Chapter 7

Key Management for Wireless Sensor Networks in Hostile Environments

Michael Chorzempa, Jung-Min Park, Mohamed Eltoweissy, and Y. Thomas Hou

Contents

7.1 Abstract

Large-scale wireless sensor networks (WSNs) are highly vulnerable to attacks because they consist of numerous miniaturized resource-constraint devices, interact closely with the physical environment, and communicate via wireless links. These vulnerabilities are exacerbated when WSNs have to operate unattended in a hostile environment, such as battlefields. In such an environment, an adversary poses a physical threat to all the sensor nodes; that is, an adversary may capture any node compromising critical security data, including keys used for confidentiality and authentication. Consequently, it is necessary to provide key management services to WSNs in such environments that, in addition to being efficient, are highly robust against attacks. In this chapter, we illustrate a key management design for such networks by describing a self-organizing key management scheme for large-scale WSNs, called *Survivable and Efficient Clustered Keying* (*SECK*). SECK is designed for managing keys in a hierarchical WSN consisting of low-end sensor nodes clustered around more capable gateway nodes. Using cluster-based administrative keys, SECK localizes the impact of attacks and considerably improves the efficiency of maintaining fresh session keys.

7.2 Introduction

Key management is crucial to the secure operation of wireless sensor networks (WSNs). A large number of keys must be managed in order to encrypt and authenticate all sensitive data. The objective of key management is to dynamically establish and maintain secure channels among communicating parties. Typically, key management solutions use administrative keys (a.k.a. key encryption keys) to securely and efficiently (re-)distribute and, at times, generate the secure channel session keys (a.k.a. data encryption keys) to the communicating parties. Session keys may be pair-wise keys used to secure a communication channel between two nodes that are in direct or indirect communication [3,4,19,21], or they may be group keys [17,18,31,32]

shared by multiple nodes. Network keys (both administrative and session keys) may need to be changed (re-keyed) to maintain secrecy and resiliency to attacks, failures, or network topology changes. Key management entails the basic functions of generation, assignment, and distribution of network keys. It is to be noted that re-keying is comprised essentially of these basic functions.

The success of a key management scheme is determined, in part, by its ability to efficiently survive attacks on the highly vulnerable and resource challenged sensor networks. Key management schemes in sensor networks can be classified broadly into dynamic or static solutions based on whether re-keying (update) of administrative keys is enabled post network deployment. Schemes can also be classified into homogeneous or heterogeneous schemes with regard to the role of network nodes in the key management process. All nodes in a homogeneous scheme perform the same functionality; on the other hand, nodes in a heterogeneous scheme are assigned different roles. Homogeneous schemes generally assume a flat network model, while heterogeneous schemes are intended for both flat as well as clustered networks. Other classification criteria include whether nodes are anonymous or have predeployment identifiers and if, when (pre-, post-deployment, or both), and what deployment knowledge (location, degree of hostility, etc.) is imparted to the nodes. In this chapter we use the primary classification of static versus dynamic keying.

Recently, numerous static key management schemes have been proposed for sensor networks. Most of them are based on the seminal random key predistribution scheme introduced by Eschenauer and Gligor [3]. In this scheme, each sensor node is assigned k keys out of a large pool P of keys in the predeployment phase. Neighboring nodes may establish a secure link only if they share at least one key, which is provided with a certain probability based on the selection of k and P. A major advantage of this scheme is the exclusion of the base station in key management. However, successive node captures enable an attacker to reveal network keys and use them to attack other nodes. Subsequent extensions to that scheme include using key polynomials [21] and deployment knowledge [20] to enhance scalability and resilience to attacks.

Another emerging category of schemes, including the scheme proposed in this chapter, uses dynamic keying and employs a combinatorial formulation of the group key management problem to affect efficient re-keying [10]. While static schemes primarily assume that administrative keys will outlive the network and emphasize pair-wise session keys, dynamic schemes advocate rekeying to achieve attack resiliency in long-lived networks and emphasize group communication keys. Table 7.1 shows the primary differences between static and dynamic keying in performing key management functions. Moharram and Eltoweissy [12] provide a performance and security comparison between static and dynamic keying.

Table 7.1 Key Management Functions in Static and Dynamic Keying

(Admin. Keys Assumed)	*Static Keying*	*Dynamic Keying*
Key assignment	Once at predeployment	Multiple times
Key generation	Once at predeployment	Multiple times
Key distribution	All keys are predistributed to nodes prior to deployment	Subsets of keys are redistributed to some nodes as needed
Re-keying	Not applicable	Multiple times; requires a small number of messages
Handling node capture	Revealed keys are lost and may be used to attack other nodes	Revealed keys are altered to prevent further attacks

SECK (Survivable and Efficient Clustered Keying) is a dynamic key management scheme that is appropriate for a network with a multi-tier hierarchical architecture deployed in a hostile environment. In such a hierarchical network, the bottom tier consists of clusters of sensor nodes, each cluster consisting of many low-end nodes and a more capable cluster head node. In this chapter we focus on robust key management within a cluster. We give more details of the network architecture in Section 7.3.

In a hostile environment, a WSN operates unattended and its nodes are highly prone to capture. SECK, originally proposed in Chorzempa et al. [1], is a self-organizing scheme that sets up key associations in the network clusters, establishes pair-wise and group keys, provides efficient methods for distributing and maintaining session keys, efficiently adds and revokes nodes, provides efficient mechanisms to recover from multiple node captures, and enables location-based reclustering of nodes. Using analytical and simulation results, we show that SECK is robust against the attacks that we identify in the threat model described in Section 7.5 of this chapter. Moreover, our results show that SECK incurs low communications and storage overhead on the sensor nodes.

The most distinguishing feature of SECK is its robustness against node compromises and its ability to recover from those compromises. SECK was designed for WSNs that must be deployed in hostile environments. In such environments, we assume that an adversary poses a physical threat to all the sensor nodes; that is, an adversary may capture any node. The noteworthy advantages of SECK are twofold: (1) SECK requires low communications overhead for key distribution and maintenance; in a hostile environment, keys must be periodically updated to maintain their trustworthiness; SECK is able to efficiently reestablish keys; and (2) SECK is able to efficiently refresh keys, revoke captured nodes, and efficiently reestablish secure group communications after node captures are detected.

In Section 7.3 we describe the WSN architecture that is used throughout the chapter. We describe the fundamental design principles of SECK in Section 7.4, present the threat model in Section 7.5, and give full details of SECK in Section 7.6. In Section 7.7 we discuss SECK's robustness against node captures. In Section 7.8 we discuss energy dissipation properties of SECK. In Section 7.9 we describe related research. Finally, we summarize our findings in Section 7.10.

7.3 The Network Architecture

In a flat network, all nodes are identical and there is no predetermined architecture. Although simple and efficient for small network sizes, the flat network architecture lacks scalability. A multi-tiered architecture provides scalability, notable energy efficiency, and security benefits [6,27,28,33,34]. Recent data aggregation techniques [8], which remove redundancy in collected data, lend themselves to this hierarchical architecture. Also, WSN routing research has shown that using a multi-tiered architecture for routing can prevent premature battery depletion among nodes near the base station, because, in a flat network, these nodes receive significantly higher traffic volume than remote nodes [29,30]. A multi-tiered architecture can also improve a network's robustness against node or key captures by limiting the effects of an attack to a certain portion of the network. For example, in a multi-tiered WSN, nodes are deployed in clusters, and each cluster can establish keys independently of other clusters. Thus, a key compromise in one cluster does not affect the rest of the network. In this section we describe a two-tiered network architecture that is suitable for large-scale WSNs.

