LEAP+: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks

SENCUN ZHU
The Pennsylvania State University
and
SANJEEV SETIA
George Mason University
and
SUSHIL JAJODIA
George Mason University

We describe LEAP+ (Localized Encryption and Authentication Protocol), a key management protocol for sensor networks that is designed to support in-network processing, while at the same time restricting the security impact of a node compromise to the immediate network neighborhood of the compromised node. The design of the protocol is motivated by the observation that different types of messages exchanged between sensor nodes have different security requirements, and that a single keying mechanism is not suitable for meeting these different security requirements. LEAP+ supports the establishment of four types of keys for each sensor node – an individual key shared with the base station, a pairwise key shared with another sensor node, a cluster key shared with multiple neighboring nodes, and a global key shared by all the nodes in the network. LEAP+ also supports (weak) local source authentication without precluding in-network processing. Our performance analysis shows that LEAP+ is very efficient in terms of computational, communication, and storage costs. We analyze the security of LEAP+ under various attack models and show that LEAP+ is very effective in defending against many sophisticated attacks such as HELLO flood attacks, node cloning attacks, and wormhole attacks. A prototype implementation of LEAP+ on a sensor network testbed is also described.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*; D.4.6 [**Operating Systems**]: Security and Protection—*Cryptographic controls*; K.6.5 [**Management of Computing and Information Systems**]: Communication Networks—*Security and Protection*

General Terms: Security,Algorithm,Design

Additional Key Words and Phrases: In-network Processing, Key Erasure, Key Management, Pairwise Key, Sensor Networks

A preliminary version of this paper appeared in Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS 03), pages 62–72, October 2003.

Authors' addresses: Sencun Zhu, Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802. Sanjeev Setia and Sushil Jajoida, Center for Secure Information Systems, George Mason University, Fairfax, VA 22030.
Email: szhu@cse.psu.edu, {setia, jajodia}@gmu.edu.

1. INTRODUCTION

Many sensor systems are deployed in unattended and often adversarial environments such as a battlefield. Hence, security mechanisms that provide confidentiality and authentication are critical for the operation of many sensor applications. Providing security is particularly challenging in sensor networks due to the resource limitations of sensor nodes. As a specific example, it is not practical to use asymmetric cryptosystems in a sensor network where each node consists of a slow (7.8 MHz) under-powered processor with only 4 KB of RAM (Mica2 Motes [xbow 2005]). Thus, most key management protocols for sensor networks are based upon symmetric key algorithms.

A fundamental issue that must be addressed when using key management protocols based on symmetric shared keys is the mechanisms used for establishing the shared keys in the first place. The limited energy, communication bandwidth and computational capacities of sensor nodes make protocols such as TLS [Dierks and Allen 1999] and Kerberos [Kohl and Neuman 1993] proposed for wired networks impractical for use in large-scale sensor networks. At present, the most practical approach for bootstrapping secret keys in sensor networks is to use pre-deployed keying in which keys are loaded into sensor nodes before they are deployed. Several solutions based on pre-deployed keying have been proposed in the literature including approaches based on the use of a global key shared by all nodes [Basagni et al. 2001; Carman et al. 2000], approaches in which every node shares a unique key with the base station [Perrig et al. 2001], and approaches based on random key sharing [Chan et al. 2003; Du et al. 2003; Eschenauer and Gligor 2002; Liu and Ning 2003a].

An important design consideration for security protocols based on symmetric keys is the degree of key sharing between the nodes in the system. At one extreme, we can have network-wide keys that are used for encrypting data and for authentication. This key sharing approach has the lowest storage costs and is very energy-efficient since no communication is required between nodes for establishing additional keys. However, it has the obvious security disadvantage that the compromise of a single node will reveal the global key.

At the other extreme, we can have a key sharing approach in which all secure communication is based on keys that are shared pairwise between two nodes. From the security point of view, this approach is ideal since the compromise of a node does not reveal any keys that are used by the other nodes in the network. However, under this approach, each node will need a unique key for every other node that it communicates with. Moreover, in many sensor networks, the immediate neighbors of a sensor node cannot be predicted in advance; consequently, these pairwise shared keys will need to be established after the network is deployed.

