Location-Aware Combinatorial Key Management Scheme for Clustered Sensor Networks

Mohamed F. Younis, *Senior Member*, *IEEE*, Kajaldeep Ghumman, and Mohamed Eltoweissy, *Senior Member*, *IEEE*

Abstract—Recent advances in wireless sensor networks (WSNs) are fueling the interest in their application in a wide variety of sensitive settings such as battlefield surveillance, border control, and infrastructure protection. Data confidentiality and authenticity are critical in these settings. However, the wireless connectivity, the absence of physical protection, the close interaction between WSNs and their physical environment, and the unattended deployment of WSNs make them highly vulnerable to node capture as well as a wide range of network-level attacks. Moreover, the constrained energy, memory, and computational capabilities of the employed sensor nodes limit the adoption of security solutions designed for wire-line and wireless networks. In this paper, we focus on the management of encryption keys in large-scale clustered WSNs. We propose a novel distributed key management scheme based on Exclusion Basis Systems (EBS); a combinatorial formulation of the group key management problem. Our scheme is termed SHELL because it is Scalable, Hierarchical, Efficient, Location-aware, and Light-weight. Unlike most existing key management schemes for WSNs, SHELL supports rekeying and, thus, enhances network security and survivability against node capture. SHELL distributes key management functionality among multiple nodes and minimizes the memory and energy consumption through trading off the number of keys and rekeying messages. In addition, SHELL employs a novel key assignment scheme that reduces the potential of collusion among compromised sensor nodes by factoring the geographic location of nodes in key assignment. Simulation results demonstrate that SHELL significantly boosts the network resilience to attacks while conservatively consuming nodes' resources.

Index Terms—Wireless sensor networks, secure group communications, key management, location-aware protocols, exclusion basis systems, combinatorial optimization, energy efficient design, collusion attacks.

1 Introduction

In recent years, there have been major advances in the development of low power microsensor nodes (sensors for short). The emergence of such sensors has led practitioners to envision networking a large set of sensors scattered over a wide area of interest into a wireless sensor network (WSN) for large-scale event monitoring and data collection and filtering [1]. A typical high-level architecture of a WSN consists of a large number of sensors that are capable of probing the environment and reporting the collected data to a command center using wireless channels [2]. To ensure scalability and to increase the efficiency of the network operation, sensors are often grouped into clusters. WSNs can serve many applications, such as monitoring nuclear reactors and test areas, disaster management, combat field surveillance, and border control. Sensors are significantly constrained in the amount of available resources, such as energy, storage, and computational capacity. In addition, many applications may require unattended operation of the networks. These constraints make the design and operation of WSNs considerably different from contemporary wireless networks and necessitate the development of resource conscious protocols and management techniques.

Confidentiality, authenticity, availability, and integrity are typical security goals for WSNs. Indeed, securing the network for applications, such as border control and tactical defense operations is among the main design objectives. However, the close interaction of WSNs with their physical environment and the unattended deployment of WSNs in hostile environments make them highly vulnerable to attacks. These attacks may be external such as traffic jamming and node damaging, or internal such as collusion among some nodes to reveal the fundamentals of the employed security scheme. WSN vulnerability to numerous attacks may diminish their value and curtail their use [3], [4]. Thus, secure communications in WSNs is vital to their acceptance and broader adoption.

The energy-constrained nature of WSNs, the absence of the static infrastructure, the use of wireless channels, and the large number of nodes make the task of incorporating security in WSNs a challenging problem. Most of the well-known security schemes designed for traditional wire-line and wireless networks introduce significant computation and communication overhead and, thus, cannot be readily applied to WSNs. The design of security schemes for WSNs should be geared toward resource conservation. Therefore,

- *M.F. Younis and K. Ghumman are with the Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250. E-mail: {younis, kajal1}@cs.umbc.edu.*
- *M. Eltoweissy is with the Bradley Department of Electrical and Computer Engineering, Virginia Tech, Falls Church, VA 22043. E-mail: toweissy@vt.edu.*

Manuscript received 11 Aug. 2005; accepted 21 Dec. 2005; published online 26 June 2006.
Recommended for acceptance by S. Olariu.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0368-0805.

the level of security versus the consumption of energy, computation, and memory resources constitute a design trade-off.

We observe that operations in a WSN are inherently collaborative, where the autonomous nodes forming the WSN collectively perform tasks. Therefore, secure group communications should be utilized to support efficient WSN operations. Secure group communications requires that each authorized member of a secure network have knowledge of one or more communication key(s) (also termed session keys) shared by a group(s) of nodes. The message source uses the communication keys for encrypting data packets before sending them to the group. Numerous conditions may introduce the need to evict a node, or a set of nodes. For example, nodes may experience failure due to energy depletion. Also, nodes may be compromised and exhibit anomalous behavior [5], [6], [7], [8], [9]. When a member node is evicted from the group, the communication keys known to that member must be changed in order to maintain data confidentiality. All remaining group members receive the new communication keys by secure transmission, which is typically accomplished by broadcasting a message containing the new key(s) to the group. Such a message must be indecipherable to the evicted member in order to prevent it from obtaining the new key. Rekeying messages are encrypted using administrative keys that are not known to the expelled nodes. Moreover, the administrative keys known to an evicted node should also be replaced. Consequently, key management is a core component in any secure group communications.

In this paper, we present a novel solution to the problem of key management in WSNs. Specifically, we propose a lightweight scheme for clustered WSNs based on a combinatorial formulation of the group key management problem. The solution is

1. hierarchical, resulting in scalable key management and multigranularity secure communications;
2. distributed, splitting the different activities related to key management among multiple nodes;
3. employs the Exclusion Basis System methodology [15], allowing for efficient rekeying in large networks; and
4. collusion resistant, reducing the potential for collusion among compromised nodes.

Our solution minimizes the energy and memory consumption of the security protocol through trading off the number of keys and rekeying messages. We also introduce a novel key assignment scheme for reducing the potential of collusion by factoring the geographic location of nodes in key assignment. We call our solution SHELL abbreviating its attributes, Scalable, Hierarchical, Efficient, Location-aware, and Light-weight, while implicitly implying its protection of the sensor network. SHELL is validated through simulation and is shown to boost the network resilience to attacks while imposing little burden on the resource constrained sensor nodes.

The balance of this paper is organized as follows: Section 2 describes the system and threat models and highlights the design goals. In Section 3, we discuss our proposed security solution. Section 4 addresses the problem of collusion among compromised nodes. In Section 5, we report on the validation and performance assessment of our security solution. The related work is discussed in Section 6. Finally, we conclude in Section 7.

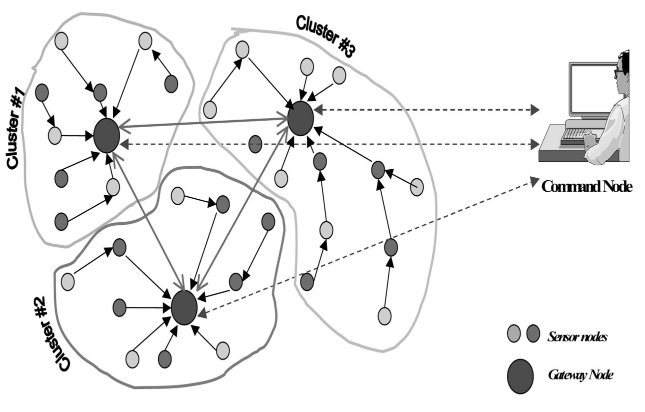


Fig. 1. Multigateway clustered sensor network.

2 System and Threat Models

2.1 System Model

We consider a wireless sensor network consisting of a *command node* and numerous sensor nodes which are grouped into clusters. The clusters of sensors can be formed based on various criteria such as capabilities, location, communication range, etc. [10], [11]. Each cluster is controlled by a cluster head, also known as *gateway*, which can broadcast messages to all sensors in the cluster. We assume that the sensor and gateway nodes are stationary and the physical location and communication range of all nodes in the network are known [12], [13]. Each gateway is assumed to be reachable to all sensors in its cluster, either directly or in multihop. Sensors perform two main functions: sensing and relaying. The sensing component is responsible for probing their environment to track a target/event. The collected data are then relayed to the gateway. Nodes that are more than one hop away from the gateway send their data through relaying nodes. Sensors communicate only via short-haul radio communication. The gateway fuses reports from different sensors, processes the data to extract relevant information and transmits it to the command node via long-haul transmission. The network architecture is depicted in Fig. 1.

Each tier of the network possesses different capabilities. The command node is resource-rich. However, the amount of traffic flowing between the command node and gateways causes the communication channel between the command node and gateways to be restrained. Most often, the command node is situated at a considerable distance from the deployment region, and might only be reachable through slow satellite links. Larger communication distances also incur increased security vulnerability and packet loss during long haul transmissions. These concerns have motivated us to reduce the role of the command node in our key management protocol, as we later explain. The gateways are moderately powerful with sufficient energy to transmit to the command node as well as perform the required key management functions. Finally, the sensor nodes are constrained in energy, computation, and communication resources. An example of the capabilities of a typical sensor node is the *MICA* mote, which includes 4K RAM and an 8-bit 4MHz processor, and runs with two AA batteries [14].

Given such resource constraints and the highly vulnerable environment, providing security services for WSNs is not a trivial task. In the remainder of this paper, we propose an efficient key management scheme for WSNs.

2.2 Threat Model

In this paper, we mainly consider an adversary that tries to manipulate the system through capturing and compromising some network nodes. No trust assumptions are made on the sensors. When sensors are captured; their memory can be read and erased or tampered with. Therefore, an adversary would know the keys of a compromised sensor. The gateways are not assumed to be tamperproof either. They can be compromised by an adversary. However, it is assumed that the compromise of a gateway is more difficult than that of a sensor. A gateway's compromise includes the uncovering of its keys through spoofing, the physical damaging of its processing and/or communication capabilities, and the manipulation of its operation after being captured by an adversary. A simplifying assumption in our design is that the adversary does not launch a coordinated attack, i.e., if more than one sensor node were to be captured, a compromised node would not be aware of the location of other compromised nodes unless they are its immediate neighbors. In addition, we assume that the adversary does not have any prior knowledge of what is stored at each node and, thus, cannot selectively direct the attack to a particular node based on what that node has.

We assume that the goal of the adversary is to uncover the keys used in the system so that he/she can discern the network operation. To achieve such a goal, the adversary attacks individual nodes and fosters collusion among compromised nodes. The main objective of node collusion is to incrementally aggregate the uncovered keys of individual nodes to a level that allows revealing all encrypted traffic in the network. The following scenarios can be identified for the compromised nodes to collude:

1. The compromised nodes have direct communication links: In such a case, collusion can be very subtle and hard to prevent.
2. The compromised nodes can only communicate through multihops: This would require the compromised nodes to be aware of the location/ID of other compromised nodes.
3. The adversary sets up separate channels for compromised nodes to communicate: This scenario implies a coordinated network attack and is not handled by SHELL. We argue that the open communication among compromised nodes or with a coordination center would make it easy to detect and thwart the attack. In addition, the adversary cannot simply use the radio of the compromised node and would have to provide an independent radio, which may not be always possible.

In this paper, we provide a solution to counter the scenario in 1) above through careful assignment of keys to communicating neighboring nodes. A fitting example of the assumed model of attacks is in a battlefield or in a border monitoring application. In such environments, attacks are typically covert and start localized in scope before spreading. For example, in a battlefield, covert infiltration is a common tactic. The use of a dedicated radio channel for coordination among members of the adversary unit would risk the detection of the infiltration attempt. Using the available radio channels of the compromised nodes would be hard to detect and gives the attackers an opportunity for causing serious damage. They may decide to stay dormant and disrupt the network operation at the most serious time, e.g., manipulate the network only when the troops start a combat operation in order to cause maximum chaos. The goal of our key assignment approach is to assign the keys in such a way that, even though compromised nodes may collude and share their keys, an adversary would not be able to access all the keys of the network unless many nodes are compromised. In other words, we try to reduce the probability that the entire network will be captured. The remainder of this paper focuses on our proposed key management solution; SHELL.