Figures 7.1a and 7.1b show the *physical* and *hierarchical* network topology for such a network, respectively. In this architecture, a small number of high-end nodes, called Aggregation and Forwarding Nodes (AFNs), are deployed together with numerous low-end sensor nodes, called micro-sensor nodes (MSNs). In addition, the network includes a globally trusted base station (BS), which is the ultimate destination for data streams from all the AFNs. The BS has powerful data processing capabilities and is directly connected to an outside network. Each AFN is equipped with a high-end embedded processor and is capable of communicating with other AFNs over long distances. The general functions of an AFN are (1) data aggregation for information flows coming from the local cluster of MSNs, and (2) forwarding the aggregated data to the next hop AFN toward the BS. An MSN is a battery-powered sensor node equipped with a low-end processor and mechanisms for short-range radio communications at low data rates. The general function of the MSN is to collect raw data and forward the information to the local AFN. The bottom tier of the network consists of multiple clusters, where each cluster is composed of numerous MSNs

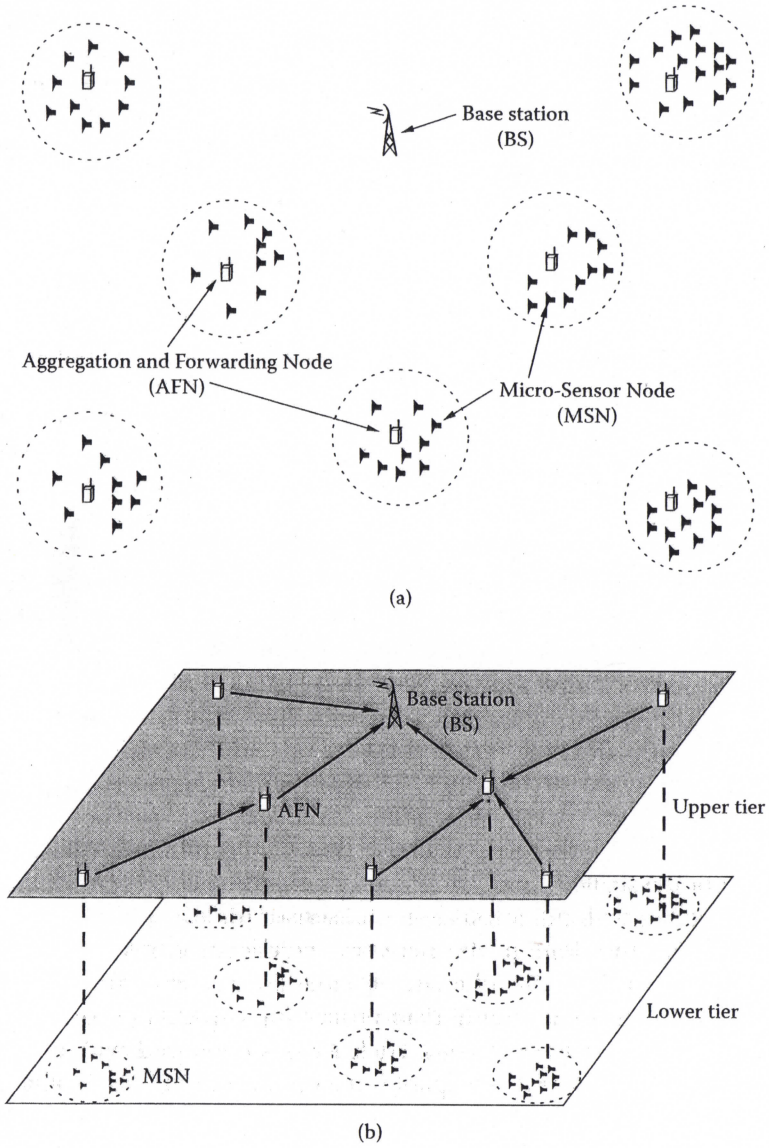

Figure 7.1 A two-tiered wireless sensor network. (a) physical topology and (b) hierarchical view.

clustered around an administrative AFN. Within each cluster, a set of keys must be deployed and managed to secure communications between the MSNs and the AFN. SECK was designed for this purpose.

7.4 Fundamental Design Principles of SECK

In this section we explain the fundamental design principles employed in SECK by describing a basic stripped-down instance of the scheme. In particular, we focus our discussions on the way SECK manages keys within a cluster. Table 7.2 presents the notation used throughout the remainder of the chapter.

To distribute and refresh session (or communication) keys, SECK assigns a set of administrative keys to each node in a network. To manage administrative keys within a cluster, SECK employs the Exclusion Basis System (EBS) [10]. EBS is based on a combinatorial formulation of the group key management problem. It essentially provides a mechanism for establishing the administrative keys held by each node. An EBS-based key management system is defined by EBS(n, k, m) where n is the number of nodes supported in the system, k is the number of keys within each key subset,

Table 7.2 Notation

Constants		*Identifiers*	
n	Number of nodes supported in a given EBS	N_i	i-th node
k	Keys held by each node in EBS	Ka_i	i-th Administrative Key
m	Keys not held by each node in EBS	Kg	Group Key
d	(Connection) degree of an AFN	Kp_i	i-th node's Base Station Pair-wise Key
h	Number of hops between the furthest MSN and AFN_p	Kt_i	i-th node's Tree Administrative Key
		AFN_p	Primary AFN
		AFN_b	Backup AFN

Sets		*Other Notation*	
U	Administrative Key set		
S_i	i-th node's Administrative Key subset		
S_{hi}	AFN's i-th hop neighbor set	$\mathbf{E}_K(M)$	Encrypt message M with key K
S_c	Compromised node set	$A \parallel B$	Bitwise concatenation of A and B
S_{ut}	Uncompromised tree set		

Table 7.3 Sample Administrative Key Subsets using EBS(10,3,2)

	N_1	N_2	N_3	N_4	N_5	N_6	N_7	N_8	N_9	N_{10}
Ka_1	1	1	1	1	1	1	0	0	0	0
Ka_2	1	1	1	0	0	0	1	1	1	0
Ka_3	1	0	0	1	1	0	1	1	0	1
Ka_4	0	1	0	1	0	1	1	0	1	1
Ka_5	0	0	1	0	1	1	0	1	1	1

and m is the number of keys from the global key set not held within a subset. We denote the administrative key set for all nodes in the system as U, where $|U| = k + m$ and the total number of keys is $k + m$. A given EBS supports ${}_{k+m}\mathbf{C}_k$ unique subsets of k key assignments. For full details of the EBS, see [5,10]. We use N_i to denote the i-th node and use S_i to denote its assigned administrative key subset, with $|S_i| = k$. The notation T_i denotes the keys not held by N_i where $T_i = U - S_i$. The following property is the main motivation for utilizing the EBS. A subset of the global key set is uniquely assigned to each node such that the remaining nodes each have at least one of the keys not assigned to that node, that is, for all $j \neq i$, $S_j \cap T_i \neq \emptyset$. We will show that this property makes node revocations and session key replacement very efficient.

The AFN serving as the key management entity for its cluster must store all $k + m$ keys, and each MSN must store k keys. Note that the key subset held by each MSN is unique. This feature is utilized by the AFN to distribute session keys, that is, keys of this subset are used to encrypt the session keys before distribution. We illustrate an instance of EBS(10,3,2) in Table 7.3. The i-th ($1 \leq i \leq 10$) node in the cluster is denoted as N_i, and the j-th ($1 \leq j \leq 5$) administrative key is denoted as Ka_j. An entry marked with a "1" indicates that the node in the corresponding column possesses the administrative key of the corresponding row.

The administrative keys effectively serve as key encryption keys. They allow the AFN to establish, refresh, and revoke session keys of any *degree* belonging to any node. We define the degree of a key as the number of nodes sharing that key. An advantage of EBS is the separation of administrative keys from session keys. A node can possess any number of session keys that it may share with different subgroups of nodes. As will be described in Section 7.6, SECK uses this feature of EBS and assigns subgroup session keys in a manner that enables recovery of nodes even if all administrative keys are compromised due to the capture of multiple nodes.

As stated earlier, to initially distribute a session key to a specific node, the AFN sends that key encrypted with the unique key combination that the node possesses. To initially distribute a cluster-wide key to all members of the cluster, one message is broadcast by the AFN to all members of the

cluster for each administrative key. This requires $k + m$ short broadcasts by the AFN. At any time, to update a session key Kg with Kg' and distribute Kg' to any number of members less than or equal to n, the basic version of SECK executes the aforementioned procedure using a maximum of $k + m$ keys. An example showing the update procedure of the session key, Kg, is shown below:

$$AFN \Rightarrow N_1, \ldots, N_{10}: \quad \mathbf{E}_{Ka_1}(\mathbf{E}_{Kg}(Kg' \parallel ID_{AFN}))$$

$$AFN \Rightarrow N_1, \ldots, N_{10}: \quad \mathbf{E}_{Ka_2}(\mathbf{E}_{Kg}(Kg' \parallel ID_{AFN}))$$

$$AFN \Rightarrow N_1, \ldots, N_{10}: \quad \mathbf{E}_{Ka_3}(\mathbf{E}_{Kg}(Kg' \parallel ID_{AFN}))$$

$$AFN \Rightarrow N_1, \ldots, N_{10}: \quad \mathbf{E}_{Ka_4}(\mathbf{E}_{Kg}(Kg' \parallel ID_{AFN}))$$

$$AFN \Rightarrow N_1, \ldots, N_{10}: \quad \mathbf{E}_{Ka_5}(\mathbf{E}_{Kg}(Kg' \parallel ID_{AFN}))$$

Here, $\Rightarrow$ denotes broadcast transmission. These broadcast messages ensure that only previous holders of Kg will be able to successfully receive the new key, Kg'. If N_1 is captured, or if its keys are compromised, it is necessary to revoke all administrative and session keys held by N_1 and thus evict it from all future secure communications. This ensures forward secrecy. To accomplish this, the AFN needs to replace the administrative keys and the session keys known to N_1. From our running example, evicting N_1 would require the following two transmissions ($m = 2$).

$$AFN \Rightarrow N_1, \ldots, N_{10}:$$

$$ID_{AFN} \parallel \mathbf{E}_{Ka_4}(\mathbf{E}_{Ka_1}(Kd_1') \parallel \mathbf{E}_{Ka_2}(Kd_2') \parallel \mathbf{E}_{Ka_3}(Kd_3')),$$

$$AFN \Rightarrow N_1, \ldots, N_{10}:$$

$$ID_{AFN} \parallel \mathbf{E}_{Ka_5}(\mathbf{E}_{Ka_1}(Kd_1') \parallel \mathbf{E}_{Ka_2}(Kd_2') \parallel \mathbf{E}_{Ka_3}(Kd_3'))$$

From Table 7.3, it can be seen that all remaining nodes will be able to decipher at least one of these messages.

