

Key Management for Long Lived Sensor Networks in Hostile Environments

Michael Chorzempa , Jung-Min Park , and Mohamed Eltoweissy

*Bradley Department of Electrical and Computer Engineering,
Virginia Polytechnic Institute and State University,
Blacksburg, VA 24061
{mchorzem,jungmin,toweissy}@vt.edu*

Abstract

Large-scale wireless sensor networks (WSNs) are highly vulnerable to attacks because they consist of numerous resource-constrained devices 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 security services to these networks to ensure their survival. We propose a novel self-organizing key management scheme for large-scale, and *long-lived* WSNs, called *Survivable and Efficient Clustered Keying (SECK)* that provides administrative services that ensures the survivability of the network. SECK is suitable 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 provides five efficient security administration mechanisms: 1) clustering and key setup, 2) node addition, 3) key renewal, 4) recovery from multiple node captures, and 5) re-clustering. All of these mechanisms have been shown to localize the impact of attacks and considerably improve the efficiency of maintaining fresh session keys. Using simulation and analysis, we show that SECK is highly robust against node capture and key compromise while incurring low communication and storage overhead.

1 Introduction

Key management is crucial to the secure operation of Wireless Sensor Networks (WSNs). Existing key management schemes focus on the efficiency of bootstrapping session keys and, to a large extent, ignore issues regarding key reconfigurations (e.g., key updates). Hence, those schemes are more appropriate for short-lifetime WSNs rather than long-lived WSNs. In many key management schemes [6][18][27][9][30] static administrative keys, or keys that are never updated, are

adequate to manage administrative events such as membership management or re-clustering. However, in long-lived and hostile networks, the survivability of these keys cannot be assumed. We propose an efficient solution for administrative key management called *Survivable and Efficient Clustered Keying* (SECK) that directly addresses the problem of reconfiguration events and membership management. We also show how SECK can efficiently be utilized as an administrative layer for many of the existing session key management schemes proposed for WSNs.

SECK is appropriate for a long-lived network with a physically hierarchical architecture deployed in a hostile environment. We assume that such a hierarchical network's bottom tier consists of numerous clusters of sensor nodes, each cluster consisting of many low-end nodes and a more capable cluster head node. The scheme presented in this paper focuses on robust administrative key management within a cluster.

In a hostile environment, a long-lived WSN operates unattended and its nodes are highly prone to capture. Therefore, supporting reconfigurations such as node additions and revocations, re-clustering, and administrative key updates are critical to maintain the WSN's security and survivability. SECK is a self-organizing scheme that sets up key associations in the network clusters, establishes administrative 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 re-clustering of nodes.

In SECK, we make a distinction between administrative keys and session keys—the latter is sometimes called traffic encryption keys (TEKs). While session keys are provided to encrypt and/or authenticate normal data packets, administrative keys are used solely for security administration events. Previous approaches for WSN key management adequately address the session key management issue, but to a large extent, did not consider security administration events. We define a security administration event as any event that requires the network to reconfigure its node membership and/or keys due to an attack or a potential attack.

The contributions of this paper lie in the establishment of a lightweight key administration layer. SECK provides such a layer through the use of administrative keys. Existing schemes assume static (and sometimes globally shared [6]) administrative keys for security administration events. Since these keys are static, as they are used more often, they become increasingly vulnerable to attacks that take advantage of key stream reuse [1]. In SECK, we maintain dynamic administrative keys that are utilized during security administration events. The two most notable features of SECK are: (1) It provides an efficient mechanism to re-establish secure group communication after multiple node captures have been detected; and (2) It provides a re-clustering scheme that utilizes neighboring clusters to notify and absorb nodes when their cluster-head node has either been compromised or expired due to energy depletion. Existing group key maintenance schemes [10][15][16][19][37][38], can

maintain secure communication when a number of nodes have been captured, but once a critical number of nodes have been captured, secure group communication cannot be re-established.

The paper is organized as follows. In Section 2, we describe related research. In Section 3, we describe the WSN architecture that is assumed throughout the paper. We describe the fundamental design principles of SECK in Section 4, present the threat model in Section 5, and give full details of SECK in Section 6. In Section 7, we discuss SECK's robustness against node captures. In Section 8, we present a thorough energy analysis of SECK. Finally, we summarize our findings in Section 9.

2 Related Work

2.1 *Static and Dynamic Keying*

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 pre or post network deployment. Schemes can also be classified into homogeneous or heterogeneous schemes with regards 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 pre-deployment 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 paper we use the primary classification of static versus dynamic keying. While static schemes primarily assume that administrative keys will outlive the network and emphasize pair-wise communication keys, dynamic schemes advocate re-keying to achieve attack resiliency in long-lived networks and emphasizes group communication keys. Table 1 shows the primary differences between static and dynamic keying in performing key management functions. Moharram and Eltoweissy [17] provide a performance and security comparison between static and dynamic keying.

Recently, numerous static key management schemes have been proposed for sensor networks. Many of them are based upon the seminal random key pre-distribution scheme introduced by Eschenauer and Gligor [8]. In this scheme, each sensor node is assigned k keys out of a large pool P of keys in the pre-deployment 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

Table 1

Key management functions in static and dynamic keying.

(Admin. Keys assumed)	Static keying	Dynamic keying
Key assignment	Once at pre-deployment	Multiple times
Key generation	Once at pre-deployment	Multiple times
Key distribution	All keys are pre-distributed to nodes prior to deployment	Subsets of keys are re-distributed 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

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. Chan et al. [25] extended this idea to provide localized attack resiliency. Liu and Ning [27] also extend this idea and pre-distribute t -degree polynomials which are used by neighboring nodes to generate a pairwise key. By using the polynomials, this scheme maintains security up to t colluding compromised nodes. Du et al. [26] pre-deploy pairwise keys based on known deployment points. By choosing keys shared with nodes likely to be in close proximity, the probability of key sharing can be increased. Most WSN applications require additional levels of key sharing among nodes beyond just pair-wise keys—specifically, group keys.

Park and Shin [6] propose a dynamic group key management protocol called LiSP. LiSP addresses the vulnerability of key stream ciphers caused by key reuse. It addresses the problem by frequently and synchronously updating the group key. LiSP utilizes broadcast transmission to distribute the group keys and uses one-way key chains to recover from lost keys. While this scheme is very efficient, LiSP requires the use of static administration keys to perform periodic administrative functions. This leaves those keys vulnerable to disclosure.