3 Description of SHELL

In this section, we present our key management scheme, SHELL, with the objective of enhancing network survivability against node capture while incurring low computation, communications, and storage overhead. We assume that the command node is secure while the gateways and sensors can be compromised by an adversary. All nodes in the network have unique identifiers. The next section overviews the principal building block of SHELL, the Exclusion Basis Systems. Section 3.2 lists the assumed capabilities and describes the role of each node in the security framework. Details of network initialization and operation follow in Section 3.3. Analysis of potential compromises and description of the recovery process are presented in Section 3.4.

3.1 The Exclusion Basis Systems (EBS)

SHELL employs an Exclusion Basis System (EBS), developed by Eltoweissy et al. in [15]. EBS is a combinatorial optimization methodology for key management of group communication setups. It exploits the trade-off between the number of administrative keys, "k," and the number of rekeying messages "m." A set of $(k+m)$ administrative keys is used to support a set of N nodes, where each node is assigned a distinct combination of "k" keys. A node can be simply admitted to the group by assigning one of the unused set of "k" keys out of the total of $C(k+m,k)$, i.e., $(k+m)!/(k!m!)$, distinct combinations. Eviction of a compromised node can be performed by broadcasting replacement of the "k" keys that the evicted node knows using the "m" keys that the node does not know. The EBS approach proves to be very scalable for large networks and enables great flexibility in network management by controlling the values of "k" and "m." Large "k" increases the storage requirements at the node, while large "m" increases communication overhead for key management. In this section, we provide an overview of the basic concepts of EBS.

An EBS is defined as a collection Γ of subsets of the set of members [15]. Each subset corresponds to a key and the elements of a subset $A \in \Gamma$ are the nodes that have that key. An EBS Γ of dimension (N,k,m) represents a situation in a secure group where there are N members numbered 1 through N, and where a key server holds a distinct key for each subset in Γ. If the subset A_i is in Γ, then each of the members whose number appears in the subset A_i know the

TABLE 1
The Canonical Matrix A for EBS(10, 3, 2)

	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9
K_1	1	1	1	1	1	1	0	0	0	0
K_2	1	1	1	0	0	0	1	1	1	0
K_3	1	0	0	1	1	0	1	1	0	1
K_4	0	1	0	1	0	1	1	0	1	1
K_5	0	0	1	0	1	1	0	1	1	1

distinct key (provided by the key server) for that subset. Furthermore, for each $t \in [1, N]$, there are m elements in Γ whose union is $[1, N] - \{t\}$. From this, it follows that the key server can evict any member t, rekey, and let all remaining members know the replacement keys for the k keys they are entitled to know, by multicasting m messages encrypted by the keys corresponding to the m elements in Γ whose union is $[1, N] - \{t\}$. Each new key is encrypted by its predecessor in order to limit decipherability only to the appropriate members.

According to EBS, every node has a unique set of keys stored at it. The total number of keys in the system can be a maximum of $k + m$ and the maximum number of key combinations is given by the binomial coefficient $C(k + m, k)$. An EBS exists when $C(k + m, k) \geq N$ (the proof is in [15]).

To construct $EBS(N, k, m)$ for feasible values of N, k, and m, we employ a canonical enumeration of all possible ways of forming subsets of k objects from a set of $k + m$ objects. We do this because the construction is based on such an enumeration. There are several algorithms for producing such a sequential enumeration. We choose an enumeration where each element of the sequence is a bit string of length $k + m$, where a 1 in the ith position of a string means that object i is included in that subset, for all $i (1 \leq i \leq k + m)$. Note that, every bit string in this enumeration will have exactly k ones.

For any k and m, let $Canonical(k, m)$ be the canonical enumeration of all $C(k + m, k)$ ways to form a subset of k elements from a set of $k + m$ objects. For the sequence of bit strings in $Canonical(k, m)$, we form a matrix A, whose $C(k + m, k)$ columns are the successive bit strings of length $k + m$, each with k ones. "A" is called the canonical matrix for $EBS(N, k, m)$. For example, the canonical matrix A for $EBS(10, 3, 2)$ contains the enumeration of all $C(5, 3)$ ways to form a subset of three keys from five keys, as shown in the Table 1.

We illustrate rekeying an EBS-based system with the help of the following example. Assume that there are 10 members in the group with keys as per Table 1. Suppose member M_0 has been compromised. Since M_0 possesses keys K_1, K_2, and K_3, these will have to be redistributed. Now, the set $(K_4 \cup K_5)$ is the set of keys known to all members except M_0. Hence, the keys used for rekeying will be K_4 and K_5. The following messages will be generated for rekeying:

Message 1: $\mathrm{E}(K_4(\mathrm{S}', \mathrm{E}(K_1(K_1')), \mathrm{E}(K_2(K_2')), \mathrm{E}(K_3(K_3'))))$
Message 2: $\mathrm{E}(K_5(\mathrm{S}', \mathrm{E}(K_1(K_1')), \mathrm{E}(K_2(K_2')), \mathrm{E}(K_3(K_3'))))$,

where $\mathrm{E}(K_1(K_1')) \rightarrow \mathrm{key} K_1'$ is encrypted with key K_1 and $\mathrm{S}'w$ is the new session key for the cluster.

The way these messages are constructed ensures that only the legitimate nodes will be able to decrypt the new keys. Thus, as a result of the above messages being broadcasted, member M_0 will be evicted and the new keys K_1', K_2', and K_3' will be made available to only those nodes that possessed keys K_1, K_2, and K_3.

A drawback of the original EBS is that it is highly vulnerable to collusion attacks. It is likely that when nodes in the system collude, their combined set of keys will reveal many keys to the colluding nodes. We present a novel solution for mitigating collusion attacks in EBS-based key management for WSNs.

3.2 System's Components and Capabilities

3.2.1 Sensor Nodes

Each sensor belongs to a cluster headed by a gateway. It is assumed that the gateway can directly reach each node in its cluster. The sensors communicate with the gateway of their cluster via one hop or multihop transmissions. The command node stores a database of all node IDs. Each sensor has a preloaded discovery key K_{sg}, as well as a one-way hash function to recompute K_{sg} as explained in Section 3.3. Each node also has two preloaded keys $\mathrm{KS}_{\mathrm{CH}}$ and $\mathrm{KS}_{\mathrm{Key}}$ for initial key distribution. In addition, every node should have the capacity for storing K administrative keys and c communication keys. The parameters K and c can be set based upon the trade off between the bandwidth utilization versus the memory available for keying functions at the sensors. We assume that the gateway is capable of detecting and identifying compromised sensors in its cluster. However, a gateway can be compromised even though it is harder to compromise a gateway than a sensor. We assume that the gateway/command node can launch an effective investigation procedure to accurately identify compromised sensors/gateway if it suspects some abnormal node behavior. The detection algorithm is beyond the scope of this paper and is part of our future research plan.

3.2.2 Gateways

Each gateway can directly communicate with at least two other gateways in the network. The keys known to each gateway are:

- K_{gc}: This is the preloaded key of the gateway and is used for direct communication between the gateway and the command node.
- $K_{\mathrm{gi,gj}}$: This is the intergateway key for communication between gateway i and gateway j. The intergateway keys are supplied by the command node during the bootstrapping phase.
- K_{sg}: This is the sensor discovery key which is provided to each gateway by the command node.

TABLE 2
List of Used Notation

Notation	***Description***
CN	Command node
C_j	Cluster j
G_j	Gateway j
S_i	Sensor i
N	Total number of sensors
N_j	Total number of sensors in cluster *j*
G_{all}	All gateways
ID_G	Gateway ID
ID_S	Sensor identifier
$G_{CH}[i]$	Gateway heading cluster *i*
$G_{K1}[i]$,$G_{K2}[i]$,...	Key generating gateways for cluster *i*
$K_{gc,j}$	Key between gateway *j* & command node
$K_{Gi,Gj}$	Keys for inter-gateway communication between gateway i and gateway j
K_{sg}	Sensor discovery key
CK[*j*]	Communication (or session) key for the cluster of gateway *j*
$KM_{i,j}$	Set of management (administrative) keys for cluster *i* from GKj[*i*]
$KM1_{Si}$, $KM2_{Si}$,... respectively	Set of management (administrative) keys for node S_i generated by GK1[*i*],GK2[*i*],…,
$KS_{Key}[S_j]$	Key shared between sensor S_j and the key generating gateway of its cluster
$KS_{CH}[S_j]$	Key shared between sensor S_j and the gateway of its cluster
Loc_ S_i	Location of sensor *i*
E (K, [data])	Encryption function of data with key K
||	Concatenation operator

Each gateway acts as the head of a cluster of a number of sensors. Each cluster will be assigned a set of distinct communication keys for data encryption. Sharing the same key among sensors in a cluster will enable selective decryption of data messages for the purpose of aggregation. Different communication keys will provide for subgrouping of nodes within a cluster based on application criteria, e.g., a separation of concerns if different subgroups are tasked with different (sets of) functions. The following are the key responsibilities of every gateway with respect to security:

- forming an EBS matrix and generating the communication key of its own cluster,
- generating administrative keys for other clusters as and when requested to do so,
- refreshing data keys of its cluster after network setup, and
- detecting and evicting compromised sensor nodes in its cluster.

3.2.3 Command Node

All tasks for the sensors are delegated by the command node. The data gathered by individual sensors is transmitted (often after some processing) to the gateways. The gateways then channel this information to the command node. The command node is assumed to be a trusted entity and cannot be compromised. The following are the key responsibilities of the command node:

- acts as a repository for valid IDs and preloaded keys (K_{sg}, KS_{CH}, and KS_{Key}) of all sensors in the region,
- authenticates the gateways and the sensors, both initially deployed as well as latently added nodes,
- distributes keys for intergateway communications and for detecting the compromise/failure of any gateway, and
- periodically performs key renewal for the intergateway communication and triggers renewal of gateway-to-sensor communication keys in order to counter potential on-going spoofing.

3.3 System Initialization and Operation

In this section, we focus on the procedures involved in system initialization and operation under normal conditions. The next section discusses the handling of compromised nodes. Table 2 lists the notation used in describing SHELL. A formal presentation of the scheme is included in Appendix A (which can be found on the CS Digital Library at http://www.computer.org/tpds/archives.htm).

3.3.1 Network Bootstrapping

We use the term, network bootstrapping, to designate the stage when gateways establish contacts among themselves, sensors are discovered, and clusters are formed. The bootstrapping process has three main phases, namely, gateway registration, sensor discovery, and clustering, and works as follows:

1. **Gateway Registration**. At the time of deployment, each gateway *i* establishes connection with the command node. The gateway broadcasts announcements of its ID along with its location encrypted

with its preloaded $K_{gc,i}$ key. Upon receiving the announcements from all gateways, the command node establishes link-specific keys for intergateway communication (that is, for each pair of gateways i and j, the command node establishes keys $K_{Gi,Gj} \neq K_{Gj,Gi}$). The command node then sends to each gateway i a message that contains its intergateway keys encrypted by $K_{gc,i}$. The gateways use the intergateway communication keys to establish contact with each other and find out which links are working and which are broken. The command node also sends each gateway the K_{sg} key to be used for establishing the initial contact with sensors. It is to be noted that even though gateways are more capable than sensor nodes, they still may be limited in resources. Consequently, we chose not to use asymmetric keys for intergateway communications. Employing asymmetric keys for such communications is a subject of further research.