If more than one node is captured within a cluster, two cases arise: (1) non-colluding node captures (e.g., attacks carried out by different adversaries); and (2) colluding node captures. In the latter case, colluding attackers may compromise all administrative keys by capturing only a few nodes (e.g., by capturing nodes N_1 and N_6 shown in Table 7.3, all administrative keys are revealed). In the former case, a maximum of m^y broadcast messages will be needed to evict y nodes at once. From our running example, to evict non-colluding nodes N_1 and N_6, four messages are needed to distribute the five new keys. One message will be doubly encrypted with Ka_2 and Ka_4, the second message with Ka_2 and Ka_5, the third message

with Ka_3 and Ka_4, while the fourth message will be encrypted with Ka_3 and Ka_5.

The basic scheme described above is efficient and functions well, provided that node captures are non-colluding. But in practice, collusion attacks can occur and the key management solution must take this threat into account. The aforementioned scheme has another drawback — it is not resilient against an AFN capture. In Section 7.6, we describe the full-fledged version of SECK that provides effective solutions for both types of attacks (i.e., colluding multiple MSN captures and AFN captures). In the next section we describe the threat model that was considered when designing SECK.

7.5 The Threat Model

We consider an attack scenario where an adversary is able to compromise one or more nodes of a WSN. Specifically, we consider three different cases with differing degrees of severity. Throughout the remainder of the chapter, when stating that a node has been compromised or captured, we assume that all key information held by that node is revealed to the attacker. In the first scenario, an adversary may be able to target and capture an AFN. A less severe attack would be when an attacker captures MSNs within the same cluster. In the third scenario, incurring the least degree of damage, an adversary would simply capture nodes at random throughout the network. One threat that has not been addressed in group key management schemes using administrative keys is the realistic possibility that multiple nodes may be captured before any node capture is detected. Researchers have pointed out that there really is no sure and efficient way to readily detect a node capture [4,13]. Therefore, for a key management solution to be truly effective in a hostile environment, it must recover from multiple node captures.

The attack scenario that possesses the greatest threat to the bottom tier of the network is the compromise of an AFN, that is, the first attack scenario. This requires an adversary to locate and visually distinguish an AFN from a MSN. Then an adversary must extract the sensitive contents of the AFN (e.g., keys). If an AFN capture is not immediately detected, all data collected by MSNs in that cluster will be compromised. After the detection of the AFN capture, the following steps must be executed to restore normal operations of the cluster: (1) notify MSNs of the capture, (2) establish a new AFN for each MSN, and (3) establish a new security relationship between the MSN and the AFN in the second step. Note that in the second step, MSNs are "reclustered" or absorbed into other existing clusters in their vicinity. If reclustering is not supported by the network, all MSNs within the affected cluster are considered off-line until a new AFN is

deployed. The AFN that is captured contains a full set of administrative keys that will need re-keying. To localize the necessary re-keying operations, it is necessary for administrative keys to be independently replaced. If the administrative key sets were globally calculated and distributed to all AFNs, all keys for all clusters would be compromised as a result of a single AFN capture.

In the second attack scenario, MSNs within the same cluster are compromised. In the basic version of SECK described in Section 7.4, it is possible for an adversary to capture not only some, but all of the administrative keys of a cluster with only a few MSN captures. This is possible if the adversary is able to pick the nodes in a strategic manner. In the running example of Section 7.4, the strategic choice would be to compromise N_1 and N_6, which would reveal all five administrative keys. Therefore, if the detection of node captures is not possible, or not prompt, the entire cluster will likely be rendered insecure unless a method of recovery is developed. SECK provides a recovery method to salvage uncompromised MSNs within a cluster when some or even all administrative keys are compromised.

In the third attack scenario, an attacker may compromise nodes randomly throughout the network. A clustered architecture's main advantage, in terms of security, is its ability to localize the effects of attacks on randomly chosen nodes. More discussions on this topic are given in Section 7.7.1. A secondary advantage is that multiple decentralized attacks may not have increased effect compared to a single attack instance. If two adversaries located randomly throughout the network compromise one node each, combining the information obtained from these nodes provides no added benefit, assuming the nodes are not within the same cluster. Of course, this is true only if each AFN generates its administrative keys independently from others.

7.6 The Complete Specification of SECK

We have shown how SECK maintains fresh keys and revokes single users from secure communication. We now describe the mechanisms needed to complete SECK. We start our discussions by describing the full set of keys stored by each MSN to support SECK. We also describe a *location-training* scheme that sets up the network clusters and the coordinate system used to establish administrative keys. Then we describe techniques for replacing administrative keys compromised by multiple MSN captures. Next we describe a reclustering scheme for salvaging MSNs within a cluster after the AFN of that cluster has been captured. Finally, we describe a method to dynamically add an MSN to the network. In Figure 7.2 we show the overall operation and components that constitute SECK.

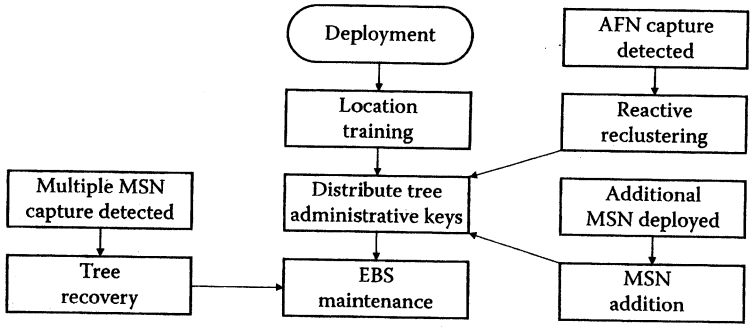

Figure 7.2 Components of SECK.

7.6.1 Required Keys

Initially each MSN will be deployed with the complete administrative key set $\{Ka_1, Ka_2, \ldots, Ka_{k+m}\}$. After a short setup period, only a subset of these keys will be stored. In addition, each MSN will be required to store key Kp_i, which is a pair-wise secret key shared with the BS, and one tree administrative key Kt_i. Kp_i is needed for the reclustering process after the capture of the AFN has been detected. The key Kt_i is used to replace compromised administrative keys after MSN captures have been detected.

7.6.2 Location Training

The location-training scheme will establish a cluster of MSNs around a primary AFN, AFN_p, and assign each MSN a cluster coordinate identifier. This identifier is needed to establish each MSN's administrative key subset, and is also used when routing data to an MSN's primary AFN. In addition, this scheme enables an MSN to store the location of the next-hop MSN, N_x, in the direction of its backup AFN, AFN_b. Each MSN is absorbed into AFN_b's cluster in the event of AFN_p's compromise or failure. A cluster coordinate established is given as (*tree*, *hopcount*), where *tree* is an integer assigned by AFN_p, and *hopcount* is the MSN's distance from AFN_p. We define a *tree* as a set of MSNs that routes packets through the same tree root when forwarding data to the AFN_p, where the tree root is an MSN that is one hop away from the AFN_p. Once a cluster coordinate has been established for each MSN, that coordinate is used to establish the corresponding key subset from the stored global administrative key set.