Carman et al. [24] conducted a comprehensive analysis of various group key schemes. The authors conclude that the group size is the primarily factor that should be considered when choosing a scheme for generating and distributing group keys in a WSN. Wong et al. [37] propose a scalable group key management protocol using key graphs. They utilize keys of multiple granularity to reduce the re-keying overhead associated with membership management. They also investigate multiple approaches for constructing re-keying messages. While efficient, this approach requires a centralized key distribution center. Zhu et al. [18] propose a comprehensive dynamic 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 secure group key distribution.

2.2 Hierarchical Network Architectures

To address the difficult problem of scalability, many have proposed hierarchical network architectures. In [9][10][16], authors utilize a clustered and hierarchical network architecture for key management. Jolly et al. [9] 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. [28]. 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. [10][16] 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.

The issue of coordination and aggregation has prompted the need for more capable aggregation techniques. Heinzelman et al. [11][34] propose LEACH, a cluster-based routing and aggregation scheme that periodically elects fresh aggregation nodes to combine sensed data from groups of sensors by eliminating redundancies. This approach was shown to increase system lifetime by an order of magnitude compared to general purpose approaches. This approach necessarily places a higher computational burden on the elected aggregation sensors. Madden et al. [13] present a scheme that makes use of overheard downstream messages to reduce redundancy in sensed data on the fly. They show that aggregation methods can be made efficient for low-end sensors to carry out. Estrin et al. [29] address the need for scalable coordination in large-scale sensor networks. This early paper addresses the need for application specific and localized algorithms. They propose using clusters to propagate relevant sensed data using an approach called directed diffusion.

To address the problem of clustering sensors in a hierarchical network architecture, Wadaa et al. [14] require directional broadcasts of variable power to partition the neighbors of a cluster head into wedges and distances in order to assign a coordinate identifier. This scheme supports nodes that contain no unique identifier prior to deployment, but does not guarantee a unique identifier for each node. Kuhn et al. [21] address initialization issues in ad hoc and sensor networks. Particularly, they propose a clustering scheme that requires no synchronization on the part of the clustering nodes. Gupta and Younis [22] propose a fault-tolerant clustering scheme that clusters sensors around a more capable cluster head based on the perceived ranges from the sensors to the cluster head. Re-clustering is then performed through the use of maintained backup information. While simple, this scheme does not

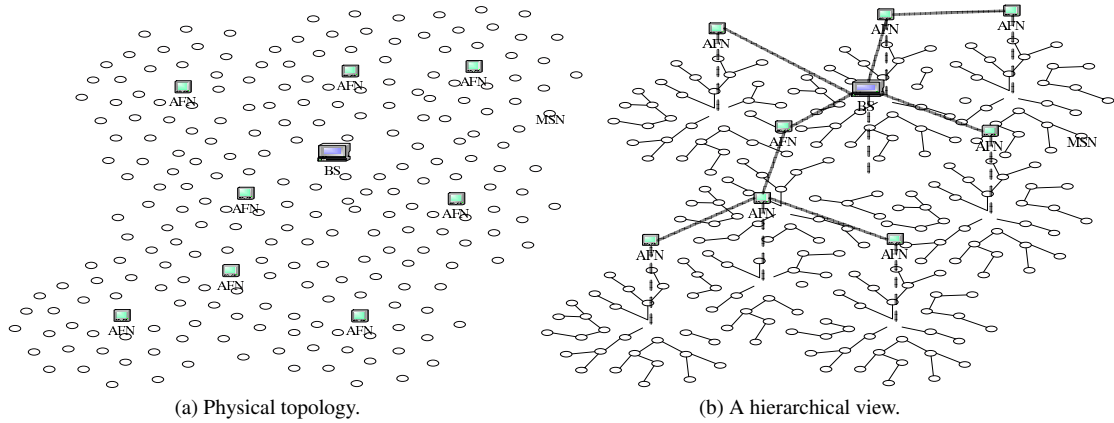


Fig. 1. A two-tiered wireless sensor network.

address authentication issues.

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 [11][33][34][39][40]. Recent data aggregation techniques [13], 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, since, in a flat network, these nodes receive significantly higher traffic volume than remote nodes [35][36]. 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 1(a) and 1(b) show the *physical* and *hierarchical* network topologies 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. A 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 clustered around an administrative AFN. Within each cluster, a set of keys needs to be deployed and managed to maintain secure communications between the MSNs and the AFN. SECK was designed for this purpose.

Since SECK is capable of managing events that require the network to reconfigure its membership and/or keys due to a network intrusion, an intrusion detection system (IDS) must also be part of the network architecture. SECK assumes the existence of an IDS [2][3][4] to monitor for anomalies in the network. Specifically, we assume the following: (1) Each AFN is equipped with a router-based IDS module that detects anomalous behavior among MSNs; and (2) AFNs and the BS are equipped with a cooperative IDS module, similar to the model proposed in [2], for detecting anomalous behavior among AFNs. These IDS modules are organized into a hierarchy where the BS ultimately receives all pertinent detection data, and is responsible for initiating reconfigurations. We do not discuss the technical challenges in implementing such an IDS as it is out of the scope of this paper.

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 the administrative keys within a cluster.

SECK assigns a set of administrative keys to each node in a network to assist the session key management scheme. To manage administrative keys within a cluster, SECK employs the *Exclusion Basis System* (EBS) [15]. EBS 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, and m is the number of keys from the global key set not held within a subset. We represent 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 ${}_{k+m}C_k$ unique subsets of k key assignments. For full details of the EBS, see [10][15]. The notation N_i denotes the i -th node, and S_i denotes its assigned administrative key subset, with $|S_i| = k$. The notation T_i denotes the set of 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 2. 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.