2. **Sensor Discovery**. After establishing the command node to gateways and intergateway links, sensor discovery starts. Sensors broadcast, repeatedly if necessary, their ID and location to the gateways. These announcements are encrypted using K_{sg}. We assume that sufficient number of gateways is deployed in order to ensure area coverage and boost the probability of discovering all sensors. Upon reception of one of the sensor announcements, a gateway will decrypt the message and tabulate the sensor's ID and location. As soon as the sensor discovery is complete, sensors generate a new K_{sg} using a one-way hashing function, such as SHA1 [18] or MD5 [19]. The hashing function is known to the command node as well as the sensor nodes, but not disclosed to the gateways. The goal is to make the current K_{sg} keys known to the gateways obsolete. This process will be critical to the recovery from a gateway compromise as we discuss in Section 3.4.
3. **Clustering**. Sensor nodes are to be partitioned into disjoint clusters. Each cluster is managed by a gateway, denoted $G_{CH}[i]$, where i is the cluster number. The gateways collaborate among themselves to form clusters. Sensors' communication range, type, and geographical location are possible criteria for grouping the nodes into clusters [10], [11]. Each sensor should be uniquely assigned to a cluster for which the sensor is directly reachable to the cluster's gateway. Once the clustering process is completed, every gateway tabulates the ID and location of the sensors in its cluster and informs them with their cluster association. At this point, the network bootstrapping stage is concluded. It is worth noting that we pursue clustering to spread the functionality of key management among multiple gateways and prevent an individual gateway from being a single point of failure (vulnerability). In addition, clustering enable scalability by splitting other nonsecurity related network management tasks among multiple data collection nodes.

3.3.2 *Initial Key Distribution*

EBS formation. Once every gateway has a list of all sensors in its region, the initial key distribution starts with each gateway defining the EBS-based key combinations for every sensor in its cluster. The gateway performs an analysis for defining the number of administrative keys needed for the cluster. As discussed earlier, EBS offers trade-offs with respect to the storage requirements and number of rekeying messages. An increase in k, increases the storage per node, while an increase in m leads to increased overhead in communicating the new keys. Depending on the size of the cluster and available memory in the sensor, the gateway decides on the parameters k and m of an EBS. For a large cluster it might be better to increase M to limit the storage requirements per node. Other deciding factors include the expected lifetime of a sensor and how risky the deployment area is. Such factors affect the frequency of sensor eviction. It would be advisable to increase the number of keys if sensor eviction rate is expected to be high in order to minimize rekeying traffic. In this paper, we investigate another factor related to the effect of the values of k and m on the network resilience to collusion attacks (see Section 5). These quantitative results provide additional insight for selecting appropriate values.

Upon conclusion of such analysis, the list of sensors along with the EBS table of key combinations is sent to the command node. It must be noted that these key combinations are just symbolic representations of the keys and not the real keys themselves. The actual keys are generated and distributed as explained below. In Section 4, we describe a novel algorithm for designating key combinations to the various sensors so that the potential of collusion among compromised nodes is reduced.

Assigning key-generating gateways. The command node designates for each cluster C_i a number of gateways (e.g., two), other than the cluster head, that will generate administrative keys for that cluster. This is a system parameter whose value may depend on the hostility of the operating environment. Let these gateways be denoted by $G_{K1}[i]$ and $G_{K2}[i]$ (assuming that two gateways are assigned that role). The command node will then inform $G_{K1}[i]$ and $G_{K2}[i]$ about their role as key generators for the cluster C_i. Subsequently, $G_{CH}[i]$ sends the relevant portion of the EBS matrix it has formed to $G_{K1}[i]$ and $G_{K2}[i]$. Basically, $G_{CH}[i]$ informs $G_{K1}[i]$ about the keys and node association without revealing the other keys a node would have from $G_{K2}[i]$ and vice versa. For example, if the node S would have three keys from $G_{K1}[i]$ and two keys from $G_{K2}[i]$, $G_{CH}[i]$ would not let $G_{K1}[i]$ know that S would have any keys from $G_{K2}[i]$. Such an arrangement limits the ability of $G_{K1}[i]$ and $G_{K2}[i]$, if captured by an adversary, to manipulate the nodes of C_i. Based on the EBS matrix that $G_{CH}[i]$ has formed and the keys each of $G_{K1}[i]$ and $G_{K2}[i]$ has generated, the command node sends to $G_{CH}[i]$ the keys KS_{CH} of all sensors in its cluster and forwards to $G_{K1}[i]$ and $G_{K2}[i]$ the keys KS_{Key} of all sensors that are supposed to know the administrative keys they generate. Thus, $G_{CH}[i]$ would know the key combinations of each node in its cluster without generating the keys themselves and would know only one of the preloaded sensor keys. On the other hand, $G_{K1}[i]$ and $G_{K2}[i]$ would know the second prelaoded and the administrative keys for each sensor in C_i. The goal of such partitioning is to prevent a single gateway from manipulating the sensors of a particular cluster, as we later elaborate.

Another approach is to have each key generating gateway generate the entire set of k+m EBS keys, then have the sensor key generated based on the keys collected from the gateways. A threshold-based t-out-of-n approach will work in this case. This approach, however, may be more taxing on the limited resources of the sensor node.

Key distribution. Upon generation of the keys, each sensor node is informed about the set of administrative keys that it was assigned. The following steps are repeated for each cluster i:

1. $G_{K1}[i]$ constructs one message per individual administrative key for each sensor $S_j \in C_i$ which is supposed to know that key. The message is encrypted by $KS_{Key}[S_j]$, and further encrypted by $K_{GK1[i],GCH[i]}$ before being sent to $G_{CH}[i]$.
2. $G_{CH}[i]$ decrypts the message. Since $G_{CH}[i]$ does not know $KS_{key}[S_j]$, it cannot reveal the administrative key that $G_{K1}[i]$ had included in the message. $G_{CH}[i]$ then encrypts the contents using keys $KS_{CH}[S_j]$ and sends it out to the sensors in its cluster. This is repeated for every sensor in cluster "i." The recipient sensor S_j would use both of its preloaded keys to uncover its administrative keys.
3. Steps 1 and 2 above are repeated for $G_{K2}[i]$.
4. $G_{CH}[i]$ generates a communication key $CK_1[i]$, or multiple of them, for its cluster and informs gateways $G_{K1}[i]$ and $G_{K2}[i]$ about it. $G_{K1}[i]$ and $G_{K2}[i]$ in return generate messages that contain the $CK_1[i]$ encrypted with the administrative keys for cluster "i" that they have generated. These messages are then sent back to $G_{CH}[i]$ encrypted by the corresponding intergateway keys. $G_{CH}[i]$ will decrypt these messages and broadcast the contents to the sensors of its cluster.

It is worth noting that if the link layer imposes some constraints on the packet size, a message can be further partitioned. For example, instead of sending 10 administration keys in one packet, we can send them using two packets and add an indicator in the message to alert the sensor about the total number it should expect (e.g., keys 1-5 out of 10, etc.). The same can be applied to the exchange of an EBS matrix between two gateways, e.g., by sending one row or a column at a time. It is also important to note that the intergateway overhead is linear in the number of gateways and is mainly dependant on the network size and the choice of K and M.

3.3.3 Normal Network Operation

At the end of the network bootstrapping phase, the gateways would have received their intergateway keys and generated the cluster communication keys. In addition, the sensors would have their respective administrative keys as well as communication keys required for exchanging information. All data messages, either originating at any sensor or being relayed by a sensor, are to be encrypted with a communication key. Sharing same data encryption keys among the nodes in the cluster would facilitate data aggregation and dissemination. This enables secure in-network processing, which has been reported in [40] as a limitation of other key management schemes such as [8], [9], [38], [39]. During normal network operation, the key management scheme will be invoked in case of rekeying

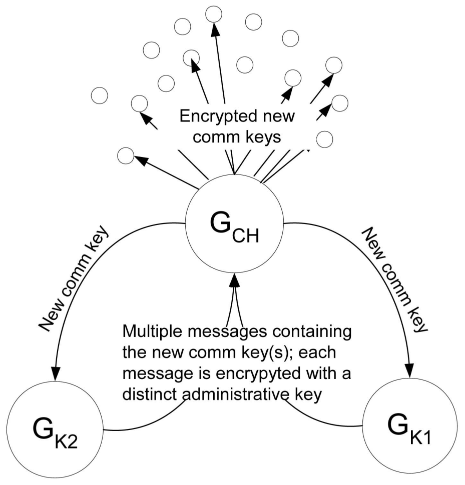

Encrypted new comm keys
G_{CH}
New comm key
New comm key
Multiple messages containing the new comm key(s); each message is encrypypted with a distinct administrative key
G_{K2}
G_{K1}

Fig. 2. Procedure of refreshing the communication key(s) of a cluster.

or the accommodation of new nodes. Handling of compromised nodes follows in the next section.

Key Refresh. The keys used in the network are periodically refreshed in order to thwart on-going cryptanalytic attacks. We emphasize that the rekeying procedure would not involve the command node. To refresh communication key(s), the cluster head $G_{CH}[i]$ sends the new communication key(s) to $G_{K1}[i]$ and $G_{K2}[i]$, which encrypt them using the administrative keys of the sensors and resend the encrypted message to $G_{CH}[i]$ for further delivery to the sensors of the cluster. Needless to say that interaction between $G_{CH}[i]$ and $G_{K1}[i]$ and $G_{K2}[i]$ are secured through the use of their respective intergateway keys. The administrative keys of the cluster can also be changed in the following way: Gateways $G_{K1}[i]$ and $G_{K2}[i]$ generate new administrative keys and encrypt them with their respective older counterpart. These messages can then be encrypted with the intergateway keys and sent to $G_{CH}[i]$. $G_{CH}[i]$ decrypts these messages and broadcasts to the sensors in its cluster. It is worth noting that $G_{CH}[i]$ will not be able to uncover the new keys since it does not know the current ones. The refresh procedure is illustrated in Fig. 2.

Addition of New Sensors. At times, the application may demand a more vigorous sensing of the environment or an extension of the area being sensed. In such scenarios, new sensors are usually added to the network. To support the inclusion of the new nodes, the command node first notifies the gateways that sensors are being deployed to boost coverage and informs the gateways of the K_{sg} to be used. The new sensors broadcast discovery messages containing their IDs and locations encrypted with K_{sg}. The gateways will decide among themselves on the cluster that the sensor should join. Once the new sensors are discovered and assigned to a cluster, every gateway that has new members will assign key combinations to these sensors and notify the command node of the sensor's ID. The command node will authenticate the sensor's identification since it has a list of all valid IDs in the system as well as IDs of all evicted sensors. Hence, the command node will only authenticate valid and unused sensor IDs. Upon certifying the authenticity of the added nodes, the command node will inform $G_{CH}[i]$, $G_{K1}[i]$, and $G_{K2}[i]$ about KS_{Key} and KS_{CH} of the new

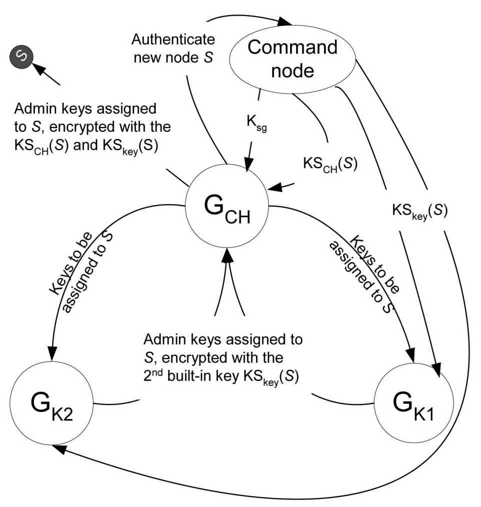

S
Authenticate new node S
Command node
Admin keys assigned to S, encrypted with the KS_CH(S) and KS_key(S)
K_sg
KS_CH(S)
KS_key(S)
G_CH
Keys to be assigned to S
Keys to be assigned to S
Admin keys assigned to S, encrypted with the 2nd built-in key KS_key(S)
G_K2
G_K1

Fig. 3. Addition of a new node to a cluster.

sensors in every cluster "i" for all clusters that have new sensors. $\mathrm{G}_{\mathrm{CH}}[i]$, $\mathrm{G}_{\mathrm{K1}}[i]$, and $\mathrm{G}_{\mathrm{K2}}[i]$ will then repeat the key distribution steps of Section 3.3.2 for the new sensors. Fig. 3 illustrates the basic protocol for node addition. In case that there is no key combination available, the gateway has to increase the number of administrative keys for the cluster. The reader is referred to [15] for description of expanding an EBS. Such scenario is unlikely unless the m and k parameters are not carefully selected. The availability of unused key combinations can also be a factor during the assignment of sensors to a cluster. If all key combinations for a cluster are assigned to nodes, that cluster may not get selected to host the added sensor.