A unique issue that arises in sensor networks that needs to be considered while selecting a key sharing approach is its impact on the effectiveness of in-network processing [Karlof et al. 2004]. In many applications, sensors in the network are organized into a data fusion or aggregation hierarchy for efficiency. Sensor readings or messages from several sensors are processed at a data fusion node and aggregated into a more compact report before being relayed to the parent node in the data fusion hierarchy [Intanagonwiwat et al. 2000]. Passive participation is another form of in-network processing in which a sensor node can take certain actions based on overheard messages [Karlof et al. 2003; Madden et al. 2002], e.g., a sensor can decide to not report an event if it overhears a neighboring node reporting the same event.

Particular keying mechanisms may preclude or reduce the effectiveness of in-network processing. As a specific example, consider a sensor node that transmits its readings along a multi-hop path to a sink node, e.g., a base station. If these readings are encrypted with a key that is private to the base station and the node itself, this effectively precludes any aggregation of sensor readings at any nodes enroute to the base station. Similarly, to support passive participation, it is essential that intermediate nodes are able to decrypt or verify a secure message exchanged between two other sensor nodes. Thus, passive participation when messages are encrypted or authenticated is only possible if multiple nodes share the keys used for encryption and authentication. Clearly, if a pairwise shared key is used for encrypting or authenticating a message, it effectively precludes passive participation in sensor networks.

Contributions We describe LEAP+ (Localized Encryption and Authentication Protocol)[1], a key management protocol for sensor networks that is designed to support in-network processing, while providing security properties similar to those provided by pairwise key sharing schemes. In other words, the keying mechanisms provided by LEAP+ enable in-network processing, while restricting the security impact of a node compromise to the immediate network neighborhood of the compromised node.

LEAP+ includes support for multiple keying mechanisms. The design of these mechanisms is motivated by the observation that different types of messages exchanged between sensor nodes have different security requirements, and that a single keying mechanism is not suitable for meeting these different security requirements. Specifically, LEAP+ supports the establishment of four types of keys for each sensor node – an individual key shared with the base station, a pairwise key shared with another sensor node, a cluster key shared with multiple neighboring nodes, and a global key shared by all the nodes in the network. Moreover, the protocol used for establishing these keys for each node is communication- and energy-efficient, and minimizes the involvement of the base station.

LEAP+ also includes an efficient protocol for providing weak local broadcast authentication based on the use of one-way key chains. A salient feature of the authentication protocol is that it supports source authentication (unlike a protocol where a globally shared key is used for authentication) without preventing passive participation (unlike a protocol where a pairwise shared key is used for authentication).

Organization The rest of this paper is organized as follows. We discuss our design goals and assumptions in Section 2, then present the LEAP+ protocol in detail in Section 3. The local broadcast authentication protocol is described in Section 3.6. In Sections 4 and 5, we analyze the performance and security of the LEAP+ protocol. A prototype implementation of LEAP+ is described in Section 5.4. We discuss related work in Section 6 before concluding the paper in Section 7.

[1]This protocol was called LEAP in our conference version. Because it is much improved in this journal version, we call it LEAP+.

2. ASSUMPTIONS AND DESIGN GOALS

We describe below our assumptions regarding the sensor network scenarios in which our keying protocols will be used, followed by a discussion of our design goals.

2.1 Network and Security Assumptions

We assume a static sensor network, i.e., sensor nodes are not mobile. The base station, acting as a controller (or a key server), is assumed to be a laptop class device and supplied with long-lasting power. The sensor nodes are similar in their computational and communication capabilities and power resources to the current generation sensor nodes, e.g., the Berkeley MICA motes [xbow 2005]. We assume that every node has space for storing hundreds of bytes of keying materials. The sensor nodes can be deployed via aerial scattering or by physical installation. We assume however that the immediate neighboring nodes of any sensor node are not known in advance.

Because wireless communications use a broadcast transmission medium, we assume that an adversary can eavesdrop on all traffic, inject packets, or replay older messages. We assume that if a node is compromised, all the information it holds becomes known to the attacker. However, we assume that the base station will not be compromised. Moreover, we do not address attacks against the physical layer and the media access control layer [Wood and Stankovic 2002].