Table 2
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

The administrative keys effectively serve as key encryption keys. Popular session key management schemes utilize a static administrative key when the session key needs to be updated, revoked, or re-configured. Our administrative keys would replace those static keys to provide a higher level of security as the increased lifetime of the network requires fresh administrative keys. Another 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 sub-groups of 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. However, to initially distribute a cluster-wide session 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 to any number of members less than or equal to n , the basic scheme executes a similar procedure using a maximum of $k + m$ keys. A session key, Kg , would be updated using the following procedure:

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

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

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

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

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

where \Rightarrow represents broadcast transmission, $\mathbf{E}_K(M)$ represents encrypting message M with key K , and \parallel represents bit-wise concatenation. These broadcast messages ensure that only previous holders of Kg will be able to successfully receive the new 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 ensuring forward secrecy. To do 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 ($m = 2$).

$$AFN \Rightarrow N_1, \dots, N_{10} : \\ ID_{AFN} \parallel \mathbf{E}_{Ka_4}(\mathbf{E}_{Ka_1}(Ka'_1) \parallel \mathbf{E}_{Ka_2}(Ka'_2) \parallel \mathbf{E}_{Ka_3}(Ka'_3)),$$

$$AFN \Rightarrow N_1, \dots, N_{10} : \\ ID_{AFN} \parallel \mathbf{E}_{Ka_5}(\mathbf{E}_{Ka_1}(Ka'_1) \parallel \mathbf{E}_{Ka_2}(Ka'_2) \parallel \mathbf{E}_{Ka_3}(Ka'_3)),$$

From Table 2, 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 need to be considered: (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 2, 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. In 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 6, we describe the full-fledged 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 relevant to SECK.

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 paper, when we state that a node has been compromised or captured, we assume that all key information held by that node is known 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 single node capture [9][18]. 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. 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 need to 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 “re-clustered” or absorbed into other existing clusters in their vicinity. If re-clustering 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. In order 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.

The next threat is the compromise of MSNs within the same cluster. In the basic scheme described in Section 4, it is possible for an adversary to capture not only some but all 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 4, the strategic choice would be to compromise N_1 and N_6 , which would reveal all five administrative keys. Therefore, if capture detection 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 7.1. 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 a 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 of others.

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 initially stored by each MSN to support SECK. We then describe a location training scheme that sets up the network clusters, and the system used to establish administrative keys. Then, we describe techniques for replacing administrative keys that have been compromised by multiple MSN captures. Next, we describe a re-clustering scheme for salvaging MSNs within a cluster after the AFN of that cluster has been captured. Finally, we describe a method to dynamically add a MSN to the network. Below, in Figure 2, we show the overall operation and components that complete SECK. Events are represented in italics and SECK components are represented in non-italics.

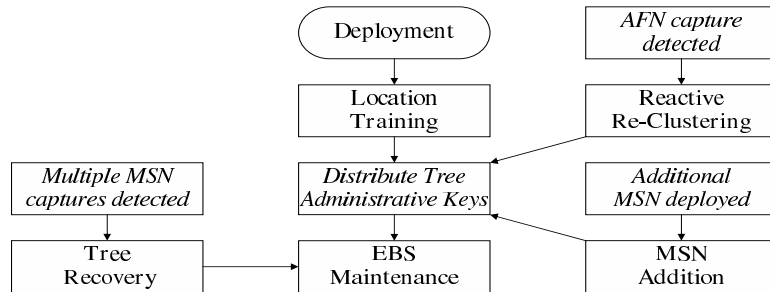


Fig. 2. Components of SECK.

6.1 Required Keys

Initially each MSN will be deployed with the complete administrative key set, $\{Ka_1, Ka_2, \dots, Ka_{k+m}\}$. After a short set up period, only a subset of these keys will be stored. In addition, each MSN will be required to store 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 re-clustering 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.

6.2 Location Training

The location training scheme will establish a cluster of MSNs around a primary AFN and assign each MSN a *cluster coordinate identifier* for this primary cluster. This identifier is needed for establishing each MSN’s administrative keys, which are used for session key establishment and routing. In addition, location training will generate inter-cluster routes between neighboring AFNs that will be collected by the BS. These routes will allow the BS to identify the most efficient backup cluster for each MSN when re-clustering becomes necessary. SECK uses coordinate establishment messages (*CEMs*) to establish cluster coordinate identifiers and the inter-cluster routes.

A cluster coordinate is defined as the tuple $(tree, hopcount)$, where *tree* is an integer assigned by the primary AFN, and *hopcount* is the MSN’s distance from its primary AFN in communication hops. We define a *tree* as a set of MSNs that route packets through the same tree root when forwarding data to its primary AFN, where the tree root is a MSN that is one hop away from its primary AFN.

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 neighbor MSNs that it has discovered to all other MSNs within 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 6.3). This broadcast transmission can be expressed as

$$AFN_x \Rightarrow \text{MSNs in transmission range :} \\ ID_{AFN_x} \parallel (ID_1, tree_1) \parallel (ID_2, tree_2) \parallel \dots \parallel (ID_{|S_{h1}|}, tree_{|S_{h1}|}),$$

where S_{h1} denotes the set of MSNs within one-hop of AFN_x . 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, say u , finds its ID on this list, it assigns its cluster coordinate as $(tree_{ID_u}, 1)$ and becomes one of the tree roots of AFN_x . If a MSN does not find its ID, but finds the ID of a neighbor, say v , it assigns itself the cluster coordinate $(tree_{ID_v}, 2)$ —the first entry corresponds to the tree number of the neighbor MSN, and the second entry indicates the hop count from AFN_x . MSNs with multiple neighbors on the neighbor list of AFN_x should randomly choose which tree to join.

The group of second hop neighbors, S_{h2} , initiates the forwarding of CEM s. These messages will be processed and forwarded by MSNs further away from AFN_x . This limited flood will terminate when the messages are received by an AFN that is a neighbor of AFN_x . When a CEM reaches its final destination (i.e., the neighbor AFN), it will contain an ordered list of intermediate MSNs that identify a route from AFN_x to the neighbor AFN. This process of acquiring inter-cluster routes is similar to the route discovery process in the ad hoc routing protocol, DSR [5].