3.4 Attack/Failure Mitigation

Key revocation procedures are involved after detecting compromised or faulty nodes. The gateway is responsible for monitoring sensors' behavior/health and detecting their failure or compromise. The command node is expected to do the same for the gateways. In this paper, we assume that there is an appropriate compromise detection mechanism employed at the command and gateway nodes. We do not distinguish the handling of a failure and a compromise. From the key management point of view, supporting revocation operations is sufficient. In this section, we discuss the handling of node failure/compromise.

3.4.1 Gateway Compromise

We assume that the compromise or the failure of the gateway will be detected by the command node. On identifying a compromised gateway, the command node notifies all gateways to relinquish the intergateway keys they share with the compromised gateway. Recovery from a compromised gateway can be handled by either deploying a new gateway or by redistributing the sensors of the compromised gateway's cluster among the remaining healthy/uncompromised gateways. The choice of the method to be used may depend on the application and/or the position of the new gateway. These two methods are discussed below:

- **Deploying a replacement**. If it is feasible to replace the faulty or compromised gateway G_{i} with another that has compatible capabilities in terms of computational resources and transmission range, the existing node-to-clusters association is not impacted, and the recovery procedure simply would be a regeneration and redistribution of keys for all nodes in C_{i}. Upon deploying the new gateway, the command node establishes keys for communication between existing gateways and such a newly deployed one. Then, the command node instructs the new gateway to form a new EBS matrix and repeat the initialization process, generate administrative keys for other clusters and repeat the key distribution process of Section 3.3.2 above. That means for every cluster $j \neq i$, for which the $\mathrm{G}_{\mathrm{evicted}}$ (where $\mathrm{G}_{\mathrm{evicted}}$ is the compromised gateway belonging to cluster i) is either $\mathrm{G}_{\mathrm{K1}}[j]$ or $\mathrm{G}_{\mathrm{K2}}[j]$, a new set of administrative keys needs to be generated for cluster "j." Given $\mathrm{G}_{\mathrm{evicted}}$'s knowledge of only $\mathrm{KS}_{\mathrm{Key}}$ for sensors in cluster "j," it cannot manipulate them. Also, since the intergateway keys are changed to isolate the evicted gateway, $\mathrm{G}_{\mathrm{evicted}}$ cannot interfere with the recovery process.
- **Sensors Redistribution**. Once a gateway is compromised, sensors belonging to its cluster are called *orphaned sensors*. In case the deployment of a new gateway is infeasible, we choose to redistribute the orphaned sensors among the remaining healthy gateways. Redistribution of the sensors is done in the following way:

  a. The command node generates new K_{sg} keys to match the keys that the sensors generated after the completion of the former discovery process. The healthy gateways are then notified with the new K_{sg}. It is worth noting that the use of a new sensor discovery key that is unknown to the gateways helps in preventing a compromised gateway from disrupting the network by erroneously initiating a discovery phase.
  b. The healthy gateways will trigger a sensor discovery phase.
  c. Each orphaned sensor is associated to a gateway according to the cluster criteria, e.g., geographical proximity.
  d. The gateways with newly added sensors will perform the procedure for adding new nodes.

If a replacement gateway is pursued and it is not reachable to the sensors of the affected cluster, a reclustering and redistribution of fresh keys would be necessary. The initialization protocol described in Section 3.3 will be then applied. Table 3 lists the potential threats of a gateway compromise and how SHELL handles them.

3.4.2 Sensor Compromise

We assume that every gateway can detect the failure or compromise of a sensor in its cluster. In case of a sensor compromise/failure, the data keys of the entire cluster will have to be changed. This is done so that the adversary cannot decrypt future messages of the network. Since G_{CH}

TABLE 3
Analysis of the Threats of a Gateway Compromise and Actions to Counter Them

Uncovered keys due to G_i compromises	Mitigation action
$K_{Gi,Gj}$ $\forall$ G_j $j \neq i$:	Command node instructs other gateways to revoke the inter-gateway keys of G_i.
$KS_{CH}[S_j]$ $\forall$ $S_j \in C_i$	Since $KS_{CH}[S_j]$ is not enough to manipulate a sensor S_j, it does not cause unrecoverable damage to the network security
$KS_{Key}[S_j]$ $\forall$ $S_j \in C_i$ $\forall$ $j \neq i$	The $KS_{Key}[S_j]$ are keys shared between the gateway G_i and the other clusters for which it acts as a key generating gateway. However, knowing KS_{Key} is insufficient for manipulating any sensor without the knowledge of their corresponding KS_{CH}, which the compromised gateway does not have.
K_{sg}	The sensors generate new K_{sg} after the discovery phase using their one way hash functions, thus making the revealed K_{sg} obsolete
EBS matrix of C_i	The recovery involves cluster, and sometimes network, re-initialization making old key assignment obsolete and irrelevant.
Administrative keys for other clusters	Such keys would be useless without involving the head of that cluster, which is not compromised. The recovery process will make these keys obsolete as well.
The communication key for the cluster	Redeploying new gateway or redistribution of sensors will establish fresh keys. The evicted gateway cannot change these keys without involving of G_{K1} and G_{k2}.

knows the EBS matrix for the cluster, it can decide which keys need to be redistributed and which keys can be used to encrypt the new keys. The EBS-based rekeying procedure is applied to evict the compromised sensors, similar to the example of Section 3.1. Node eviction is illustrated in Fig. 4 and is performed as follows:

1. Assume node S in cluster i must be evicted. Since $G_{CH}[i]$ knows the EBS matrix of its cluster, it determines the keys that S knows. These keys must be replaced.
2. $G_{CH}[i]$ informs $G_{K1}[i]$ and $G_{K2}[i]$ of the keys that need to be redistributed.
3. $G_{K1}[i]$ and $G_{K2}[i]$ will generate the new keys and encrypt them with the old keys. This rekey message is then encrypted with the keys that the evicted node does not know.
4. $G_{K1}[i]$ and $G_{K2}[i]$ then send these messages to $G_{CH}[i]$ after having encrypted them with the intergateway keys $K_{GK1[i],GCH[i]}$ and $K_{GK2[i],GCH[i]}$, respectively.
5. The cluster head $G_{CH}[i]$ decrypts the messages and broadcasts the new administrative keys to the sensors in the cluster (note the cluster head cannot decipher the new keys since it does not know the current ones).

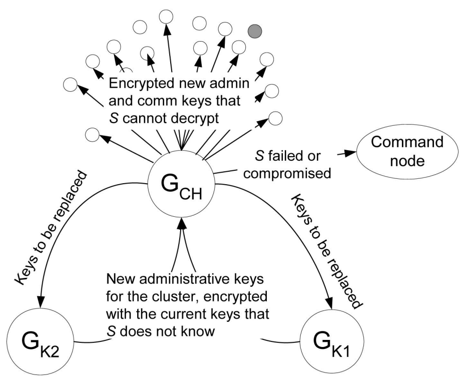

Encrypted new admin and comm keys that S cannot decrypt
S failed or compromised
Command node
G_CH
Keys to be replaced
Keys to be replaced
New administrative keys for the cluster, encrypted with the current keys that S does not know
G_K2
G_K1

Fig. 4. Procedure for evicting a faulty or a compromised node.

4 Collusion Prevention

EBS-based key management can be prone to collusion attacks [16]. Two nodes collude when they share their keys with each other. In other words, colluding nodes would grow their knowledge about the network security measures. In SHELL, keys are reused in multiple nodes and only key combinations are unique. Therefore, it is conceivable that few compromised nodes can collude and reveal all the keys employed in the network to an adversary. Such a scenario is considered as capturing the entire network since the adversary would be capable of revealing all encrypted communications in the network. Optimal assignment of key combinations to nodes so that the scope of collusion is prevented from widening to the level of capturing the network is a classical resource allocation problem and is thus NP-hard in nature. In this section, we present an efficient heuristic for key assignment that reduces the probability of capturing the network. SHELL exploits the physical proximity of nodes so that a node would share most keys with reachable nodes. The main idea is to increase the number of nodes that need to be compromised for revealing all the employed keys. In the next section, we analyze the potential for a collusion attack to lead to network capture and develop a metric for quantifying such potential. Section 4.2 builds on this analysis and presents a key assignment algorithm that increases the network resilience to collusion.

4.1 Probability of Capturing the Network

We define a neighborhood of a node S_1 as all those nodes, which are in the transmission range of S_1. In order to collude, two nodes must be in the transmission range of one another, otherwise they have to collude through a third party. According to the attack model of Section 2.1, collusion is most likely to occur when two neighboring nodes are compromised. The compromise of a node implies that the node has been captured by an adversary and can be manipulated. Thus, when two nodes in the same neighborhood are compromised, they may be manipulated to collude. As a result of collusion, the two colluding nodes would each know their own keys as well as the keys of the node it has colluded with. Collusion is transitive in nature. If S_1 is a neighbor of S_2, S_2 is a neighbor of S_3, and S_1 colludes with S_2, the resultant keys known to both of them would be $\text{Keys}(S_1) \cup \text{Keys}(S_2)$. Thereafter if S_2 and S_3 collude, the keys known to S_2 and S_3 would be $\text{Keys}(S_1) \cup \text{Keys}(S_2) \cup \text{Keys}(S_3)$. Thus, it can be seen that if multiple nodes collude, they are likely to uncover all employed keys.

In an EBS system, each key combination can be represented in the form of bit strings of k 1's and m 0's, where k is the number of keys stored at each node and m is the number of rekey messages required. A value of "1" indicates the node knows the corresponding key. The Hamming distance between any two combinations is defined as the number of bits that the two combinations differ in. Let d be the Hamming distance between a pair of key combinations. The value of d is bounded by:

$$2 \leq d \leq 2k \qquad k < m, \tag{1}$$

$$2 \leq d \leq 2m \qquad m < k, \tag{2}$$

$$2 \leq d \leq k + m \qquad k = m. \tag{3}$$

When two nodes collude, they both will know at least d keys, since d is the number of keys that they differ in. In addition, they will also know all the keys that are common to both nodes. The common keys are equal to $k - d/2$. Thus, we get the number of keys known to the two colluding nodes as $k + d/2$. This leads us to the conclusion that the lower the Hamming distance (the value of d) is, the fewer the total number of potentially revealed keys will be. This is illustrated with the help of the following example.