It is assumed that at this point MSNs have completed neighborhood discovery, and every MSN and AFN is aware of the unique identities (IDs) of all one-hop neighbors through broadcasted "hello" messages (we assume each MSN and AFN is embedded with a unique ID before deployment). Now,

each AFN broadcasts the list of one-hop neighbors that it has discovered to all MSNs within each transmission range. Each entry in the list is a tuple of an MSN ID and its assigned tree number (assignment of tree numbers is described in Section 7.6.3). This broadcast transmission can be expressed as

$$AFN_i \Rightarrow N_1, \ldots, N_m :$$

$$ID_{AFN_i} \| (ID_1, tree_1) \| (ID_2, tree_2) \| \ldots \| (ID_{|S_{b1}|}, tree_{|S_{b1}|}),$$

where m denotes the number of MSNs within the transmitting range of the AFN, and S_{b1} denotes the set of MSNs within one hop of AFN_i. Each one-hop node will serve as a tree root for multi-hop nodes established in that tree. MSNs search this list for their ID and the ID of their discovered neighbors. If a node finds its ID on this list, it assigns its cluster coordinate as $(tree_{ID}, 1)$ and becomes one of the tree roots of AFN_p. If an MSN does not find its ID, but finds the ID of a neighbor, say i, it assigns itself the cluster coordinate $(tree_{ID_i}, 2)$ — the first entry corresponds to the tree number of the neighbor MSN and the second entry indicates the hop count from AFN_p. MSNs with multiple neighbors on the neighbor list of AFN_p should randomly choose which tree to join.

The group of second-hop neighbors, S_{b2}, initiates the propagation of the coordinate establishment message by broadcasting their cluster coordinate and ID_{AFN_p} as follows:

$$\text{For all } N_i \in S_{b2},\ N_i \Rightarrow \text{ neighbors}: \qquad ID_{AFN_p} \| (tree, hopcount)_{N_i}$$

Upon hearing this message, MSNs not yet holding a cluster coordinate will know how many hops away from AFN_p they are, and what their cluster coordinate should be. As these messages propagate, all MSNs will establish their cluster coordinates. An MSN will always forward the first coordinate establishment message it receives. Once an MSN begins to receive additional coordinate establishment messages, it will forward this message only if it is the closest backup AFN heard so far, that is, if $hopcount_{new} < hopcount_b$, where $hopcount_b$ is the number of hops to AFN_b. Each MSN will store ID_{AFN_b}, $hopcount_b$, and the MSN ID of the node that sent the coordinate establishment message containing ID_{AFN_b}. This MSN will serve as N_x toward AFN_b. We describe, in Section 7.6.5, the importance of establishing N_x and AFN_b. Every MSN, with the exception of the $|S_{b1}|$ established root nodes, must broadcast one message when establishing their primary cluster coordinate. Every MSN must transmit at least one additional message when establishing AFN_b. We will assume the case where the optimal AFN_b is established with the first coordinate establishment message received. Then, the total communication overhead for the location training process

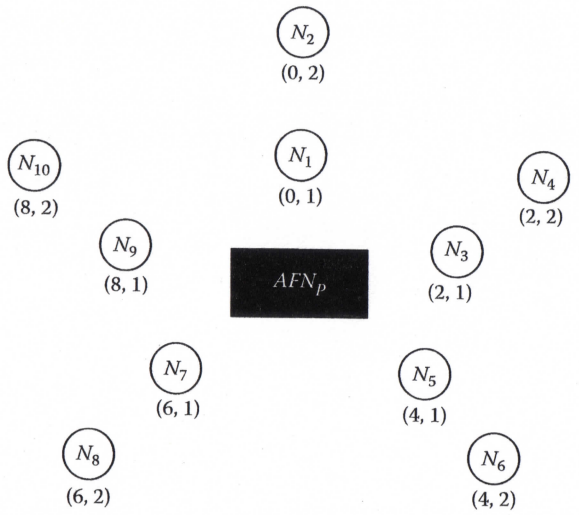

Figure 7.3 Optimal cluster coordinates established in a single cluster.

is $1 + 2n - |S_{b1}|$ transmissions per cluster, where n denotes the number of MSNs in the cluster. Using our running example, Figure 7.3 shows cluster coordinates established within a ten MSN cluster. This figure shows the optimal case, because each node receives a unique cluster coordinate. In the following section we explain how the sum of the entries in each node's cluster coordinate yields a unique subset identifier between 1 and 10.

7.6.3 Establishment of Administrative Keys

The administrative key subset is determined by the assignment of the AFN cluster coordinate. We propose an algorithm for assigning tree numbers based on the expected number of hops within a cluster. We assume that the expected number of hops can be estimated based on the density of AFN deployment relative to MSN deployment. We denote the number of neighbors found by an AFN (or degree of AFN) as d, ($d = |S_{b1}|$) and the maximum number of estimated hops within a cluster as h. We assign tree numbers as $tree_i = i \cdot h$ for $0 \leq i < d$. The administrative key subsets are then determined by the sum of the two elements of a node's cluster coordinate. Because a cluster coordinate consists of *(tree,hopcount)*, an MSN's key subset identifier is calculated as *tree* + *hopcount*. Each unique cluster coordinate within the expected maximum number of hops within a cluster, h, will generate a unique key subset identifier in the range of $[1, d \cdot h]$.

Each AFN is responsible for generating and distributing the tree administrative keys for each tree in its cluster. This requires $d \cdot h$ messages

transmitted by the AFN, one for each of the unique key subset identifiers used in the cluster. The first step is to generate d tree administrative keys, where Kt_j is the tree administrative key for all nodes in the j-th *tree*. Recall that a *tree* consists of all nodes that utilize the same forwarding route. From Figure 7.3, we see that N_1 and N_2 are members of $tree_0$. Each Kt_j is distributed to each N_i in $tree_j$ as follows:

$$\text{For all } N_i \in tree_j,\ \text{AFN} \rightarrow N_i: \qquad \mathbf{E}_{Ka_1}(\mathbf{E}_{Ka_2}(\ldots \mathbf{E}_{Ka_{|S_i|}}(Kt_j)\ldots))$$

where $\rightarrow$ represents unicast transmission and $S_i = \{Ka_1, Ka_2, \ldots, Ka_{|S_i|}\}$. It is important for the administrative key set, U, to be independently updated within each cluster after all tree administrative keys have been established. If the same set of administrative keys is maintained globally, the compromise of a single cluster's keys would compromise the entire network's keys. We described in Section 7.5 that in order to isolate an attack, the administrative keys must be independently generated in each cluster and not shared among clusters.

7.6.4 Administrative Key Recovery

If all administrative keys have been compromised due to multiple node captures within an isolated set of trees, then the remaining trees can be salvaged using their tree administrative keys to reestablish their administrative key subsets.

In Section 7.5 we stated that the capture of a group of MSNs can compromise most or all of the administrative keys in the basic version of SECK. When all of the administrative keys of the cluster have been compromised, even the MSNs that are not captured are excluded from any further communications with the rest of the network. Hence, it is necessary to distribute new session keys to those MSNs. We assume a scenario where an AFN at some point receives notification that a set of nodes, which we denote as S_c, has been compromised, resulting in the compromise of all administrative keys. In response, the AFN computes the set of trees not containing any node from S_c, which we denote as S_{ut}, as

$$S_{ut} = \{tree_i : tree_i \cap S_c = \emptyset, \forall tree_i\}$$

The AFN then creates $|S_{ut}|$ messages, each containing a set of new administrative keys, and transmits the messages to the appropriate trees as shown in the following expression:

$$\text{For all } tree_i \in S_{ut},\ \text{AFN} \Rightarrow tree_i:$$

$$\mathbf{E}_{Kt}(\mathbf{E}_{Ka_1}(Kd_1') \parallel \mathbf{E}_{Ka_2}(Kd_2') \parallel \ldots \parallel \mathbf{E}_{Ka_{k+m}}(Kd_{k+m}'))$$

Note that the technique described above cannot salvage MSNs that belong to a tree in which some of its nodes have been compromised. In Section 7.7.3, we show that the technique described above can salvage a greater percentage of MSNs when the attack is more localized (i.e., concentrated in a specific region of a cluster).

7.6.5 Reactive Reclustering after AFN Capture

Unlike an MSN, an AFN carries out several key tasks that are essential to its cluster. Because the loss or capture of an AFN can incapacitate the entire cluster, giving the MSNs the ability to recover from such a situation greatly improves the survivability of the cluster by removing the single point of failure [15]. To keep the MSNs operational after an AFN capture, we need to recluster the MSNs into neighboring clusters and establish new security relationships between the reclustered MSNs and their respective backup AFNs. We identify two approaches to address this problem. First, MSNs may proactively maintain backup security relationships with neighboring AFNs, similar to the approach in Gupta and Younis [16]. This requires the MSNs to participate in periodic key update procedures with multiple AFNs. The second approach involves MSNs reactively utilizing a trusted third party (TTP) to establish a new security relationship with a backup AFN after the capture of the primary AFN is detected. It is reasonable to assume that, in mission-critical applications, the AFNs would be equipped with tamper-resistant hardware because of their importance. We assume that successful AFN captures are infrequent events and adopt the reactive approach because it does not require the MSNs to maintain backup information.