A CEM contains the ID of the AFN whose cluster this CEM originated from, ID_{AFN_x} , the cluster coordinate of the last MSN to handle this CEM , and the ordered list of IDs of MSNs that have handled this CEM , which we will denote as $route_{AFN_x}$. The following notation describes the CEM forwarding process:

$$\text{For all } N_i \in S_{h2}, N_i \Rightarrow \text{neighbors} : \\ ID_{AFN_x} \parallel (tree, hopcount)_{N_i} \parallel route_{AFN_x}.$$

Upon hearing this CEM , MSNs not yet holding a cluster coordinate will assign its primary AFN to be ID_{AFN_x} , assign its cluster coordinate as $(tree_{N_i}, hopcount_{N_i} + 1)$, append its ID to $route_{AFN_x}$, and re-broadcast this CEM . As these messages propagate, all MSNs will establish a cluster coordinate with one primary AFN.

Upon hearing subsequent CEM s as $(ID_{AFN_y} \parallel (tree, hopcount)_{N_j} \parallel route_{AFN_y})$, an MSN, N_i , will face one of three situations. First, if $hopcount_{N_j} + 1 < hopcount_{N_i}$, N_i will reassign its primary AFN and cluster coordinate, append its MSN ID to $route_{AFN_y}$, and re-broadcast this CEM . Second, if $hopcount_{N_j} \geq hopcount_{N_i}$ and $ID_{AFN_y} = ID_{AFN_x}$ the MSN will discard this message, as it is from an MSN within N_i 's primary cluster and does not improve the hop count to the primary AFN. Third, if $hopcount_{N_j} \geq hopcount_{N_i}$ and $ID_{AFN_y} \neq ID_{AFN_x}$, N_i will append its ID to $route_{AFN_y}$ and unicast this CEM toward its already know primary AFN, AFN_x . This third scenario will enable AFNs to collect routes to MSNs within neighboring clusters.

The final step in the location training procedure is for each AFN to forward all of the received CEM s to the BS. Using every $route_{AFN}$ received from all AFNs, the BS can generate a primary and backup cluster membership list for each MSN in the network. We will refer to these lists as the *membership lists*. A simple method for generating the *membership lists* is to store a route from an MSN to the AFN with the shortest number of hops (primary cluster) and the AFN with the second-shortest number of hops (backup cluster) for every MSN in the network. This could also be done at the AFNs to limit the number of coordinate establishment messages forwarded to the BS. While our scheme is designed for clusters of any number of hops, location training is more efficient, in terms of communication overhead, when the maximum number of hops within a cluster is small. Therefore, the efficiency of this scheme improves when the AFNs are densely deployed. Figure 3 shows a partition of a network around AFN_x 's primary cluster. This figure demonstrates

the cluster coordinates established and the information used by the BS to generate N_2 's entries in the *membership lists*. The BS would receive five *CEMs*, each containing a route that includes N_2 . The BS would establish AFN_x as N_2 's primary AFN and AFN_A as the backup AFN with (N_{13}, N_{12}) as the route to the backup AFN in the *membership lists*.

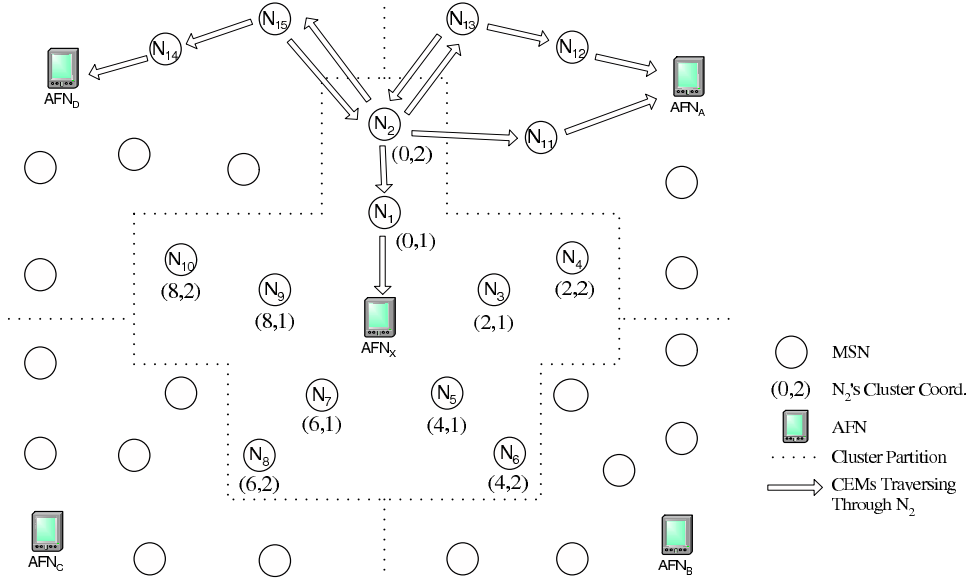


Fig. 3. A network partition illustrating location training.

6.3 Establishment of Administrative Keys

The administrative key subset is determined by the assigned AFN cluster coordinate. We propose an algorithm for assigning tree numbers that uniformly distribute assigned key subsets based on the expected number of MSNs within a cluster. We assume that the expected number of MSNs can be estimated based on the density of AFN deployment relative to MSN deployment. We will denote the number of neighbors found by an AFN—referred to as the *degree* of an AFN—as d ($d = |S_{h1}|$) and the estimated number of MSNs within a cluster as n . We assign tree numbers as $tree_i = i \cdot \lfloor d/n \rfloor$ 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. Since a cluster coordinate consists of $(tree, hopcount)$, a 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 $[1, n]$.

Each AFN is also responsible for generating and distributing the tree administrative keys for each tree in its cluster. This requires n 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 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:

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

where \rightarrow represents unicast transmission, and $S_i = \{Ka_1, Ka_2, \dots, Ka_{|S_i|}\}$. Recall that S_i is N_i 's administrative key subset. The encryptions are nested to ensure only N_i successfully receives Kt_j on this transmission.

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 are maintained globally, the compromise of a single cluster's keys would compromise the entire network's keys. We described in Section 5 that in order to isolate an attack, the administrative keys must be independently generated in each cluster and not shared among clusters.

6.4 Administrative Key Recovery

SECK provides a mechanism to recover compromised administrative keys from multiple MSN captures within an isolated set of trees. The MSNs in the remaining trees can be salvaged by reestablishing their administrative key subsets using their tree administrative keys.

In Section 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 is notified by the BS 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.

$$\text{For all } tree_i \in S_{ut}, \text{AFN} \Rightarrow tree_i : \\ \mathbf{E}_{Kt_i}(\mathbf{E}_{Ka_1}(Ka'_1) \parallel \mathbf{E}_{Ka_2}(Ka'_2) \parallel \dots \parallel \mathbf{E}_{Ka_{k+m}}(Ka'_{k+m})).$$

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 7.3, we will 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).