Consider three nodes with the following key combinations:

$S_1 : 1001100$, $S_2 : 1001010$, $S_3 : 0110001$
Hamming distance between S_1 and $S_2 = 2$
Hamming distance between S_1 and $S_3 = 6$
If S_1 and S_2 colluded, they would both have a key combination 1001110.
If S_1 and S_3 colluded, they would both have a key combination 1111101.
Therefore, collusion of S_1 and S_3
cause more keys to be uncovered than the collusion of S_1 and S_2.

Since EBS is based on the distinction between the key combinations assigned to nodes rather than the keys themselves, a minimum of two nodes would be required to collude in order to reveal all keys used in the system, and a maximum of N nodes will be required when every node knows just one key. Even though the idea of every node storing just one key seems attractive in terms of increasing the resilience of the network, having the parameter $k = 1$ leads to numerous rekey messages which increases the overhead for the network and is therefore not a desirable option. Therefore, if "p" is the probability that a node can be compromised, the probability "P" for capturing a network of N nodes would be (assuming statistical independence): $p^2 \geq P \geq p^N$.

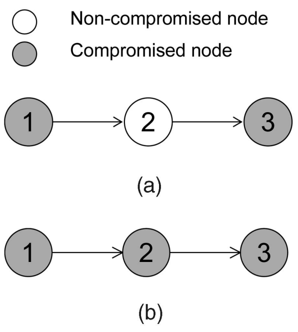


Fig. 5. (a) Nodes 1 and 3 do not collude since they are not in direct communication range. (b) Nodes 1, 2, and 3 can collude since they are in direct communication range.

We define a collusion chain to be the transitive closure of a set of colluding nodes so that the union of assigned keys includes all the $k + m$ keys of the network. For example, in Fig. 5a, although node 1 and node 3 have been compromised, they cannot collude since node 2 is an uncompromised node and there is no direct communication link between nodes 1 and 3. However, in Fig. 5b, all three nodes have been compromised and we have a collusion chain. Therefore, it is essential that the key combinations be assigned in a careful manner so that the cardinality (length) of the collusion chain increases. Long chains will decrease the probability of capturing the network (If x nodes are required in the collusion chain, the probability that the network is captured will be p^x. Since $p < 1$, as x increases p^x decreases).

Since two nodes cannot share their keys without being in direct communication range, we exploit this requirement as well as the location of nodes when assigning keys. We assign key combinations to neighboring nodes such that combinations having a lower Hamming distance are assigned to nodes that are physically closer to each other. The advantage gained by this assignment scheme is that even if these nodes collude, they will not have knowledge of all the keys in network. Further collusions will be required to unveil all the keys, which would lower the probability of capturing the network, as we explained above.

4.2 Key Assignment Approach

In this section, we present a novel approach for diminishing the probability of network capturing through collusion among compromised nodes. Before presenting our algorithm, we formulate the assignment problem and analyze its complexity.

4.2.1 Problem Formulation

Per the EBS methodology, we need to pick for each of the N nodes in the cluster a unique combination of "k" out of a set of $k + m$ keys, where k is the number of keys stored at each node and m is the number of rekey messages. Assuming that the physical location and transmission range of all nodes are known, the approach exploits the reasoning explained earlier while assigning keys to nodes. Nodes that are in such a close physical proximity that make them in the reachable communication range of each other are assigned key combinations with a lower Hamming distance than those that are far apart. In other words, we try to increase the required number of colluding nodes for capturing the network.

We represent the assignment problem as a graph $G(V, E)$, where V is the set of N nodes and E represents the set of edges. An edge is said to exist between two nodes if the nodes are in communication range of one another, in which case we refer to them as neighbors. Each edge is assigned a weight, which is equal to the Euclidean distance between the nodes at each end of the edge. The goal is to assign a unique key combination to each node such that neighbor nodes are assigned combinations with the smallest mutual Hamming distances. Edge weights are used to prioritize the neighbors. Close neighbors are favored for lower Hamming distance combinations since they generally stay reachable for long time compared to further neighbors whose reachability may deteriorate over time due to increased noise, battery exhaustions or even the placement of obstacles.

Performing such key assignment can be modeled as a register-allocation problem, widely considered in optimizing compilers [20]. The goal is to keep the most live variables stored in registers in order to expedite the program execution. Typically, variables are modeled as vertices. Two vertices are connected by an edge if they represent live variables. When possible, distinct registers are assigned to connected vertices so that overwrite is avoided and access to live variables from the nonregister memory is minimized. In our case, an edge exists between two nodes if they are in the communication range of each other and key combinations with the smallest Hamming distances are to be assigned to connected nodes in order to minimize the probability of network capturing. The register allocation problem is a known NP-hard problem [20] and, therefore, we pursue a heuristic approach, as we explain next.

4.2.2 Key Assignment Algorithm

We propose an algorithm that strives to optimize the key assignment locally and proceeds in a greedy fashion to cover the entire network. For every combination KC_a of k keys, we calculate the Hamming distance d_{ab} to every other combination KC_b, i.e., $\forall b\ d_{ab} | a \neq b\ \&\ b \in \mathrm{C}(m+k, k)$. We then sort these combinations (i.e., $KC_b\ \forall b$) in a nondecreasing order of the calculated Hamming distances (d_{ab}). The problem can now be stated as "given the keys of a node 'i,' assign key combinations to the nodes $\{\forall j | (i, j) \in E\}$ picked in an ascending order of their Hamming distance." That is, we start assigning the key combination with the least Hamming distance and so on. Since we employ homogenous key combinations, i.e., every combination has k 1's and m 0's, the Hamming distances will be bounded by (1), (2), and (3) given earlier.

The algorithm starts with the node having the largest number of neighbors, designating it as a root, and proceeds in a breadth first manner. After assigning a key combination CK_{root} to the root, we identify the nodes that are reachable to it (its neighbors) and assign some unused key combinations that have the least Hamming distance to CK_{root}. As we mentioned earlier, closer neighbors are favored for combinations that has the least Hamming distances to CK_{root}. Subsequently, for every node that has not been visited, the node's neighbors are identified and key combinations are assigned to each of them. Analogous to graphs, we refer to the neighbors of a node under consideration as its children. Each time a key combination is assigned to a node, we try to assign keys in such a way that the Hamming distance to the key combination of the node's parent is minimal, needless to say that every node is assigned a unique key combination.

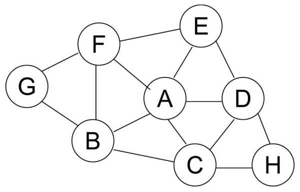


Fig. 6. An example of an 8-node network with an assumed equal link length.

Very often, two nodes may share the same parent and be neighbors to each other at the same time. In this case, the key assignment has to be done carefully. SHELL calls for assigning key combinations to the nodes so that not only the Hamming distances between the parent and the children are minimized, but also those between any two children that are connected with an edge. This is done by either selecting another unused combination that satisfies the above condition or by swapping the key combinations between the node under consideration and any previously assigned combinations. Since this can lead to potentially trying all combinations and making the assignment heuristics lean toward an exhaustive search, we limit the swapping to sibling nodes. Such restriction is logical since we pursue a breadth-first ordering and the impact of key assignment to one node will be limited to those nodes that are on the same level or one level above in the parsing hierarchy [21]. Again, a swap is accepted only if it does not increase the Hamming distance between the key combination of the node chosen for swapping and the key sets of all its neighbors. Thus, SHELL tries to minimize the maximum Hamming distance between any two combinations.

The swapping process is applied first to a child node and if it turns to be unsuccessful, swapping the keys of parent node (that is being visited) is tried. Preference would be given to swapping with a neighbor sibling that matches the visiting/nonvisiting status of the child node under consideration. Two neighbor nodes that were visited, most probably, would have been assigned key combinations with the least possible Hamming distance. Thus, swapping the keys of these visited neighbors has a high potential of being acceptable. On the other hand, it is better for nonvisited nodes to be swapped with nonvisited siblings, especially neighboring ones since the scope of the validation of such swapping will be limited to their parent in the parsing tree. It should be noted that a node whose keys were swapped before with the same node is marked as ineligible for consideration. When a suitable swap is not possible, an unused key combination with the least possible Hamming distance is picked. In the balance of the discussion, we will illustrate the idea with an example. The pseudocode for the algorithm along with the definition of the used notation can be found in Appendix B (which can be found on the CS Digital Library at http://www.computer.org/tpds/archives.htm).

Let us consider the scenario in Fig. 6. For simplicity we assume all links span the same Euclidian distance. We employ the EBS scheme with $k = 2$ and $m = 3$. Table 4 and Table 5 enumerate the possible key combinations and their mutual Hamming distances. Since the node A has the largest degree (five links), we pick it as a root and assign KC_1 to it. Then, nodes B, C, D, E, and F are assigned KC_2 to KC_5, respectively, and "A" is marked as visited. Next, we consider node B and assign KC_8 to G. While the keys of B and C fit nicely, B and F do not since the Hamming

TABLE 4
$Canonical(2, 3)$

KC					
1	1	1	0	0	0
2	1	0	1	0	0
3	1	0	0	1	0
4	1	0	0	0	1
5	0	1	1	0	0
6	0	1	0	1	0
7	0	1	0	0	1
8	0	0	1	1	0
9	0	0	1	0	1
10	0	0	0	1	1

distance between KC_2 and KC_6 is high. Because KC_7 would not make things better we look for a swap. Since node F has not been visited yet, we can swap its keys with either node C, D or E all of whom have a Hamming distance of 2 to the keys of B. Let's just pick E. Thus, F switches to KC_5 and E gets KC_6. It is now the turn for node C, for which the combinations assigned to its neighbors B and D are perfect. Node H is assigned KC_{10}. Node D has neighbors C and H with key combinations of Hamming distance of 2. However, the Hamming distance to the keys of the other neighbor E is 4. KC_7 is a perfect match for E since it fits nicely with node A and D. When visiting E no action is to be taken. Finally, node F is visited with all its neighbors having good key combinations (each with a Hamming distance of 2 to KC_5). A pictorial illustration of the described steps is shown in Fig. 7.

5 Validation and Performance Evaluation

We implemented the key assignment algorithm using C++. A network was simulated by assigning random positions to the sensors and creating an adjacency matrix assuming the same transmission ranges for all sensors. We analyze the performance of SHELL in terms of the network's resiliency to being captured and the energy consumption overhead. In this section, we discuss the validation experiments and the performance results.

5.1 Collusion Prevention

We have performed two sets of experiments to study the effectiveness of our key assignment procedure against network capture. As explained in Section 4, the network is said to be captured when all keys used in the WSN are uncovered by the adversary through colluding nodes. Both sets of experiments have been performed on a single cluster WSN deployed in an area of $500\mathrm{m} \times 500\mathrm{m}$, assuming a transmission range of 55m for all nodes. In the first set, we assess the potential of network capture through node collusion by determining the length of collusion chains. In the second set, we study the effect of the size of compromised nodes' population. These experiments are discussed separately in the sections below.

TABLE 5
Mutual Hamming Distance among KCs

KC	1	2	3	4	5	6	7	8	9	10
1	0	2	2	2	2	2	2	4	4	4
2	2	0	2	2	2	4	4	2	2	4
3	2	2	0	2	4	2	4	2	4	2
4	2	2	2	0	4	4	2	4	2	2
5	2	2	4	4	0	2	2	2	2	4
6	2	4	2	4	2	0	2	2	4	2
7	2	4	4	2	2	2	0	4	2	2
8	4	2	2	4	2	2	4	0	2	2
9	4	2	4	2	2	4	2	2	0	2
10	4	4	2	2	4	2	2	2	2	0

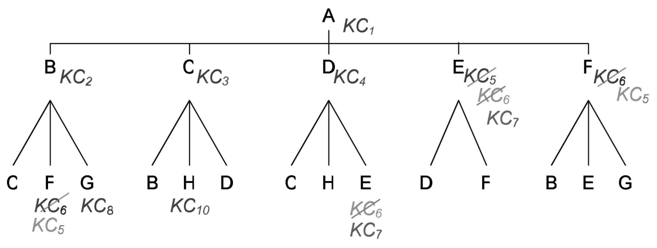


Fig. 7. Illustration for how the algorithm is applied.