In SECK's approach, the BS acts as the TTP. The BS establishes a security relationship between the MSN (that is being reclustered), N_i, and AFN_b. The BS authenticates AFN_b and N_i, and distributes a new pair-wise key to those nodes so that a new administrative key subset may be established.

Before describing the recluster procedure, we make some prefatory comments. We assume that the AFNs are each equipped with a high-end embedded processor (e.g., Intel Xscale PXA250) that is capable of executing asymmetric cryptographic operations. Recall that at the completion of the location training procedure, each MSN is aware of the identities of the AFN_B and the next-hop MSN en route to the AFN_b, N_x.

Upon detection that N_i's primary AFN has been captured, N_i constructs a short recovery request message destined for the BS. Then N_i sends the following message to N_x.

$$N_i \rightarrow N_x: \qquad ID_{N_i} \parallel ID_{AFN_b} \parallel nonce_i \parallel MAC_{Kp_i}$$

where MAC_{Kp_i} represents a message authentication code (MAC) generated with N_i's base station pair-wise key, Kp_i. We assume that N_x routes this message to AFN_b in the same manner as forwarding sensed data to AFN_b for data aggregation. After receiving this message, AFN_b digitally signs this message, appends the signature to the message, and forwards the following message to the BS:

$$AFN_b \rightarrow \text{BS}: \qquad ID_{N_i} \parallel ID_{AFN_b} \parallel nonce_i \parallel MAC_{Kp_i} \parallel Sign_{AFN_b}$$

where $Sign_{AFN_b}$ represents AFN_b's signature. When the BS receives this message, it authenticates N_i using the MAC and then authenticates AFN_b using the signature. If both nodes are authenticated, the BS generates a secret key $K_{AFN_b-N_i}$ and transmits it to both AFN_b and N_i as shown in the following equations:

$$\text{BS} \rightarrow AFN_b:$$

$$ID_{N_i} \parallel ID_{AFN_b} \parallel \mathbf{E}_{K_{AFNb+}}(K_{AFN_b-N_i} \parallel ID_{N_i} \parallel ID_{AFN_b} \parallel nonce)$$

$$\text{BS} \rightarrow N_i:$$

$$ID_{N_i} \parallel ID_{AFN_b} \parallel \mathbf{E}_{Kp_i}(K_{AFN_b-N_i} \parallel ID_{N_i} \parallel ID_{AFN_b} \parallel nonce)$$

where K_{AFNb+} represents the public key of AFN_b. Using $K_{AFN_b-N_i}$, N_i and AFN_b can now establish a security relationship. Using $K_{AFN_b-N_i}$, AFN_b finalizes N_i's membership in its cluster by sending N_i a subset of administrative keys and the corresponding subset identifier. It is noted that if an EBS reaches its maximum number of nodes, that is, $n = {}_{k+m}\mathbf{C}_k$, adding a new node will require the expansion of the EBS by adding a new key. For brevity, we do not describe the process of extending an EBS in this chapter; we simply assume that the size of a cluster's initial EBS will be sufficient for node additions and refer the reader to Eltoweissy et al. [10] for an EBS expansion mechanism.

7.6.6 MSN Addition

Throughout the lifetime of a WSN, it may be necessary to deploy additional MSNs. We propose a way of adding nodes to an existing WSN. We assume that MSNs are randomly deployed and that the MSN's resulting cluster is unknown. In SECK, each cluster maintains a unique and private set of keys. For this reason it is impossible for a new MSN to be predeployed with any keys that will enable authentication with a MSN or AFN directly. However, each new MSN will contain a unique base station pair-wise key, Kp_i, which allows the BS to facilitate a new security relationship between the new MSN and existing AFN in the same way as the reclustering process

described in Section 7.6.5. The only difference between deploying a new MSN and reclustering an MSN is that the newly deployed MSN has no knowledge of a backup AFN. Once a newly deployed MSN knows which AFN's cluster to join, it starts the reclustering process described above.

To determine which AFN's cluster is most appropriate to join, a new MSN, N_{new}, conducts a survey of neighbor nodes' cluster status. First, N_{new} broadcasts a short "hello" message to announce its presence. Each neighbor, $neighbor_i$, that overhears this message replies with its cluster information as follows:

$$neighbor_i \rightarrow N_{new}: \qquad ID_{N_i} \parallel ID_{AFN_p} \parallel hopcount_{N_i}$$

where ID_{AFN_p} is $neighbor_i$'s primary AFN, and $hopcount_{N_i}$ is the hopcount in $neighbor_i$'s cluster coordinate identifier. N_{new} uses these replies to determine the most appropriate cluster to join and the most efficient neighbor to use as a next-hop node to that cluster's AFN. N_{new} determines the most appropriate AFN, AFN_{max}, based on the number of neighbors belonging to AFN_{max}. N_{new} selects the most efficient neighbor based on the neighbor's *hopcount* to AFN_{max}. The neighbor with the smallest *hopcount* is the most efficient node to use as a next-hop node in the direction of AFN_{max}. Now that the next-hop node to AFN_{max} is known, N_{new} conducts the reclustering procedure described in Section 7.6.5.

7.7 Robustness of SECK against Node Capture

Node capture in hostile environments is inevitable, and an effective key management scheme should be able to recover from such attacks to be effective. We describe some of the inherent security advantages of utilizing a clustered and hierarchical network architecture. Then, using the threats identified in Section 7.5, we analyze how well SECK recovers from those attacks.

7.7.1 Robustness of a Clustered Architecture

A clustered and hierarchical framework for WSNs provides many beneficial security properties. Isolation is the primary benefit of a clustered key management scheme. Each AFN is responsible for independently calculating and periodically distributing new administrative keys. Hence, an attack that yields keys within one cluster will not impact any other cluster in the network. An adversary must perform a global attack to completely compromise the network. This is not the case for most nonhierarchical key management schemes. In nonhierarchical key management solutions, a localized attack often has a global impact on the network's keys [3,17,19,21].

7.7.2 Robustness against MSN Node Capture

In the example given in Table 7.3, it is possible for an adversary to capture all the administrative keys of a cluster with only a small number of node captures by strategically selecting the nodes to capture. For example, the capture of N_1 and N_6 would reveal all five administrative keys. Fortunately, such attacks are difficult to carry out; to be successful, an adversary needs to know which administrative keys are stored in each MSN. Of course, the adversary can always randomly choose the nodes to capture and hope that a large proportion of administrative keys is revealed by those captures. Using simulations, we show that when the nodes are compromised randomly, then the proportion of compromised keys is commensurate with the well-known probabilistic key distribution scheme proposed in Eltoweissy and Gligor [3].

Eschenauer and Gligor (EG) [3] proposed a probabilistic key establishment technique for WSNs in which pair-wise secret keys are selected from a global key pool. In the past few years, similar schemes have been proposed [19,21]. The EG scheme has been shown to have acceptable key resiliency properties. That is, the capture of a limited number of nodes does not reveal much key material of other nodes. Figure 7.4 compares three instances of SECK with the EG scheme by plotting their ratio of keys captured as a function of ratio of nodes captured. For SECK, the keys being considered are administrative keys; for EG, the keys being considered are pair-wise

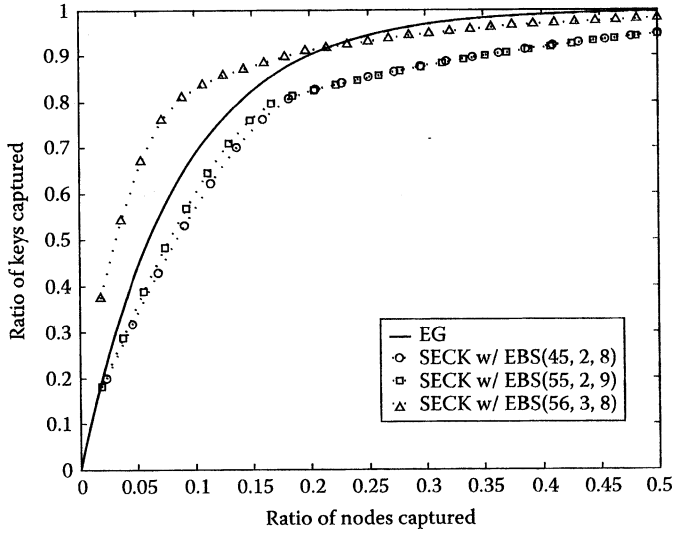

Figure 7.4 Expected ratio of keys captured versus ratio of nodes captured.

