighboring nodes located within a small number of physical hops away from node  $v$ . In fact, based on the simulation results, the sets of proxies discovered by these two approaches exhibit a large overlap.

Using either approach, only local nodes are involved in Path Key establishment. Consequently the performance of the proposed scheme is independent of the network size. It only depends on the key pre-distribution as such the scheme scales to large size networks.

---

**Algorithm 1 K\_Proxies:** Generate  $k$  request to discover  $k$  proxies

---

- 1: **Define:**
  - 2:  $u, v$ : Two end nodes to set up a path key.
  - 3:  $w$ : Proxy candidate.
  - 4:  $ID_u$ : Key ID list for node  $u$ .
  - 5:  $ID_v$ : Key ID list for node  $v$ .
  - 6:  $ID_{self}$ : Key ID list for one node of itself.
  - 7:  $R$ : Request to set up path key
  - 8:  $neighbors_x$ : 1-hop neighbors of any node  $x$ .
  - 9:  $k$ : Number of proxies need to be found.
- 
- 10: **K\_Proxies** ( $k$ ): Executed at node  $v$
  - 11: **for**  $i = 1$  to  $k$  **do**
  - 12:     Randomly select a node in  $neighbors_v$ , send  $R$
  - 13: **end for**
  - 14: **if** Receive positive ACK from node  $w$  **then**
  - 15:     Register  $w$  as a proxy
  - 16: **end if**
  - 17: **Check\_1** ( $R$ ): Executed at all nodes receiving  $R$
  - 18: **if**  $R$  is not seen before **then**
  - 19:     **if**  $ID_{self} \cap ID_u$  is not empty **then**
  - 20:         **if**  $ID_{self} \cap ID_v$  is not empty **then**
  - 21:             register itself as a proxy for node pair  $u$  and  $v$
  - 22:             Send back positive ACK to node  $v$
  - 23:             Exit the procedure
  - 24:         **end if**
  - 25:     **end if**
  - 26: **end if**
  - 27:     Randomly select a neighbor other than the sender to forward  $R$
- 

### 3.2 Security analysis

The security analysis of our scheme focuses on two aspects, namely secrecy or privacy of the system and security against node capture. With respect to secrecy, the scheme must not allow any node other than the end nodes to know the shared path key. The second aspect focuses on the like-

---

**Algorithm 2 LocalFlood:** Incrementally discover  $k$  proxies by local flooding

---

```

1: Define:
2:    $u, v$ : Two end nodes to set up a path key.
3:    $w$ : Proxy candidate.
4:    $ID_u$ : Key ID list for node  $u$ .
5:    $ID_v$ : Key ID list for node  $v$ .
6:    $ID_{self}$ : Key ID list for one node of itself.
7:    $R$ : Request to set up path key
8:    $t$ : TTL(Time To Leave) in each packet.
9:    $Timeout(t)$ : Timeout for  $t$ -hop communication
10:   $k$ : Number of proxies need to be found.
11:   $c$ : counter variable

12: LocalFlood ( $t, k$ ): Executed at node  $v$ 
13:   Broadcast request including  $ID_u$  and  $ID_v$  to set up
14:   path key with  $TTL = t$ 
15:    $c \leftarrow 0$ 
16:   while NOT Timeout( $t$ ) do
17:     if  $c == k$  then
18:       break
19:     end if
20:     if Receive positive ACK from node  $w$  then
21:        $c \leftarrow c + 1$ 
22:       Register  $w$  as a proxy
23:     end if
24:   end while
25:   if  $c != k$  then
26:     Increase  $t$ 
27:      $LocalFlood(t, k)$  {incrementally flood the local
28:     network}
29:   end if

29: Check_2 ( $R$ ): Executed at all nodes receiving  $R$ 
30:   if  $R$  is not seen before then
31:     if  $ID_{self} \cap ID_u$  is not empty then
32:       if  $ID_{self} \cap ID_v$  is not empty then
33:         register itself as a proxy for node pair  $u$  and  $v$ 
34:         Send back positive ACK to node  $v$ 
35:       end if
36:     end if
37:   end if
38:   if  $TTL != 0$  then
39:     Reduce TTL
40:     broadcast  $R$ 
41:   end if

```

---

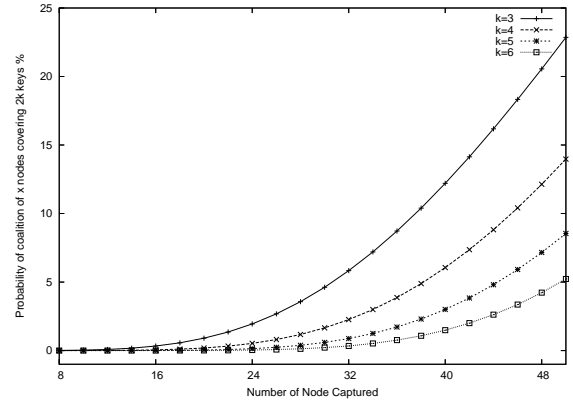
likelihood that an attacker who captures a certain number of nodes may be able to obtain the key.