5.1.1 Assessing the Potential for Collusion

As explained earlier, SHELL strives to minimize the potential for network capturing by extending the length of collusion chain necessary to reveal all keys. In the experiments, once the keys are assigned, we check for the shortest collusion chain in the network. Basically, we start checking the potential of a node colluding with each individual node in its neighborhood. We repeat this step for every node in the network generating all possible collusion sets of two nodes. We check whether any set would have gained knowledge to all keys. If not, we repeat for another iteration to form all possible collusion sets of three nodes. We repeat these steps until a collusion chain is formed. The length of the chain (number of iterations + 1) indicates the quality of the key assignment. While this procedure is analogous to finding a route from a node to every other node in the network, the complexity is still polynomial in N and |E|. In addition, we stop when all keys are known to a set of colluding nodes, which usually happens after a few iterations. It should be noted that we finish when all *used*, not all possible, keys in the network are revealed. This is important for small networks, for which only a subset of the keys are assigned to nodes. We will revisit this point when we discuss the experimental results.

The implementation of the described collusion assessment procedure is significantly simplified by the representation of key combinations as a bit vector. Two nodes colluding would mean that they collectively have knowledge of the keys known to each of them and, thus, can be performed by a logical OR operation of the two vectors. The pseudocode for the collusion assessment procedure can be found in Appendix C (which can be found on the CS Digital Library at http://www.computer. org/tpds/archives.htm).

Analysis of Results. We ran experiments for varying number of nodes in the system, and observed how the system performs for different values of k and m. Each set of readings was averaged over 10 test runs. For each run, a distinct seed is applied to generate a random network topology. We observed that with 90 percent confidence level, the simulation results stay within 6-10 percent of the sample mean.

We have chosen the k and m values such that the sum of $k + m$ is constant. Table 6 shows the possible key combinations for different values of k and m. The results plotted in

TABLE 6
Possible Number of Key Combinations for Different Values of k and m

3,7	4,6	5,5
120	210	25

Fig. 8 show the relation between different values of k and m and the corresponding number of nodes required in the collusion chain for capturing the network. As indicated, $k + m = 10$ and the number of nodes in the network varies from 20 to 200. The right vertical axis reflects the length of the collusion chain while the left vertical axis indicates the total number of keys with "k" reported in the bottom part of the bar. The experiments were repeated using random assignment of keys to nodes. The results of such random allocation of keys are plotted in Fig. 9 and are used as a baseline for comparison.

It should be noted that for small networks and large number of key combinations, SHELL tends to use the least number of keys since it minimizes the Hamming distances among neighbors. In the experiment, we have considered the network to be captured when the set of "employed" keys are revealed, which may not be all the allowed $k + m$ keys. For example, in Table 6, for $k = 4$ there are 210 different combinations. However, only 20 of these are assigned to nodes in case of a 20 node network. When applying SHELL only seven distinct (out of the 10) keys were employed. We also like to clarify that for large networks, it is infeasible to use small values of k since insufficient number of key combinations would be available. Based on Fig. 8 and Fig. 9, the following observations can be made:

1. SHELL consistently outperforms the random assignment and is distinctly more superior for small k values. To illustrate, let us consider the case for $k = 4$, $N = 120$. Our approach achieves a collusion chain length of 11 nodes compared to three when a random assignment is pursued. For a probability of a node compromise "p" of .01, the worst-case (best-case for an attacker) probability of capturing the network using SHELL is lower by a factor of 10^{16}.

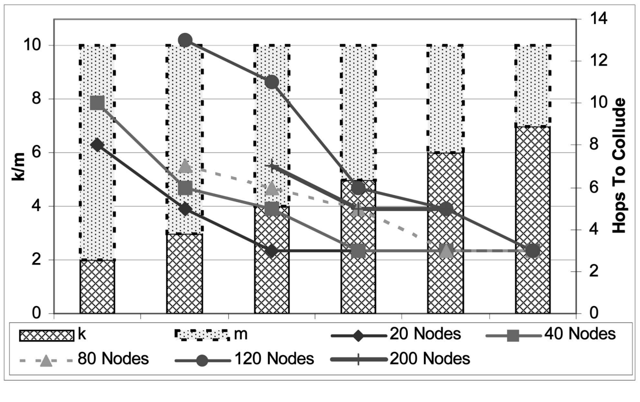


Fig. 8. The length of the collusion chain for different "k" with $k + m = 10$ while using SHELL and for different networks sizes.

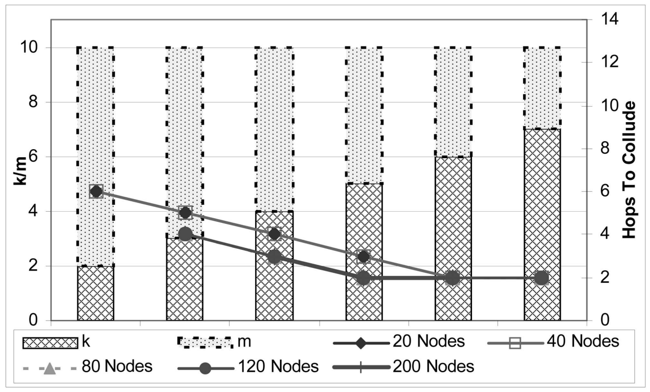


Fig. 9. The length of the collusion chain for different "k" with $k + m = 10$ under *random* key assignment and for varying networks sizes.

2. For a given number of nodes, as the k value increases, the number of hops required to collude is observed to decrease. This is in agreement with the analytical reasoning since uncovering all keys in the network would need the collusion of a fewer number of compromised nodes when the number of keys per node increases (while keeping the $k + m$ constant). In other words, the union of a fewer number of key combinations will yield all the keys in the system. However, SHELL makes this less threatening compared to the random key assignment.
3. As the network size grows, Fig. 8 indicates that the length of the collusion chain increases in most cases with one exception corresponding to the 200-nodes network. For small networks, e.g., 20, 40, and 80 nodes, a boost in the number of nodes would increase the number of distinct keys employed by SHELL and, thus, would have a positive impact (recall our earlier note regarding the tendency of SHELL in using the least number of distinct keys for small networks). However, a significant enlargement of the network can have a negative effect since most key combinations would have to be assigned to nodes and our key assignment heuristics become somewhat constrained. For example, when increasing the network size from 120 to 200 nodes, the length of the collusion chain went down from 11 to five when $k = 4$, and from five to four when $k = 5$.
4. The results of Fig. 9 show that a random key allocation, in contrary to SHELL, does not take advantage of small networks in doing a better job in collusion prevention. It is conceivable that the network will be captured by as little as two compromised nodes.
5. The results also can be insightful in determining the k and m parameters for the EBS scheme. While contemporary system design issues such as storage, communication-related energy consumption and frequency of rekeying are usually considered, the implication of the choice of k and m on the network resilience to collusion attacks can lead to a trade-off.

5.1.2 Effect of Size of Compromised Nodes' Population

In the second set of experiments, we tried to capture the number of times the adversary would have been successful in capturing the network. The experiments were performed on a

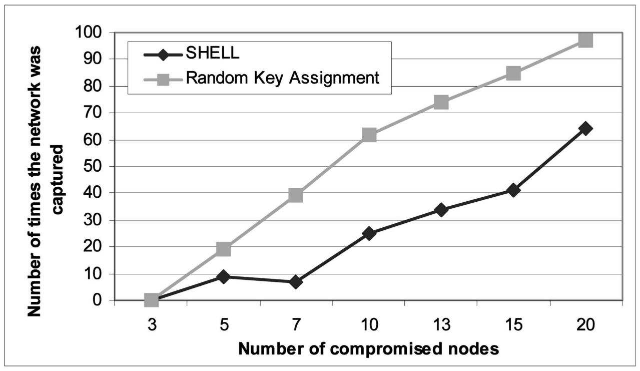


Fig. 10. The number of times, during 100 runs, the network was captured due to collusion for varying levels of node compromises.

network of 100 nodes, again deployed in 500×500 meters area and setting a transmission range of 55 meters. The k and m parameters were chosen as 3 and 7, respectively. We have randomly picked a set of compromised nodes and checked to see if they could successfully collude to uncover all the keys used in the system. We ran the simulation for different numbers of compromised nodes. The experiments were run 100 times using both our key assignment and the random key assignment methods. We then counted the number of times the network was captured in both cases. The results are shown in the Fig. 10.

From the figure, we can see that the number of times the network was captured under the random key assignment approach is greater than the number of times it was captured using SHELL. When the number of compromised nodes is few, SHELL increases the resilience of the network by at least a factor of two compared to the case of random assignment of keys. As more nodes are compromised, the gain starts to decline since it becomes more difficult to defend the network under such an intense and large-scale attack. It is worth noting that similar effect was observed when the goal of the attacker was altered to be uncovering only a subset of the networks keys and SHELL continued to demonstrate significantly more resilience than a random assignment.

5.2 Energy Consumption

Although security is an important issue in sensor networks, it is necessary to ensure that the security protocol does not weigh heavily on the scarce energy resources of the sensor nodes. Through SHELL, we have tried to provide an energy efficient security solution. We assume an efficient underlying encryption scheme, such as, XTEA [46] or RC6 [47], that incurs negligible computational cost (as compared to key communication cost). In the results discussed below, we report the average energy consumed during key management, the energy overheads due to each stage of the protocol, and the distribution of energy consumed by the sensor nodes. We compare SHELL to two security protocols. The first is the Kerberos protocol [22] and the second is a low energy key management protocol [22] proposed by Jolly et al. Kerberos is a trusted third party scheme, and requires to establish keys for every session. Jolly et al. pursues a predeployed keying strategy using preloaded sensor keys.

We used the environment of [23], which simulates the operation of a sensor network in target tracking application. The cluster head manages the tasking of sensors and the routing of data. We have instrumented traces in the source code to capture the energy consumption related to the security related activities. We have considered the energy consumed by gateways as well as that consumed by the sensors. The command node's energy is assumed to unlimited and, therefore, not considered in our analysis. The simulation parameters used are shown in Table 7. In the experiments, we have considered a network with seven clusters, and number of nodes varying from 500, 600, 700, 800, and 900.

TABLE 7
List of Simulation Parameters Used

Transmission range	150 m
Deployment region	850m x 500m
Simulation time	5 x 10^3 sec
Frequency of key renewals	5 x 10^2 sec
Data packet length	10Kbits
Routing packet length	2Kbits
Key Size (SHELL)	128 bits
Key Size (Jolly et al.)	128 bits
Key Size (Kerberos)	56 bits

Fig. 11 reports the average energy consumed by the system under normal operation. To measure the energy overhead, we have considered energy consumed by the sensors as well as the gateways in transmission and reception during the entire simulation runtime. It is important to include the gateway's energy to compare the various schemes in a comprehensive way; especially when noting that the approach of Kerberos performs all key management by a trusted third party. As depicted in Fig. 11, the average energy consumed by SHELL is in the order of 20μJ, whereas it is about 50μJ for the scheme of Jolly et al. The energy consumed by Kerberos is significantly higher (approximately 200μJ). The approach of Jolly et al. involves more message exchange than ours during key renewal as well as revocation phases. Kerberos uses a trusted third party and, therefore, not very energy efficient. It is worth noting that the number of keys in SHELL grows logarithmically with an increase in the sensors' population, and thus for large networks, the size of the EBS matrix does not grow much. Such property causes the overhead to increase at a much lower rate (note that, per node overhead decreases as the network size grows). The same is also true for increasing the number of gateways while fixing the number of sensors because the number of keys required per cluster is reduced and the EBS matrix shrinks in size.