6.5 Reactive Re-clustering after AFN Capture

Unlike a 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 [20]. To keep the MSNs operational after the type of AFN capture described in Section 5, we need to re-cluster the MSNs into backup clusters and establish new security relationships between the re-clustered MSNs and their respective backup AFNs (AFN_b). We assume that the event of a successful AFN capture is infrequent enough as to warrant a reactive approach so as not to require MSNs to maintain security associations with backup clusters, similar to common re-clustering approaches such as [22]. We must also assume that the backup AFNs have overlapping coverage, that is, are necessarily able to directly transmit to all MSNs joining its cluster. If not, re-clustering would not be feasible.

In SECK's approach, the BS acts as a trusted third party (TTP). Once notified of an AFN capture by the IDS, the BS consults its *membership lists* to first identify the MSNs belonging to the captured AFN's (AFN_c) cluster, and second to identify the necessary backup AFNs. The BS must then notify these backup AFNs as to which MSNs will need to be absorbed into their cluster, these backup AFN's then absorb the MSNs by issuing them updated administrative keys and routing information using a shared secret generated by the TTP.

Using the *membership lists* that identify the MSNs within AFN_c 's cluster, the BS generates tickets for each of these MSNs using that MSN's pair-wise key, K_{pi} , as follows:

$$Ticket_{N_i} = \mathbf{E}_{K_{pi}}(K_{AFN_b-N_i} || ID_{AFN_b} || ID_{N_i} || route_{N_i-AFN_b} || nonce),$$

where N_i is an MSN from AFN_c 's cluster. Note that ID_{AFN_b} and $route_{N_i-AFN_b}$ is obtained from the *membership lists* generated during the location training procedure described in Section 6.2. The key, $K_{AFN_b-N_i}$, must be generated for each N_i , to be used by AFN_b for securely distributing updated keys to N_i . The *nonce* is attached to prevent replay attacks.

Once all of the tickets for the nodes in AFN_c 's cluster have been generated, they are distributed to the corresponding backup AFNs as follows:

$$\text{BS} \rightarrow AFN_b : \quad \mathbf{E}_{K_{AFN_b}}(K_{AFN_b-N_i} || ID_{N_i} || Ticket_{N_i},$$

where AFN_b is N_i 's backup AFN, and K_{AFN_b} is a pair-wise key shared between AFN_b and the BS.

When AFN_b receives this message, it assigns an unused key subset identifier to N_i and generates an absorption message as follows;

$$Absorb_{N_i} = \mathbf{E}_{K_{AFN_b-N_i}}(S_i || subset_identifier_{N_i}),$$

where S_i is N_i 's newly assigned set of administrative keys and $subset_identifier_{N_i}$ identifies which keys from AFN_b 's administrative key set are included in S_i . This message will securely establish N_i 's membership within AFN_b 's cluster.

N_i is first notified of AFN_c 's capture when it overhears the following message from AFN_b .

$$AFN_b \rightarrow \text{neighbors} : \quad ID_{N_i} || ID_{AFN_b} || Absorb_{N_i} || Ticket_{N_i}.$$

Upon hearing this message from AFN_b , N_i first authenticates this message using the BS pair-wise key and $Ticket_{N_i}$. Next, N_i re-assigns its administrative key subset to the one specified in $Absorb_{N_i}$. Finally, N_i sends an acknowledgement message to AFN_b using the route specified in $Ticket_{N_i}$, signalling the successful absorption into AFN_b 's cluster. This message can be expressed using the following notation, assuming that the next hop node in $route_{N_i-AFN_b}$ is N_x and $S_i = \{Ka_1, Ka_2, Ka_3\}$:

$$N_i \rightarrow N_x : \quad ID_{N_i} || \mathbf{E}_{Ka_1}(ID_{N_i}) || (\mathbf{E}_{Ka_2}(ID_{N_i})) || (\mathbf{E}_{Ka_3}(ID_{N_i})).$$

Note that if an EBS reaches its maximum number of nodes, i.e., $n =_{k+m} C_k$, 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 paper; we simply assume that the size of a cluster's initial EBS will be sufficient for node additions. We refer the reader to [15] for an EBS expansion mechanism.

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 additional MSNs are randomly deployed, and the MSN's resulting cluster is unknown. In SECK, the keys maintained by an MSN are generated post-deployment. For this reason it is impossible for a new MSN to be pre-deployed with any keys that will enable authentication with a MSN or AFN directly. However, each new MSN will

contain a unique pairwise key, K_{p_i} , shared with the BS. This key allows the BS to act as a TTP between the new MSN and the existing AFN.

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

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

where ID_{AFN_p} is $neighbor_i$'s primary AFN, and $hopcount_{N_i}$ is the hopcount in $neighbor_i$'s cluster coordinate identifier. Node 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. Node N_{new} determines the most appropriate AFN to join, AFN_x , using the AFN with the maximum number of neighbors. Node N_{new} selects the most efficient neighbor, N_x , based on the neighbor's $hopcount$ to AFN_x . The neighbor with the smallest $hopcount$ is the most efficient node to use as a next hop node in the direction of AFN_x . Now that the next hop node to AFN_x is known, N_{new} constructs a short authentication request message destined for the BS. Node N_{new} sends the following message to N_x :

$$N_{new} \rightarrow N_x : \quad (ID_{N_{new}} \parallel ID_{AFN_x} \parallel nonce) \parallel MAC_{K_{p_i}},$$

where $MAC_{K_{p_i}}$ represents a message authentication code (MAC) of this message generated with N_{new} 's base station pairwise key, K_{p_i} . Node N_x then routes this message to AFN_x in the same manner as forwarding sensed data to AFN_x for aggregation. After receiving this message, AFN_x attaches its own MAC using its pair-wise key shared with the BS, K_{AFN_x} .

$$AFN_x \rightarrow BS : \quad (ID_{N_{new}} \parallel ID_{AFN_x} \parallel nonce) \parallel MAC_{K_{p_i}} \parallel MAC_{K_{AFN_x}}.$$

When the BS receives this message, it authenticates N_{new} and AFN_x using the MACs. If both nodes are authenticated, the BS generates the following ticket for N_{new} :

$$Ticket_{N_{new}} = \mathbf{E}_{K_{p_i}}(K_{AFN_x-N_{new}} \parallel ID_{AFN_x} \parallel ID_{N_{new}} \parallel nonce).$$

Node N_{new} is then able to establish a security association with AFN_x in a manner that is identical to how a node is re-clustered as described in Section 6.5.