To evaluate the secrecy of the system, we determine the probability that  $x$  collusive nodes may cover all the keys used to encrypt  $k$  nuggets during the path key establishment phase, thereby violating the end nodes' exclusive path key sharing property. The vulnerability of the system to node capture is measured by computing the likelihood that an attacker who captures  $x$  nodes may obtain all  $k$  key nuggets.

For simplicity, we assume that there are  $2k$  distinct keys used to secure key nuggets by  $k$  proxies. Consider a set of  $x$  collusive nodes. The probability,  $Pr$ , for one of the  $2k$  keys to be installed onto a given network node is  $Pr = \frac{\text{Key ring size}}{\text{Key pool size}} = \frac{R}{P}$ . Then the probability of this key is contained in the union of the  $x$  collusive nodes is  $1 - (1 - Pr)^x$ . Therefore, the probability,  $Px$ , that colluding  $x$  nodes cover all  $2k$  keys is:

$$Px = \left(1 - \left(1 - \frac{R}{P}\right)^x\right)^{2k} \quad (1)$$

Note this probability is independent of the number of nodes deployed. As such, this probability defines the system security for a given key pool size and key ring size. Furthermore, these  $k$  proxies may use less than  $2k$  keys to secure  $k$  key nuggets. Consequently, Equation 1 gives a lower bound of the probability that a set of  $x$  collusive nodes cover all the keys used to securely set up the path key.



**Figure 4. Probability of  $x$  collusive nodes covering all  $2k$  keys**

Figure 4 plots the probability of  $x$  collusive nodes covering all  $2k$  keys. We observe that as  $x$  increases,  $Px$  increases rapidly. It is desired that we keep  $Pr$  small, however, this will affect the choice of method to discover the proxies. It is left to the designer to choose appropriate network specification given a required security level. Furthermore, we can use the cooperation scheme in [6], whereby a proxy uses all the keys that it shares with one end node to encrypt the key nugget to further reduce the likelihood of all  $2k$  keys being covered.

Another observation, which can be made based on the results of Figure 4, is that even when  $k = 3$ , a set of 50

colluding nodes is required to cover all  $k$  key nuggets, with probability of 22.9%. It is therefore unlikely a smaller number of colluding nodes can determine the Path Key by overhearing the traffic.

The vulnerability of the network to node capture depends on the ability of the attacker to acquire the key nuggets directly. If either  $u$  or  $v$  is captured, the path key is revealed. In a network of  $n$  nodes, if  $x$  ( $k < x$ ) nodes are captured, the probability,  $P1$ , that one or both end nodes are among these nodes is:

$$P1 = \frac{\binom{2}{1}\binom{n-2}{x-1} + \binom{2}{2}\binom{n-2}{x-2}}{\binom{n}{x}} = \frac{2n-x-1}{x-1} \times \frac{\binom{n-2}{x-2}}{\binom{n}{x}} \quad (2)$$

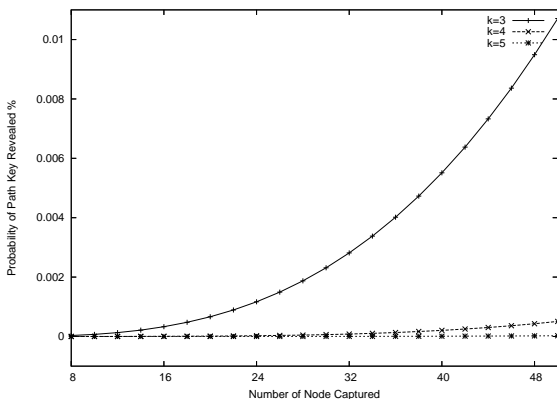
The probability,  $P2$ , that  $x$  nodes containing no end nodes but covering all  $k$  proxies is:

$$P2 = (1-p) \times \frac{\binom{n-k-2}{x-k}}{\binom{n}{x}} \quad (3)$$

where  $p$  is the probability that two nodes share keys. The factor  $(1-p)$  in Equation 3 accounts for the fact that  $k$  proxies are used. Hence, the probability  $Pc$  of all key shared being revealed after capture of  $x$  nodes is:

$$Pc = P1 + P2 = \frac{2n-x-1}{x-1} \frac{\binom{n-2}{x-2}}{\binom{n}{x}} + (1-p) \frac{\binom{n-k-2}{x-k}}{\binom{n}{x}} \quad (4)$$