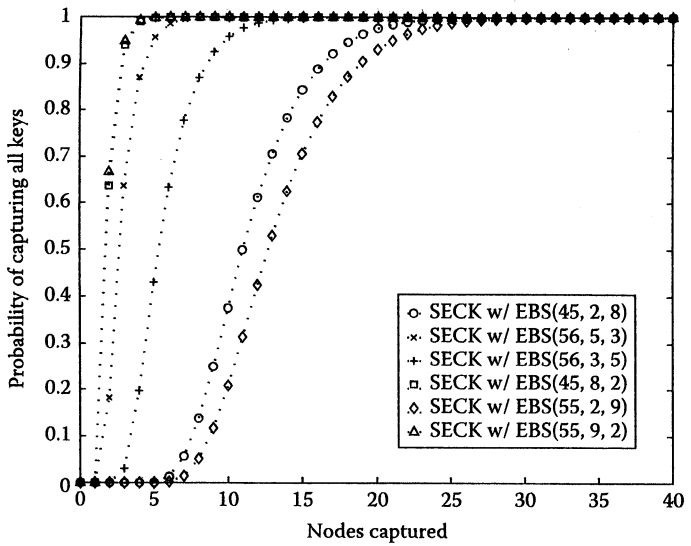

Figure 7.5 Probability that all keys have been compromised versus the number of nodes captured.

secret keys. Figure 7.4 shows that SECK's robustness against random node (MSN) captures is commensurate with that of the EG scheme.

Now we attempt to answer the following question: how many MSNs must be captured before all the administrative keys of a cluster ($k + m$ keys in total) are compromised? In Figure 7.5, the probability of capturing all the administrative keys as a function of the number of nodes captured is plotted for several instances of SECK, with all instances supporting approximately 50 nodes. We are interested in instances of SECK that support about 50 nodes, as that is the expected size of a typical cluster that we envision. The figure shows that the ratio k/m has a direct impact on the robustness of SECK against node captures — decreasing the ratio improves robustness against node captures and increasing the ratio has the opposite effect. However, setting the ratio k/m to a low value incurs a cost — as the ratio decreases, the communication overhead required to distribute session keys increases. One can see that there exists a communication overhead versus node capture resiliency trade-off. Further discussions on this issue are continued in the next section.

7.7.3 Evaluation of the Administrative Key Recovery Procedure

Recall that SECK generates d tree administrative keys for a cluster consisting of n MSNs in which every MSN is within h hops of its primary AFN.

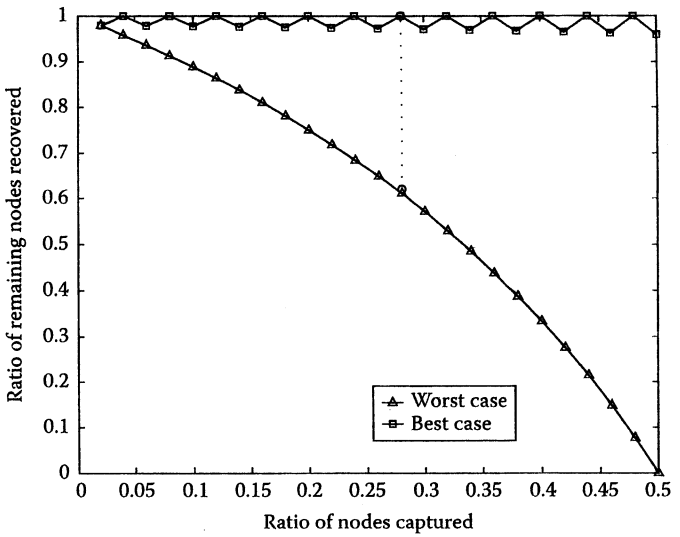

Figure 7.6 Administrative key recovery procedure evaluation.

Figure 7.5 shows that SECK, using EBS(55,2,9), provides the most resiliency against node captures. If 14 MSNs have been randomly captured in this case, there is a good chance (>60 percent) that the complete administrative key set has been compromised. If all the administrative keys of a cluster are compromised, the network needs to execute the MSN administrative key recovery procedure. In the following paragraphs we discuss the effectiveness of the administrative key recovery procedure in two distinct cases — best and worst case scenarios — assuming that x MSNs have been captured.

The worst case occurs when each of the x captures occurs in separate trees. This leaves $d - x$ trees unaffected, and $(d - x) \cdot h$ nodes can be recovered. If we approximate d with n/h, the ratio of nodes that can be recovered is $(n - x \cdot h)/(n - x)$. Suppose that every MSN in the cluster is within two hops of the AFN (i.e., $h = 2$) and EBS(55, 2, 9) is used. Then our procedure recovers 0.66 of the uncompromised nodes in the worst case.

The best case occurs when the attack is completely localized. That is, all nodes within a single tree are captured before the attacker moves on to the next tree. This will affect $\lceil x/h \rceil$ trees, leaving $d - \lceil x/h \rceil$ trees unaffected. If we again approximate d with n/h, the ratio of nodes that can be recovered is $(n - h \cdot \lceil x/h \rceil)/(n - x) \approx 1$. From the above analysis we can observe that SECK's recovery procedure performs best in localized attacks.

In Figure 7.6, we have plotted the ratio of recoverable nodes in the best and worst case attack scenarios. We assume that EBS(55, 2, 9) is employed. In the figure, we have highlighted the case where 14 nodes have been captured. In practice, the actual recovery ratio is expected to be somewhere

between 0.6 and 1.0 when 14 nodes are captured. As expected, as the ratio of nodes captured increases, the ratio of recoverable nodes decreases.

7.8 Evaluation of Communication and Storage Overhead

7.8.1 Energy Dissipation

In typical key management schemes, the energy required for computation is three orders of magnitude less than that required for communication [15]. Moreover, the amount of energy consumed for computation varies significantly with hardware. Hence, we only consider energy costs associated with radio signal transmission and reception, and do not consider energy costs associated with computation. To calculate the amount of energy dissipated during the execution of SECK, we use the power usage specification of Sensoria's WINS NG: the RF radio component consumes 0.021 mJ/bit for transmission, and 0.014 mJ/bit for reception when operating at 10 kbps [26]. We make the same assumptions as in Carman et al. [15] with regard to the following message element sizes:

- All node IDs are 64 bits.
- All nonces are 64 bits.
- All symmetric keys are 128 bits.
- The *tree* identifier is 32 bits.
- The *hopcount* is 32 bits.
- All MACs are 128 bits.
- The RSA modulus used for digital signatures is 1024 bits.

Table 7.4 shows the amount of energy dissipated by a single node to complete one instance of each process (six processes are listed). We assume that EBS(55,2,9) is employed in a cluster consisting of 50 MSNs with $h = 2$. Each AFN has a neighbor degree of 20; that is, $|S_{b_1}| = 20$. In the following we discuss the major factors that contribute to the energy consumption for each of these processes.

The energy required for the location training process is affected by the efficiency at which the MSNs find their optimal backup AFN. We assume that the backup AFN is found when a node receives the second coordinate establishment message. To distribute the tree administrative keys to all MSNs in its cluster, an AFN must generate one message for each MSN in its cluster. Hence, the energy consumed by each AFN is directly proportional to the number of MSNs in its cluster. The communication overhead required for the EBS maintenance process is the message transmission needed for refreshing all administrative keys, and therefore this overhead depends on the dimension of the EBS that is used. The energy cost of the tree recovery

Table 7.4 SECK Communication Energy Consumption (in mJ)

		Transmission	*Reception*	*Total*
Location training	AFN	55.10	N/A	55.10
	MSN	1.10	36.74	37.84
Distribute tree administrative key	AFN	336.00	N/A	336.00
	MSN	N/A	4.48	4.48
EBS maintenance	AFN	44.35	N/A	44.35
	MSN	N/A	29.56	29.56
Tree recovery	AFN	29.57	N/A	29.57
	MSN	N/A	19.71	19.71
Reactive reclustering	AFN	38.98	11.65	50.63
	MSN	17.47	11.65	29.12
MSN addition	AFN	38.98	11.65	50.63
	MSN	20.67	11.65	32.32

process shown in Table 7.4 is the energy dissipated by the AFN for each *tree* recovered. The energy cost of reactive reclustering calculated in Table 7.4 is the energy needed to forward the recovery messages needed to recover one MSN. Therefore, the total energy dissipated during the cluster recovery process depends on the number of MSNs being recovered with a specific backup AFN. Finally, the energy cost during node addition is calculated as the energy needed to respond to, then to forward the recovery messages of one newly added MSN. It can be seen from Table 7.4 that SECK effectively offloads much of the energy-intensive operations to the more capable AFN.