7 Robustness of SECK Against Node Captures

Node captures in hostile environments is inevitable. 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 5, we analyze how well SECK recovers from those attacks.

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 in order to completely compromise the network. This is not the case for most non-hierarchical key management schemes. In non-hierarchical key management solutions, a localized attack often has a global impact on the network's keys [8][23][25][27].

7.2 Robustness against MSN Node Captures

In the example given by Table 2, 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 instance, 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 are revealed by those captures.

In Figure 4, the expected ratio of keys captured is plotted as a function of the number of nodes captured 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. 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 tradeoff. Further discussions on this issue are continued

in the next section.

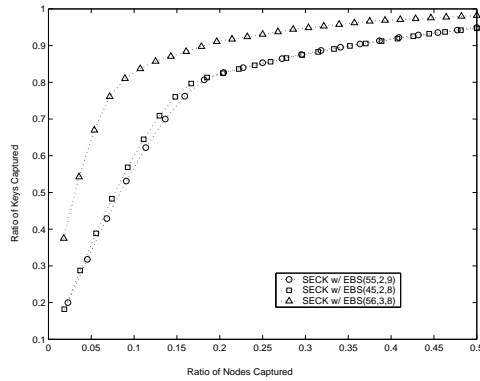


Fig. 4. Expected ratio of keys captured vs. ratio of nodes captured.

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 4 shows that SECK, using EBS(55,2,9), provides the most resiliency against node captures. 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 is 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 $(n - h \cdot x)/(n - x)$. 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 $(n - h \cdot \lceil h/x \rceil)/(n - x) \approx 1$. From the above analyses, we can observe that SECK’s recovery procedure performs ideally in localized attacks.

In Figure 5, 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 fourteen nodes have been captured. In practice, the actual recovery ratio is expected to be somewhere between 0.6 and 1.0 when

fourteen nodes are captured. As expected, as the ratio of nodes captured increases, the ratio of recoverable nodes drops.

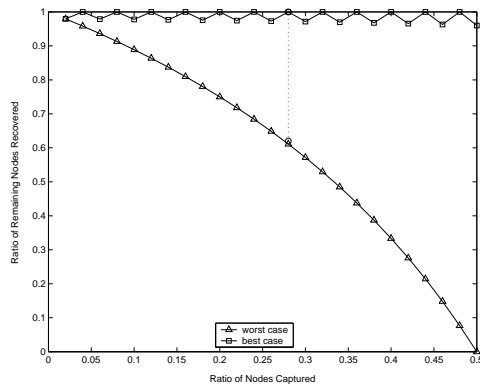


Fig. 5. Administrative key recovery procedure evaluation.

7.4 Evaluation of the Location Training Procedure

During the location training procedure discussed in Section 6.2, it is possible for multiple MSNs to establish the same cluster coordinate. This situation occurs if multiple MSNs within the same tree have the same hop count from the same primary AFN. Each of these MSNs have identical cluster coordinates, and thus holds identical administrative and session keys at any point in time. It follows that the set of MSNs sharing a cluster coordinate collectively constitute a single group member in SECK. It also follows that this set of MSNs is the granularity of eviction. Simulation results indicate that SECK’s location training scheme keeps the number of nodes that share a cluster coordinate low.

Our simulation environment consisted of 25, 50, and 75 MSNs, randomly distributed around a centrally-located AFN in a uniform manner. We also utilized a free space propagation model for each MSN with a transmission radius equal to $1/4$ of the grid diameter, e.g., 20x20m grid with 5m sensor transmission radius. Figure 6 illustrates the distribution of cluster coordinates averaged over 100 simulations.

In Figure 6, almost 70% of the cluster coordinates are associated with a single MSN, while less than 20% of the cluster coordinates are shared by two MSNs. It can also be seen that the density of MSN deployment has little impact on the likelihood of nodes sharing cluster coordinates. In contrast, the number of an AFN’s 1-hop neighbors does influence the likelihood of nodes sharing cluster coordinates—Figure 7 illustrates this relationship. The results shown in the figure can be explained by considering the relationship between the 1-hop neighbors and the number of trees. If more 1-hop neighbors are found by an AFN, then more trees are established. This, in turn, increases the number of unique cluster coordinates that are possible.

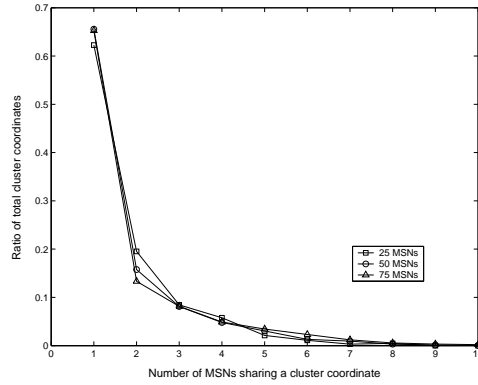


Fig. 6. Density effect on cluster coordinate sharing.

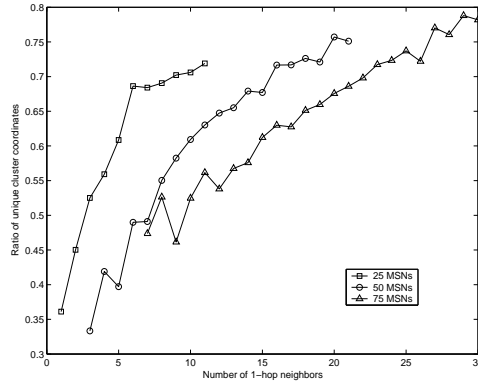


Fig. 7. 1-hop neighbor effect on cluster coordinate sharing.