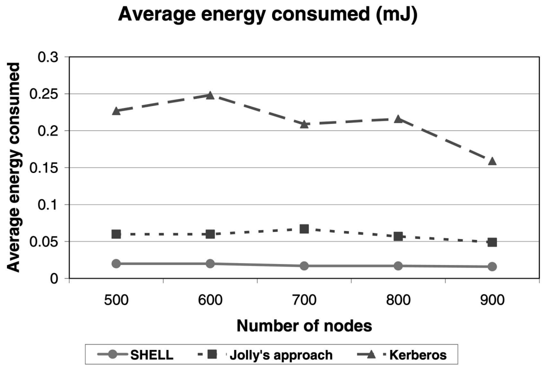


Fig. 11. Comparative average energy consumption in the network.

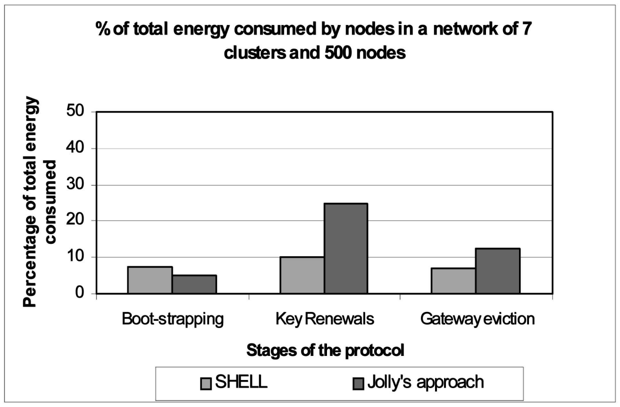


Fig. 12. Percentage of total energy consumed by each protocol's stage.

Fig. 12 shows the energy consumed by each stage of the protocol, namely, the bootstrapping, the key renewal, and the gateway eviction phases, as a percentage of the total energy consumed. The total energy represents the energy consumed by sensors and gateways for the specified duration of the simulation and includes both the energy overhead due to the security protocol and energy consumed in normal network operation, e.g., data routing. The energy consumed during key renewals represents the aggregate energy consumed during all key renewal phases (keys are refreshed periodically). Although a gateway eviction is not part of the normal operation of the network, we simulated an attack on the gateway, wherein the gateway is compromised. As explained in Section 3, we can recover from gateway's failure by using one of the two approaches: 1) deploy a new gateway and 2) redistribute sensors of the evicted gateway. During the experiment, the approach used was decided by the position of a randomly deployed new gateway. If the new gateway was within a certain range of the old gateway, it was selected; otherwise, the sensors were redistributed. The energy reported in the gateway eviction phase represents the energy consumed in recovering from gateway compromises.

Fig. 12 also indicates that the energy consumed during bootstrapping is almost the same for SHELL as well as Jolly et al.'s. During the bootstrapping phase, the energy consumed is mainly in the gateway registration phases and the sensor's announcements. These procedures are present in both schemes and therefore they consume almost the same amount of energy. On the other hand, SHELL consumes considerably lower energy during key renewals and in recovering from gateway failure. In SHELL, key renewal is a matter of a single message from the key generating gateways to the cluster-head and the broadcast of this message to the sensors of the respective cluster, whereas, in Jolly et al.'s, the new keys are generated by the command node and pushed to the gateways which further transmit these to the sensors of each cluster. Also, the length of the rekey messages is much smaller in our case.

It should be noted that the energy consumed in renewal phase reported in Fig. 12 does not represent a single key renewal, but is the energy consumed during all key renewals. The frequency of key renewals can be altered to trade off security versus energy consumed. In Jolly et al. [22], recovery from a gateway compromise is done in a manner similar to the key renewal procedure used by them, whereas SHELL either adds a new gateway, or redistributes the sensors of the evicted gateway's cluster. Adding a new gateway is not energy consuming since it is similar to key refreshing. However, redistributing the sensors requires the key establishment procedure to be repeated as in initial key establishment and, therefore, consumes considerably more energy. Since the choice of approach is decided at runtime, the energy consumed is averaged out during the 10 experiments. It must be noted that we report here the average energy consumed during gateway eviction.

6 Related Work

Group key management schemes can be classified as centralized, decentralized, or distributed [24]. Examples of centralized schemes are Group Key Management Protocol (GKMP) [25], Logical Key Hierarchy (LKH) [26], [27], One-way Function Chain Tree (OFCT) [28], and Efficient Large Group ELK [29]. Examples of decentralized group key management schemes are MARKS [30] and Kronos [31], while examples of distributed protocols include Conference Key Agreement CKA [32] and Distributed LKH [33]. The objective of most of these protocols is to balance communications with storage. ELK and Kronos also attempt to balance security and efficiency by using small-size hints and batch rekeying, respectively. A major drawback that hampers the use of most of these group key management protocols in WSNs is the lack of support for faulty and misbehaving nodes, and the overhead incurred to support key management activities including setup and rekeying.

Quite recently, the Hybrid Key Tree (HKT) scheme was proposed in [34] to balance security and efficiency using a two level hybrid key tree. HKT has a sublinear storage complexity at the controller due to the use of clusters. Cluster sizes are adjusted to resist collusion. Another effort is reported in [16] proposing an EBS-based key management solution for ad hoc networks with collusion resistance among evicted nodes. However, their work does not consider collusion among active nodes currently in session. Similar to [16], [17], [34]. The scheme presented in this paper, SHELL, is also based on key management using EBS. Distinguishing features of SHELL are the use of node proximity in a WSN to reduce the probability of collusion and the decoupling and partitioning of the keying functions among multiple nodes.

Reliance on a trusted third party (key server) to distribute the cryptographic keys has been the approach of systems like Kerberos [35] and SPINS [36]. In Kerberos, parties wishing to communicate are required to establish a secure session by requesting secret keys from a trusted key server. Although Kerberos provides authentication, its main drawback is the reliance on a single trusted key server, which is not scalable for WSNs. In addition, Kerberos requires secure key establishment for every session making it energy-inefficient. In the SPINS protocol, the base station (the gateway) is assumed to be a trusted entity. Any pair of communicating nodes uses the base station as an intermediary for trusted communication and the key distribution is achieved through one-to-one communications, which does not scale well. SHELL splits the responsibility of key generation and distribution among multiple gateways, which provides scalability and prevents the manipulation of the network due to gateway compromise.

Recently, a number of key predistribution schemes have been proposed for WSNs [8], [9], [38], [39], [40]. Eschenauer and Gligor [8] proposed a probabilistic key predistribution scheme in which a key ring of n keys, randomly chosen from a large pool of keys, is assigned to each node. For any two nodes to communicate they need to find a common key from their key rings. If there is no key in common, intermediate nodes have to be involved. This idea has been extended by Chan et al. [9] by proposing three new schemes for key predistribution in order to increase the network's resiliency to node capture. Further, Liu et al. [38] exploited the use of node location in key selection. Another effort by Du et al. [39] presents a key predistribution approach in which the probability that nodes, other than the ones already compromised, will be affected is close to zero if the number of compromised nodes is less than a threshold. However, key predistribution schemes do not provide rekeying capabilities and they may impede in-network processing, which is widely pursued in WSNs to reduce communications traffic and conserve network resources [40].

Another class of key management schemes is based on threshold secret sharing. For example, in the MOCA framework [41], some mobile nodes (known as MOCA) within the network are designated to act as a distributed certification authority. A locality driven key management scheme based on the principles of threshold cryptography has also been proposed in [42]. This scheme uses a number of mutually trusting certification authorities each of which provides authentication for its own community. Meanwhile, the key establishment scheme of [43] is based on the principles of probabilistic key sharing and threshold secret sharing. The proposed scheme is distributed in nature and is secure against collusion up to a certain number of nodes. Threshold-based cryptographic solutions generally incur high communication and computation overhead and thus may not be suitable for WSNs.

To optimize the resource requirement for key management in WSNs, Jolly et al. [22] proposed a low energy key management scheme that is based on the Identity-Based Symmetric Keying scheme [5]. Although their approach requires very few keys to be stored at each node, the rekeying procedure is inefficient due to the higher number of messages exchanged for key renewals. In addition, they require the command node to play a major role in key management. We significantly reduce the command node's role in key management. In [45], Park et al. present LiSP, a lightweight security protocol for WSNs that makes a tradeoff between security and resource consumption. LiSP employs a hierarchical keying protocol that provides efficient key distribution. However, LiSP relies on a trusted third party for authentication between any two entities. Our scheme, SHELL, use a group key for communication and administrative keys for distributing and renewing the group key periodically. Moreover, SHELL reduces the role of a central base station and distributes the key management responsibility among multiple gateways, thereby providing higher fault and attack tolerance.

7 Conclusion

The number of sensor network applications is expected to grow, especially in fields ranging from healthcare to warfare. It is important to ensure the security of data obtained from these networks. An essential component of any key-based security solution is managing the encryption keys in the system. In addition to guarding the network against attacks and localizing the effect of node compromises, key management in such vast resource-constrained networks should be efficient and scalable. In this paper, we have presented, SHELL, a distributed key management scheme for clustered sensor networks. SHELL decouples the key management functions and distributes them among multiple nodes in the network in order to provide attack/failure resilient solutions and avoid unbalanced utilization of network resources. SHELL is hierarchical; allowing scalable multigranularity secure communications. SHELL is also efficient in resource utilization. SHELL is based on Exclusion Basis Systems (EBS) methodology for group key management. EBS eliminates the need of storing a large number of keys at each sensor node. It further allows trading off the number of keys stored versus amount of network traffic due to the rekeying operations. EBS simplifies the addition and eviction of nodes and performs key refreshing through the exchange of few messages. However, EBS-based schemes can be prone to collusion attacks. Therefore, we have introduced a novel heuristic for key assignment that decreases the probability of capturing the network through the collusion of compromised nodes. SHELL exploits the physical proximity of nodes so that a node would share most keys with reachable nodes and, thus, very few additional keys would be revealed when colluding. Simulation results have demonstrated that SHELL significantly boosts the network resiliency to node capture while conservatively consuming the network's critical resources such as energy.

Future work will include further reduction of the involvement of the command node in key management, extending SHELL to handle more dynamic network topologies, for example, due to mobility of gateways or introduction of new gateways and mitigating more sophisticated attacks on sensors and gateways, for example, multihop node collusion.

References

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless Sensor Networks: A Survey," *Computer Networks,* vol. 38, no. 4, pp. 393-422, Mar. 2002.

[2] S. Tilak, N.B. Abu-Ghazaleh, and W. Heinzelman, "A Taxonomy of Wireless Microsensor Network Models," *ACM Mobile Computing and Comm. Rev.,* vol. 6, no. 2, pp. 1-8, 2002.

[3] H. Yang et al., "Security in Mobile Ad-Hoc Wireless Networks: Challenges and Solutions," *IEEE Wireless Comm. Magazine,* vol. 11, no. 1, pp. 38-47, Feb. 2004.

[4] L. Zhou and Z.J. Haas, "Securing Ad Hoc Networks," *IEEE Networks,* vol. 13, no. 6, pp. 24-30, Nov./Dec. 1999.