7.8.2 Storage Overhead

Prior to deployment, each MSN needs to store a key subset matrix and the complete administrative key set. The key subset matrix is a $(k+m) \times n$ bit matrix that identifies the keys associated with each subset. Table 7.3 is a sample 5×10 key subset matrix. The key subset matrix requires $n \cdot (k+m)$ bits to store, and the complete administrative key set requires $128 \cdot (k+m)$ bits to store (here, we assume that 128-bit AES keys are used). Additionally, one 128-bit base station pair-wise key is stored at each MSN, for a total initial storage requirement of $((128 + n) \cdot (k+m) + 128)$ bits. With this formula, one can calculate that each MSN would need to store 266 bytes of initial keying material if EBS(55,2,9) is employed.

After deployment, each MSN deletes most of its initial keying material immediately after the conclusion of the location training processes. Specifically, each MSN deletes its key subset matrix and the unused administrative keys. Recall that each MSN is assigned one 128-bit tree administrative key after deployment. Assuming that EBS(55,2,9) is employed, each MSN would

need to store 64 bytes of keying material or four 128-bit keys at the end of the location training process.

7.8.3 Comparison of Communication Overhead

The keying communication overhead of SECK is incurred during (1) the initial key establishment phase and (2) periodic key maintenance procedures. We compare the communication overhead incurred in (1) and (2) with those incurred in the Localized Encryption and Authentication Protocol (LEAP) [13] and Simple Key Distribution Center (SKDC) [15], respectively. A unique feature of SECK is its use of both administrative keys and session keys. Because of this feature, we cannot compare SECK, in its entirety, with a single scheme. Instead, we decompose SECK into (1) and (2) and compare the two constituent parts with LEAP and SKDC that respectively carry out similar functions. LEAP supports multiple levels of communication through multiple degrees of key sharing. SECK provides a similar level of flexibility within a clustered architecture. SKDC is a session key distribution scheme that utilizes a leader node to distribute the session key. It was shown in Carman et al. [18] that this basic method used by SKDC is the most efficient means to distribute a session key.

7.8.3.1 Key Establishment

In this subsection we compare the communication overhead incurred by SECK during the key establishment process with that of LEAP. LEAP is a well-known key management scheme that provides communication flexibility by establishing multiple classes of keys. In SECK, a location-training process is executed to establish administrative key sets; then a distribution process is executed to distribute a session key. LEAP goes through similar key establishment processes to establish each node's pair-wise and cluster keys.

LEAP restricts each node to three secure communication groups. SECK places no such restriction and provides a simple mechanism to establish secure communication groups of any degree within a cluster. In LEAP, every MSN has a distinct pair-wise key established with each neighbor. In SECK, however, every MSN establishes a pair-wise key only with its primary AFN.

In LEAP, each node calculates a single pair-wise key to share with each neighbor. This calculated pair-wise key is unicasted to each of the node's d neighbors. In addition, establishing a LEAP cluster key requires $n(d-1)^2/(n-1) \approx (d-1)^2$ key transmissions throughout the network [13]. LEAP does not provide a concrete message format. To compute the energy dissipation incurred by LEAP, we assume that LEAP has the same message format as that used in SECK for key distribution. Here, we only consider communication-related energy dissipation. In Figure 7.7 we compare SECK and LEAP by plotting the energy dissipated by the network for key establishment as a function of the network size (for SECK, network size implies cluster size). Note that the size of a network (i.e., number of nodes) has the biggest impact on the energy required to establish keys. We set the connection degree as 20, which was suggested by the authors of LEAP.

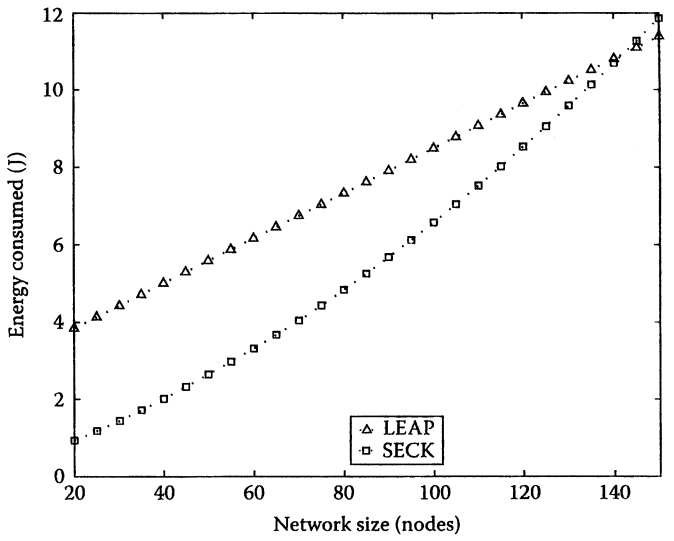

Figure 7.7 Key establishment communication overhead.

It is shown in Figure 7.7 that SECK is more efficient for small network sizes. Distributing similar amounts of keying material using the method described in SECK is better than that of LEAP when fewer nodes are considered. For the clustered network architecture described in Section 7.3, we assume that clusters consist of approximately 50 nodes — for such network sizes, SECK outperforms LEAP.

7.8.3.2 Updating Session Keys

We now switch our attention to the communication overhead incurred by SECK to maintain session keys. Carman et al. [18] show that the straightforward technique of unicasting a session key to each group member incurs the least amount of communication overhead among session key distribution schemes. SKDC is a simple instance of such a method and is most effective for a single instance of session key distribution. Our emphasis here is on the efficiency of session key distribution over multiple key update periods.

For a network deployed in a hostile environment, where node captures are expected, it is important to be able to continually distribute new session

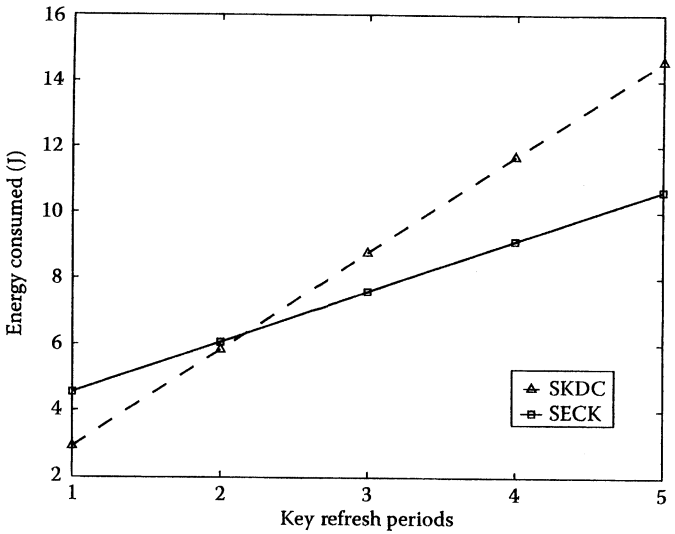

Figure 7.8 Key refresh communication overhead.

keys efficiently. SKDC creates an individual message for each group member every time a session key must be updated, while SECK requires one message for each administrative key. In Figure 7.8 we compare SECK and SKDC in terms of the communication energy dissipated by the network due to session key redistributions. The plot shows that SKDC outperforms SECK during the initial session key update periods. This is because of SECK's initial overhead for establishing the administrative keys. However, SECK outperforms SKDC as the accumulated number of session key redistributions increases. This result was expected — the administrative keys employed in SECK reduce the cumulative number of messages that must be transmitted by the AFN for session key redistributions.

7.9 Related Work

In the past few years, several technical approaches have been proposed to provide WSNs with confidentiality and authentication services via pair-wise secret (or key) sharing [3,13,19,21]. As stated earlier, Eschenauer and Gligor [3] propose a static technique for probabilistic distribution of pair-wise keys. In their scheme, keys are randomly chosen from a global key pool. Chan et al. [19] extended this idea to provide localized attack resiliency. Most WSN applications require additional levels of key sharing beyond pair-wise shared keys — specifically, group keys.

Carman et al. [18] conducted a comprehensive analysis of various group key schemes. The authors concluded that group size is the primary factor that should be considered when choosing a scheme for generating and distributing group keys in an WSN. Zhu et al. [13] proposed a comprehensive key management scheme called LEAP that establishes multiple keys for supporting neighborhood as well as global information sharing. Although LEAP includes several promising ideas, it does not adequately address scalability issues concerning the distribution and maintenance of group keys.