We can see that the first term in Equation 4 is solely dependent on the scale of the network deployment and the number of nodes captured. No scheme can protect the capture of communicating end nodes unless the nodes are tamper-proof. We are therefore more interested in the second term which relates the number of paths used and the scale of the system. It can be noted that the factor  $(1-p)$  is omitted because once the key pool size and key ring size are chosen, this factor becomes a constant. The plot describing the variation of  $P = \frac{\binom{n-k-2}{x-k}}{\binom{n}{x}}$  as a function of  $x$  is depicted in Figure. 5, for  $n = 1000$  and various values of  $k$ .  $P$  is the probability that  $k$  proxies are among those  $x$  captured nodes.



**Figure 5. Probability of Path Key revealed after  $x$  nodes being captured**

Based on the result, a satisfactory security level ( $5.1 \times$

$10^{-4}\%$ ) can be achieved even when a large percentage of nodes (5%) are captured and  $k$  is small ( $k = 4$ ).

## 4. Conclusion and future work

This paper addresses the path-key establishment exposure problem commonly encountered in key pre-distribution schemes in WSNs. We propose a Path Key Establishment scheme, which uses multiple secured paths for the negotiation and exchange of symmetric keys between end nodes. Since the scheme ensures that each key share can be revealed only to one node on each path, the exposure of that key nugget is minimized. The analysis shows that the proposed scheme can greatly improve the security of key establishment. Furthermore this scheme assumes no specific routing protocols thus it is not dependent on the physical topology of the network. As long as the network is connected and there are enough nodes deployed, the proposed scheme can be incorporated to most key pre-distribution schemes without significant changes.

Currently, the proposed scheme cannot defend against Denial of Service attacks, such as the case when an attacker lies on one or multiple paths from the proxies to the end nodes and drops packets. This can be overcome by inserting redundancy into the key nuggets such that not all  $k$  nuggets are needed to recover the path key. Furthermore, node-disjoint paths are preferred to transport these key nuggets. We will study this as part of our future work.

## 5. REFERENCES

- [1] D. W. Carman, P. S. Kruus, and B. J. Matt. Constrains and approaches for distributed sensor network security. *NAI Labs Technical Report 00-010*, September 2000.
- [2] Haowen Chan, Adrian Perrig, and Dawn Song. Random key predistribution schemes for sensor networks. *IEEE Symposium on Security and Privacy*, pages 197–213, May 2003.
- [3] Wenliang Du, Jing Deng, Yunghsiang S. Han, Shigang Chen, and Pramod K. Varshney. A key management scheme for wireless sensor networks using deployment knowledge. *Proceedings of the IEEE INFOCOM*, March 2004.
- [4] Wenliang Du, Jing Deng, Yunghsiang S. Han, and Pramod K. Varshney. A pairwise key pre-distribution scheme for wireless sensor networks. *Proceedings of the 10th ACM Conference on Computer and Communication Security (CCS)*, pages 42–51, October 2003.
- [5] Laurent Eschenauer and Virgil D. Gligor. A key-management scheme for distributed sensor networks. *Proceedings of the 9th ACM conference on Computer and Communication Security*, pages 41–47, November 2002.
- [6] R. D. Pietro, L. V. Mancini, and A. Mei. Random key assignment for secure wireless sensor networks. *ACM Workshop on Security of Ad Hoc and Sensor Networks*, October 2003.

- [7] Sencun Zhu, Shouhuai Xu, Sanjeev Setia, and Sushil Jajodia. Establishing pairwise keys for secure communication in ad hoc networks: a probabilistic approach. *proceedings of the 11th IEEE International Conference on Network Protocols(ICNP'03)*, November 2003.
- [8] Hui Ling and Taieb Znati. End-to-End Pairwise Key Establishment using Multi-path in Wireless Sensor Network. *Proceedings of the 2005 IEEE Global Communications Conference(GLOBECOM 2005) (To appear)*, December 2005.