[5] D. Carman, P. Kruus, and B. Matt, "Constraints and Approaches for Distributed Sensor Networks Security," Technical Report 00-010, NAI Labs, Sept. 2000.

[6] G. Jolly, M. Kuscu, P. Kokate, and M. Younis, "A Low-Energy Key Management Protocol for Wireless Sensor Networks," *Proc. Eighth IEEE Symp. Computers and Comm. (ISCC '03),* June 2003.

[7] TinySec, http://www.cs.berkeley.edu/nks/tinysec/, 2006.

[8] L. Eschenauer and V. Gligor, "A Key Management Scheme for Distributed Sensor Networks," *Proc. Ninth ACM Conf. Computing and Comm. Security (CCS '02),* Nov. 2002.

[9] H. Chan, A. Perrig, and D. Song, "Random Key Predistribution Schemes for Sensor Networks," *Proc. IEEE Symp. Security and Privacy,* May 2003.

[10] G. Gupta and M. Younis, "Load-Balanced Clustering in Wireless Sensor Networks," *Proc. Int'l Conf. Comm. (ICC '03),* May 2003.

[11] O. Younis and S. Fahmy, "HEED: A Hybrid, Energy-Efficient, Distributed Clustering Approach for Ad Hoc Sensor Networks," *IEEE Trans. Mobile Computing,* vol. 3, no. 4, pp. 366-379, Oct.-Dec. 2004.

[12] K. Langendoen and N. Reijers, "Distributed Localization in Wireless Sensor Networks: A Quantitative Comparison," *Computer Networks,* vol. 43, no. 4, pp. 499-518, Nov. 2003.

[13] A. Youssef, A. Agrawala, and M. Younis, "Accurate Anchor-Free Localization in Wireless Sensor Networks," *Proc. First IEEE Workshop Information Assurance in Wireless Sensor Networks (WSNIA '05),* Apr. 2005.

[14] M. Horton et al., "Mica: The Commercialization of Microsensor Motes," *Sensors Online Magazine,* http://www.sensorsmag.com/articles/0402/40/main.shtml, Apr. 2002.

[15] M. Eltoweissy, H. Heydari, L. Morales, and H. Sadborough, "Combinatorial Optimization of Key Management in Group Communications," *J. Network and Systems Management,* vol. 12, no. 1, pp. 33-50, Mar. 2004.

[16] M. Moharram, R. Mukkamala, and M. Eltoweissy, "TKGS: Threshold-Based Key Generation Scheme for Wireless Ad Hoc Networks," *Proc. IEEE Int'l Conf. Computer Comm. and Networking (ICCCN '04),* Oct. 2004.

[17] M. Eltoweissy, M. Younis, and K. Ghumman, "Lightweight Multi-Granularity Key Management for Secure Wireless Sensor Networks," *Proc. IEEE Workshop Multihop Wireless Networks (MWN '04),* Apr. 2004.

[18] D. Eastlake and P. Jones, "US Secure Hash Algorithm 1 (SHA-1)," RFC 3174, IETF, Sept. 2001.

[19] R. Rivest, "The MD5 Message-Digest Algorithm," RFC 1320, MIT and RSA Data Security, Inc., Apr. 1992.

[20] P. Briggs, K. Cooper, K. Kennedy, and L. Torczon, "Coloring Heuristics for Register Allocation," *Proc. ASCM Conf. Program Language Design and Implementation,* June 1989.

[21] R. Diestel, *Graph Theory,* second ed. Springer-Verlag, Feb. 2000.

[22] C. Neuman and T. Ts'o, "Kerberos: An Authentication Service for Computer Networks," *IEEE Comm.,* vol. 32, no. 9, pp. 33-38, Sept. 1994.

[23] M. Younis, M. Youssef, and K. Arisha, "Energy-Aware Management in Cluster-Based Sensor Networks," *Computer Networks,* vol. 43, no. 5, pp. 649-668, Dec. 2003.

[24] S. Rafaeli and D. Hutchison, "A Survey of Key Management for Secure Group Communication," *ACM Computing Surveys,* vol. 35, no. 3, pp. 309-329, Sept. 2003.

[25] H. Harney and C. Muckenhirn, "Group Key Management Protocol (GKMP) Specification," Report # RFC 2093, The Internet Soc. (ISOC), 1997.

[26] D. Wallner, E Harder, and R. Agee, "Key Management for Multicast: Issues and Architectures," Report # RFC 2627, The Internet Soc. (ISOC), Reston, Va., 1999.

[27] K. Wong, M. Gouda, and S. Lam, "Secure Group Communications Using Key Graphs," *IEEE/ACM Trans. Networking,* vol. 8, no. 1, pp. 16-30, Feb. 2000.

[28] R. Canetti, T. Malkin, and K. Nissim, "Efficient Communication-Storage Tradeoffs for Multicast Encryption," *Proc. Conf. Advances in Cryptology (EUROCRYPT '99),* pp. 459-474 1999.

[29] A. Perrig, D. Song, and J. Tygar, "ELK, A New Protocol for Efficient Large-Group Key Distribution," *Proc. IEEE Symp. Security and Privacy,* May 2001.

[30] B. Brisco, "MARKS: Multicast Key Management Using Arbitrarily Revealed Key Sequences," *Proc. First Int'l Workshop Networked Group Comm.,* Nov. 1999.

[31] S. Setia, S. Koussih, and S. Jajodia, "Kronos: A Scalable Group Rekeying Approach for Secure Multicast," *Proc. IEEE Symp. Security and Privacy,* May 2001.

[32] C. Boyd, "On Key Agreement and Conference Key Agreement," *Proc. Information Security and Privacy: Australasian Conf.,* 1997.

[33] O. Rodeh, K. Birman, and D. Dolev, "Optimized Group Rekey for Group Communication Systems," *Proc. Network and Distributed System Security Symp.,* 2000.

[34] C. Duma, N. Shahmehri, and P. Lambrix, "A Hybrid Key Tree Scheme for Multicast to Balance Security and Efficiency Requirements," *Proc. 12th Int'l Workshop Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '03),* June 2003.

[35] A. Perrig, R. Canetti, J.D. Tygar, and D. Song, "Efficient Authentication and Signing of Multicast Streams over Lossy Channels," *Proc. IEEE Symp. Security and Privacy,* May 2000.

[36] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J.D. Tygar, "SPINS: Security Protocols for Sensor Networks," *J. Wireless Networks,* vol. 8, no. 5, pp. 521-534, Sept. 2002.

[37] C. Karlof and D. Wagner, "Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures," *J. Ad-Hoc Networks,* vol. 1, nos. 2-3, pp. 293-315, Sept. 2003.

[38] D. Liu and P. Ning, "Establishing Pairwise Keys in Distributed Sensor Networks," *Proc. 10th ACM Conf. Computer and Comm. Security (CCS '03),* Oct. 2003.

[39] W. Du, J. Deng, Y.S. Han, and P.K. Varshney, "A Pairwise Key Predistribution Scheme for Wireless Sensor Networks," *Proc. 10th ACM Conf. Computer and Comm. Security (CCS '03),* Oct. 2003.

[40] S. Zhu, S. Setia, and S. Jajodia, "LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks," *Proc. 10th ACM Conf. Computer and Comm. Security (CCS '03),* Oct. 2003.

[41] S. Yi and R. Kravets, "MOCA: Mobile Certificate Authority for Wireless Ad Hoc Networks," *Proc. Second Ann. PKI Research Workshop (PKI '03),* Apr. 2003.

[42] G. Xu and L. Iftode, "Locality Driven Key Management Architecture for Mobile Ad-Hoc Networks," *Proc. First IEEE Int'l Conf. Mobile Ad-Hoc and Sensor Systems (MASS '04),* Oct. 2004.

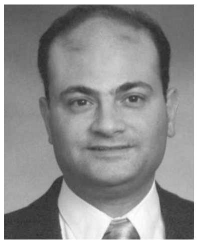
[43] S. Zhu, S. Xu, S. Setia, and S. Jajodia "Establishing Pair-Wise Keys for Secure Communication in Ad Hoc Networks: A Probabilistic Approach," Technical Report ISE-TR-03-01, George Mason Univ., Mar. 2003.

[44] B. Dutertre, S. Cheung, and J. Levy, "Lightweight Key Management in Wireless Sensor Networks by Leveraging Initial Trust," SDL Technical Report SRI-SDL-04-02, Apr. 2004.

[45] T. Park and K.G. Shin, "LiSP: A Lightweight Security Protocol for Wireless Sensor Networks," *ACM Trans. Embedded Computing Systems,* vol. 3, no. 3, pp. 634-660, Aug. 2004.

[46] R.M. Needham and D.J. Wheeler, "TEA Extensions," technical report, Computer Laboratory, Univ. of Cambridge, Oct. 1997.

[47] R. Rivest, M.J.B. Robshaw, R. Sidney, and Y.L. Lin, "The RC6 Block Cipher," *Proc. First Advanced Encryption Standard (AES) Conf.,* Aug. 1998.



Mohamed F. Younis received the BS degree in computer science and the MS degree in engineering mathematics from Alexandria University in Egypt in 1987 and 1992, respectively. In 1996, he received the PhD degree in computer science from New Jersey Institute of Technology. He is currently an assistant professor in the Department of Computer Science and Electrical Engineering at the University of Maryland Baltimore County (UMBC). Before joining UMBC, he was with the Advanced Systems Technology Group, an Aerospace Electronic Systems R&D organization of Honeywell International Inc. While at Honeywell, he led multiple projects for building integrated fault-tolerant avionics, in which a novel architecture and an operating system were developed. This new technology has been incorporated by Honeywell in multiple products and has received worldwide recognition by both the research and the engineering communities. He also participated in the development the Redundancy Management System, which is a key component of the Vehicle and Mission Computer for NASA's X-33 space launch vehicle. His technical interest includes network architectures and protocols, embedded systems, fault tolerant computing, and distributed real-time systems. Dr. Younis has four granted and three pending patents. He served on multiple technical committees and published more than 60 technical papers in refereed conferences and journals. He is a senior member of the IEEE.

Kajaldeep Ghumman received the BE degree in information technology from Sardar Patel College of Engineering, Mumbai, India, in 2002 and the MS degree in computer science from the University of Maryland Baltimore County in 2004. She is currently working as a software engineer with Symantec Corporation in Redwood City, California. Her research interest includes mobile ad hoc and sensor networks.

Mohamed Eltoweissy received the MS and BS degrees in computer engineering from Alexandria University, Egypt in 1989 and 1986, respectively, and the PhD degree in computer science from Old Dominion University in 1993. He is an associate professor in The Bradley Department of Electrical and Computer Engineering at Virginia Tech. He also holds a courtesy appointment in the Department of Computer Science. He is founder and director of the Center for Cyber Assurance and Trust (CyCare). His research interests are in the areas of information assurance and trust, networking in large-scale, unstructured and resource-constrained environments, service-oriented architectures, and group communications. He has more than 85 publications in archival journals and respected books and conference proceedings. Among his research contributions are novel combinatorial-based survivable key management schemes for sensor and ad hoc networks, service-oriented architecture for sensor networks, and stochastic models for the optimization of security protocols. Dr. Eltoweissy is the guest editor of the special issue of Elsevier's *Journal of Computer Communications on Sensor-Actuator Networks* (SANETs). He also is active in serving on program committees and US National Science Foundation panels, in journal editorials, and organization of professional meetings. He is a senior member of the IEEE, and a member of the ACM, ACM SIGBED, and ACM SIGSAC. He is also a member of Upsilon Pi Epsilon and Phi Kappa Phi. In 2003, he was nominated for the Virginia SCHEV outstanding faculty awards; the highest honor for faculty in Virginia.