8 Evaluation of Communication and Storage Overhead

8.1 Energy Dissipation

In key management schemes, the energy required for computation is three orders of magnitude less than that required for communication [20]. Moreover, 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 0.021mJ/bit for transmission, and 0.014mJ/bit for reception when operating at 10kbps [32]. We make the same assumptions as in [20] 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 3 shows the amount of energy dissipated by a single node to complete one instance of each process (five 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 $|S_{h_1}| = 20$. In the following, we discuss the major factors that contribute to the energy consumption for each of these processes.

Table 3
SECK communication energy consumption.

		Transmission (mJ)	Reception (mJ)	Total (mJ)
Location Training	AFN	55.10	268.80	323.90
	MSN	7.24	43.91	51.15
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 Re-clustering	AFN	322.56	225.79	578.35
	MSN	9.41	11.20	20.61
MSN Addition	AFN	9.41	4.48	13.89
	MSN	16.13	29.12	45.25

The energy required for the location training process is affected by the number of CEMs handled by each MSN. We have assumed that each MSN handles one CEM to establish its primary AFN and one CEM to establish a route to its backup AFN. Location training becomes more costly if many more CEMs are handled by each MSN. 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 is dependent on the dimension of the EBS that is used. The energy cost of the tree recovery process shown in Table 3 is the energy dissipated by the AFN for each *tree* recovered. The energy cost of reactive re-clustering calculated in Table 3 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 attempting to recover with a specific backup

AFN. Finally, the energy cost of node addition calculated is the energy needed to respond to, then to forward the recovery messages of one newly added MSN. It can be seen from Table 3 that SECK effectively offloads much of the energy-intensive operations to the more capable AFN.

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 2 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.

8.3 *Comparison of Communication Overhead*

The keying communication overhead of SECK is incurred during (1) the initial key establishment phase and (2) during periodic key maintenance procedures. We compare the communication overhead incurred in (1) and (2) with those incurred in Localized Encryption and Authentication Protocol (LEAP) [18] and Simple Key Distribution Center (SKDC) [20], 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 [24] that this basic method used by SKDC is the most efficient means to distribute a session key.

8.3.1 Key Establishment

In this section, 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 pairwise 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 pairwise key established with each neighbor. In SECK, however, every MSN establishes a pairwise key only with its primary AFN.

In LEAP, each node calculates a single pairwise key to share with each neighbor. This calculated pairwise key is unicasted to each of this 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 [18]. 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 8, 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.

It is shown in Figure 8 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 3, we assume that clusters consist of approximately 50 nodes—for such network sizes SECK outperforms LEAP.

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. [24] 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.

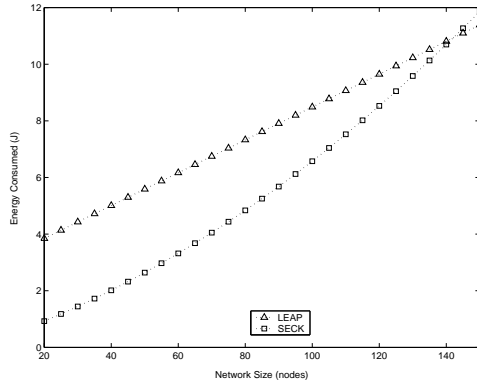


Fig. 8. Key establishment communication overhead.

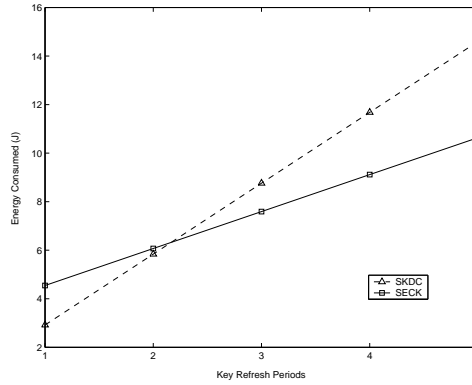


Fig. 9. Key refresh communication overhead.

For a network deployed in a hostile environment, where node captures are expected, it is important to be able to continually distribute new session keys efficiently. SKDC creates an individual message for each group member every time that a session key must be updated, while SECK requires one message for each administrative key. In Figure 9, 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 need to be transmitted by the AFN for session key redistributions.

9 Conclusion

In a large-scale WSN with clustered formations of sensor nodes, a key management scheme is needed to manage the large number of keys in the system. In this paper, we have describe a cluster-based dynamic key management scheme, SECK,

that was designed to address this specific issue. SECK meets the stringent efficiency and security requirements of WSNs by using a set of administrative keys to manage other types of keys such as session keys. SECK is a comprehensive 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 re-clustering 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 requiring a low level of communication overhead.

References

- [1] N. Borisov, I. Goldberg, D. Wagner, "Intercepting mobile communications: the insecurity of 802.11," in *Proc. IEEE/ACM MobiCom*, pp. 180–189. Italy, July 2001.
- [2] Y. Zhang and W. Lee, "Intrusion detection in wireless ad-hoc networks," in *Proc. of IEEE/ACM MobiCom*, pp. 275–283. Boston MA, August 2000.
- [3] R. Zhang, D. Qian, C. Bao, W. Wu, X. Guo, "Multi-agent based intrusion detection architecture," in *Proc. International Conference on Computer Networks and Mobile Computing*, pp. 494–501. Beijing China, October 2001.
- [4] S. Ganeriwal and M. Srivastava, "Reputation-based framework for high integrity sensor networks," in *Proc 2nd ACM Workshop on Security of Ad Hoc and Sensor Networks*, pp. 66–77. Washington DC, October 2004.
- [5] D. Johnson, D. Maltz, and Y. Hu, "The dynamic source routing protocol for mobile ad hoc networks (DSR)," (Internet-Draft). Mobile Ad-hoc Network (MANET) Working Group, IETF, July 2004.
- [6] T. Park and K. Shin, "LiSP: A lightweight security protocol for wireless sensor networks," in *ACM Transactions on Embedded Computing Systems* Vol. 3, No. 3, pp. 634-660, August 2004.
- [7] 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.
- [8] 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 DC, November 2002.
- [9] 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.