To address the difficult problem of scalability, many have proposed hierarchical network architectures, similar to the one described in this chapter. In [4,5,11], the authors utilize a clustered and hierarchical network architecture for key management. Jolly et al. [4] employ a hierarchical network organization to establish *gateway-to-sensor* keys. The clustering technique used by Jolly et al. was originally developed by Gupta et al. [22]. By using GPS signals, Gupta et al. propose to form a cluster in which all nodes are within one hop of the cluster head. Eltoweissy et al. [5,11] present another hierarchical key management scheme based on the Exclusion Basis System to efficiently maintain group and session key information. This approach supports key recovery only if node captures can be immediately detected.

Bohge et al. [2] propose a hierarchical authentication technique to establish and recover keys. Their approach requires a broadcast authentication scheme, and they employ a variation of μTESLA [24] for this purpose.

7.10 Conclusion

In a large-scale WSN deployed in a hostile environment, a key management scheme is needed to manage the large number of keys in the system. In this chapter we described a cluster-based dynamic key management scheme, SECK, that is designed to address this specific issue. SECK meets the stringent efficiency and security requirements of WSNs using a set of administrative keys to manage other types of keys such as session keys. SECK is a key management solution that includes (1) a location training scheme that establishes clusters and the cluster coordinate system used in the MSN recovery procedure; (2) a scheme for establishing and updating administrative keys; (3) a scheme for distributing session keys using administrative keys; (4) a scheme for recovering from multiple node captures; and (5) a scheme for reclustering and salvaging MSNs in the event that their AFN has been captured. Through analytical and simulation results, we have shown that SECK is resilient to node and key captures while incurring low levels of communication overhead.

References

1. M. Chorzempa, J. Park, and M. Eltoweissy, "SECK: Survivable and efficient clustered keying for wireless sensor networks," in *Proc. IEEE Workshop on Information Assurance in Wireless Sensor Networks,* pp. 453–458. Phoenix AZ, April 2005.
2. M. Bohge and W. Trappe, "An authentication framework for hierarchical ad hoc MSN networks," in *Proc. of the ACM Workshop on Wireless Security (WiSe),* pp. 79–87. San Diego CA, September 2003.
3. L. Eschenauer and V.D. Gligor, "A key-management scheme for distributed MSN networks," in *Proc. of the 9th ACM Conf. on Computer and Communications Security (CCS),* pp. 41–47. Washington D.C., November 2002.
4. G. Jolly, M. Kusu, and P. Kokate, "A hierarchical key management method for low-energy wireless MSN networks," in *Proc. of the 8th IEEE Symposium on Computers and Communication (ISCC),* pp. 335–340. Turkey, July 2003.
5. M. Eltoweissy, A. Wadaa, S. Olariu, and L. Wilson, "Scalable cryptographic key management in wireless sensor networks," *Journal of Ad Hoc Networks: Special issue on Data Communications and Topology Control in Ad Hoc Networks,* Vol. 3, No. 5, September 2005.
6. W. Heinzelman, "Application-Specific Protocol Architectures for Wireless Networks," Ph.D. dissertation. Massachusetts Institute of Technology, June 2000.
7. C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for MSN networks," in *Proc. 6th Conference on Mobile Computing and Networking (MobiCom),* pp. 56–67. Boston, MA, August 2000.
8. S. Madden, R. Szewczyk, M. Franklin, and D. Culler, "Supporting aggregate queries over ad-hoc wireless MSN networks," in *Proc. 4th IEEE Workshop on Mobile Computing Systems and Applications,* pp. 49–58. Callicoon, NY, June 2002.
9. A.Wadaa, S. Olariu, L. Wilson, M. Eltoweissy, and K. Jones, "Training a sensor network," *Mobile Networks and Applications,* Vol. 10, No. 1, pp. 151–168, February 2005.
10. M. Eltoweissy, H. Heydari, L. Morales, and H. Sudborough, "Combinatorial optimizations of group key management," *Journal of Networks and Systems Management,* Vol. 12, No. 1, pp. 30–50, March 2004.
11. M. Eltoweissy, M. Younis, and K. Ghumman, "Lightweight key management for wireless MSN networks," in *Proc. IEEE International Conference on Performance, Computing, and Communications,* pp. 813–818. Phoenix, AZ, April 2004.
12. M. Moharram and M. Eltoweissy, "Key management schemes in sensor networks: dynamic versus static keying," *ACM Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN 2005).* Montreal, Canada, October 2005.
13. S. Zhu, S. Setia, and S. Jajodia, "LEAP: efficient security mechanisms for large-scale distributed MSN networks," in *Proc. of the 10th ACM Conference on Computer and Communication Security (CCS),* pp. 62–72. Washington D.C., October 2003.

14. R. Moharrum, M. Mukkamala, and M. Eltoweissy, "CKDS: an efficient combinatorial key distribution scheme for wireless," in *Proc. International Conference on Performance, Computing, and Communications (IPCCC),* pp. 63–636. Phoenix, AZ, April 2004.
15. D. Carman, P. Kruus, and B. Matt, "Constraints and approaches for distributed MSN network security," Network Associates Inc. (NAI) Labs Technical Report No.00010. September 2000.
16. G. Gupta and M. Younis, "Fault-tolerant clustering of wireless MSN networks," *IEEE Wireless Communications and Networking,* Vol. 3, No. 1 pp. 1579–1584, March 2003
17. B. Matt, "A preliminary study of identity-based, group key establishment protocols for resource constrained battlefield networks," Network Associates Labs Technical Report 02-034. September 2002.
18. D. Carman, B. Matt, and G. Cirincione, "Energy-efficient and low-latency key management for MSN networks," in *Proc. of 23rd Army Science Conference.* Orlando, FL, December 2002.
19. H. Chan, A. Perrig, and D. Song, "Random key predistribution schemes for MSN networks," in *Proc. IEEE Symposium on Security and Privacy,* pp. 197–213. Oakland, CA, May 2003.
20. W. Du, J. Deng, Y.S. Han, S. Chen, and P.K. Varshney, "A key management scheme for wireless sensor networks using deployment knowledge," in *Proc. IEEE INFOCOM'04.* March 2004.
21. D. Liu and P. Ning, "Establishing pairwise keys in distributed sensor networks," in *Proc. ACM Conference on Computer and Communications Security (CCS),* pp. 52–61. Washington D.C., October 2003.
22. G. Gupta and M. Younis, "Performance evaluation of load-balanced clustering of wireless MSN networks," in *Proc. 10th International Conference on Telecommunications,* pp. 1577–1581. Papeete, French Polynesia, February 2003.
23. D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next century challenges: scalable coordination in sensor networks," in *Proc. 5th International Conference on Mobile Computing and Networks (MobiCom),* pp. 263–270. Seattle, WA, August 1999.
24. A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. Tygar, "SPINS: Security protocols for sensor networks," *Wireless Networks,* Vol. 8, No. 5, pp. 521–534, November 2002.
25. G.J. Pottie and W.J. Kaiser, "Wireless integrated network sensors," *Communications of the ACM,* Vol. 43, No. 5, pp. 51–58, May 2000.
26. Sensoria Corporation, "WINS NG Power Usage Specification: WINS NG 1.0." January 2000. Available: http://www.sensoria.com/
27. J. Chou, D. Petrovis, and K. Ramchandran, "A distributed and adaptive signal processing approach to reducing energy consumption in sensor networks," in *Proc. IEEE INFOCOM,* Vol. 2, pp. 1054–1062. San Francisco, CA, April 2003.
28. W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," in *Proc. 33rd Hawaii Int. Conference on System Sciences,* pp. 3005–3014. Maui, HI, January 2000.

29. C. Karl and D. Wagner, "Secure routing in wireless sensor networks: attacks and countermeasures," *Ad Hoc Networks Journal,* Vol. 1, No. 1, pp. 293–315, January 2003.
30. A.D. Wood and J.A. Stankovic, "Denial of service in sensor networks," *IEEE Computer,* Vol. 35, No. 10, pp. 54–62, June 2002.
31. C.K. Wong, M. Gouda, and S. Lam, "Secure group communications using key graphs," in *Proc. IEEE Transactions on Networking,* Vol. 8, No. 1, pp. 16–29, February 2000.
32. H. Harney and C. Muckhenhirn, "Group Key Management Protocol (GKMP) Specification," RFC 2093, July 1997.
33. Y.T. Hou, Y. Shi, and H.D. Sherali, "Rate allocation in wireless sensor networks with network lifetime requirement," in *Proc. ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc),* pp. 67–77. Tokyo, Japan, May 2004.
34. J. Pan, Y.T. Hou, L. Cai, Y. Shi, and S.X. Shen, "Topology control for wireless sensor networks," in *Proc. ACM International Conference on Mobile Computing and Networking (Mobicom),* pp. 286–299. San Diego, CA, September 2003.