- [10] M. Eltoweissy, A. Wadaa, S. Olariu, and L. Wilson “Scalable cryptographic key management in wireless sensor networks,” in *Proc. Distributed Computing Systems Workshops*, Tokyo Japan, March 2004.
- [11] W. Heinzelman, “Application-specific protocol architectures for wireless networks,” *Ph.D. Dissertation*, Massachusetts Institute of Technology, June 2000.
- [12] 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, Aug. 2000.
- [13] 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.
- [14] 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. Feb. 2005.
- [15] 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.
- [16] 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.
- [17] 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.
- [18] S. Zhu, S. Setia, 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 DC, October 2003.
- [19] 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*, pp. 630-636. Phoenix AZ, April 2004.
- [20] 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.
- [21] F. Kuhn, T. Moscibroda, and R. Wattenhoffer. “Initializing newly deployed ad hoc and sensor networks,” in *Proc. International Conference on Mobile Computing and Networking (MobiCom04)*, pp. 260–274. Philadelphia PA, September 2004.
- [22] 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
- [23] 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.

- [24] 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.
- [25] 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.
- [26] 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.
- [27] 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 DC, October 2003.
- [28] 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.
- [29] 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.
- [30] 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.
- [31] G. J. Pottie and W. J. Kaiser, "Wireless integrated network sensors," *Communications of the ACM*, vol. 43, no 5, pp. 51-58. May 2000.
- [32] Sensoria Corporation, "WINS NG Power Usage Specification: WINS NG 1.0," January 2000. Available: <http://www.sensoria.com/>
- [33] 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.
- [34] 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.
- [35] 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.
- [36] A. D. Wood and J. A. Stankovic, "Denial of service in sensor networks," *IEEE Computer* vol. 35, no. 10, pp. 54-62, June 2002.
- [37] 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.
- [38] H Harney and C Muckenhirn, "Group key management protocol (GKMP) Specification," RFC 2093, July 1997.

- [39] 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.
- [40] 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.

Appendix: Pseudocode

MSN Location Training Procedure

Notation

AFN_i = AFN in range of MSN

AFN_p = Primary AFN

$neighbor_i$ = neighboring MSN in range of this MSN

$AFN_ineighborlist(ID)$ = tree identifier of MSN with id= ID

$CEM_y = (AFN_y \parallel tree_y \parallel hopcount_y \parallel route_{AFN_y})$

$timeout$ = appropriate time to allow all necessary coordinate messages to propagate

$forward_{to_r}(m)$ = forward message m toward node r

1. $hopcount = tree = \infty$;
2. $foreach(AFN_i)\{$
3. $recv(AFN_ineighborlist)$;
4. $if(my_id \in AFN_ineighborlist)\{ //1 hop neighbor of $AFN_i$$
5. $AFN_p = AFN_i$
6. $tree = AFN_ineighborlist(my_id)$;
7. $hopcount = 1$;
8. $\}else\{$
9. $foreach(neighbor_i)\{$
10. $if(neighbor_i \in AFN_ineighborlist)\{ //2 hop neighbor of $AFN_i$$
11. $AFN_p = AFN_i$
12. $tree = AFN_ineighborlist(neighbor_i)$;
13. $hopcount = 2$;
14. $\}$
15. $\}$
16. $\}$
17. $\}$
18. $if(hopcount = 2) //send CEM$
19. $broadcast(AFN_p \parallel tree \parallel hopcount \parallel route_{AFN_p})$;
20. $while(!timeout) //receive overheard CEM$
21. $recv(CEM_y)$;
22. $if(hopcount > hopcount_y + 1) //this should be primary AFN$
23. $AFN_p = AFN_y$
24. $tree = tree_y$

```

25.   hopcount = hopcounty + 1;
26.   broadcast( $AFN_p \parallel tree \parallel hopcount \parallel route_{AFN_p} \parallel my\_id$ );
27. }else if((hopcounty ≥ hopcount)&&(AFNnew! = AFNp)){
28.   forwardto $_{AFN_p}(CEM_y \parallel my\_id)$ ;
29. }else //not a useful AFN
30.   drop message;
31. }
End

```

Reactive Re-clustering Procedure

Notation

AFN_c = captured AFN

$Ticket_{N_i} = E_{K_{p_i}}(K_{AFN_b-N_i} \parallel ID_{AFN_b} \parallel ID_{N_i} \parallel route_{N_i-AFN_b} \parallel nonce)$

$Absorb_{N_i} = E_{K_{AFN_b-N_i}}(S_i \parallel subset_identifier_{N_i})$

$ACK_{N_i} = ID_{N_i} \parallel E_{K_{a_1}}(ID_{N_i}) \parallel (E_{K_{a_2}}(ID_{N_i})) \parallel (E_{K_{a_3}}(ID_{N_i}))$ where $K_{a_1}, K_{a_2}, K_{a_3} \in S_i$

$verify_k(m)$ = verify message m using key k

$send_{to_r}(m)$ = send message m to node r

$forward_{to_r}(m)$ = forward message m toward node r

MSN Procedure

```

1. recv( $ID_{N_i} \parallel ID_{AFN_b} \parallel Absorb_{N_i} \parallel Ticket_{N_i}$ );
2. if( $ID_{N_i} == my\_id$ ){
3.   verify $_{K_{p_i}}(Ticket_{N_i})$ ;
4.   assign administrative key set to  $S_i$ ;
5.   forwardto $_{AFN_b}(ACK_{N_i})$ ;
6. }

```

End

AFN Procedure

```

1. forall( $N_i$ ){ //Receive Tickets from BS
2.   recv( $E_{K_{AFN_b}}(K_{AFN_b-N_i} \parallel ID_{N_i}) \parallel Ticket_{N_i}$ );
3.   generate  $Absorb_{N_i}$ ;
4.   sendto $_{N_i}(ID_{N_i} \parallel ID_{AFN_b} \parallel Absorb_{N_i} \parallel Ticket_{N_i})$ ;
5. }
6. while( $\exists N_i$  yet to respond with  $ACK_{N_i}$ ){
7.   recv( $ACK_{N_i}$ );
8.   forall( $K_j \in S_i$ ){
9.     verify $_{K_j}(ACK_{N_i})$ ;
10.  }
11. }

```

End

Base Station Procedure

1. *forall*($N_i \in AFN_c$'s cluster){
2. generate $Ticket_{N_i}$;
3. *send_to* $_{AFN_b}(E_{K_{AFN_b}}(K_{AFN_b-N_i} || ID_{N_i}) || Ticket_{N_i})$;
4. }

End