2.2 Design Goals

LEAP+ is designed to support secure communications in sensor networks; therefore, it provides the basic security services such as confidentiality and authentication. In addition, LEAP+ is designed to meet several security and performance requirements that are specific to sensor networks.

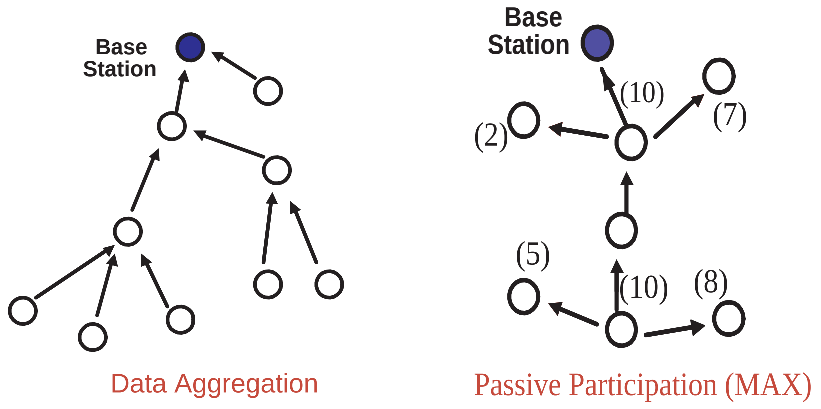
—**Supporting Various Communication Patterns** There are typically three types of communication patterns in sensor networks: unicast (addressing a message to a single node), local broadcast (addressing a message to all the nodes in the neighborhood), and global broadcast (addressing a message to all the nodes in the network). Different security mechanisms providing confidentiality and authentication are needed to support all these communication patterns.

—**Supporting In-network Processing** Security mechanisms should permit in-network processing operations such as data aggregation and passive participation (See Figure 1). In-network processing can significantly reduce energy consumption in sensor networks.

—**Survivability** Due to the unattended nature of sensor networks, an attacker could launch various security attacks and even compromise sensor nodes without being detected. To survive in such a harsh environment, the compromise of a few sensor nodes should not break the security of the entire network. That is, if an attack succeeds, its impact should be minimized and localized to a small region.

—**Energy Efficiency** Due to the limited battery lifetime of sensor nodes, security mechanisms for sensor networks must be energy efficient. The number of message transmissions and the number of expensive computations should be minimized.

—**Avoiding Message Fragmentation** A unique challenge in sensor networks



(a) A sensor node aggregates the values received from its children and forwards the aggregated value to its parent

(b) A node can suppress the transmission of its own sensor reading if it is smaller than an overheard reading.

Fig. 1. Examples of In-network Processing

arises from the small packet size supported by underlying communication protocols. In TinyOS [Hill et al. 2000], the operating system for Mica series sensors [xbow 2005], the default supported packet size is only 36 bytes. Thus, messages in a security protocol have to be small enough to fit within one packet to avoid message fragmentation. Message fragmentation is very undesirable for sensor networks because it increases the implementation complexity. The relatively high packet loss rates found in sensor networks and the limited buffer space of sensor nodes also contribute to this difficulty.

As we will show in the following sections, LEAP+ meets all these requirements, except that it does not include a global broadcast authentication protocol. We assume the existence of such a protocol, e.g., μTESLA [Perrig et al. 2001].

3. LEAP+: LOCALIZED ENCRYPTION AND AUTHENTICATION PROTOCOL

LEAP+ provides multiple keying mechanisms for providing confidentiality and authentication in sensor networks. We first motivate and present an overview of the different keying mechanisms in Section 3.1 before describing the schemes used by LEAP+ for establishing these keys. The local broadcast authentication mechanism that is part of LEAP+ is discussed separately in Section 3.6.

3.1 Overview

The packets exchanged by nodes in a sensor network can be classified into several categories based on different criteria, e.g., control packets vs. data packets, broadcast packets vs. unicast packets, queries or commands vs. sensor readings, etc. The security requirements for a packet will typically depend on the category it falls in. Authentication is required for all types of packets, whereas confidentiality may only be required for some types of packets. For example, routing control information usually does not require confidentiality, whereas (aggregated) readings reported by a sensor node and the queries sent by the base station may require confidentiality.

We argue that *no single* keying mechanism is appropriate for all the secure communications that are needed in sensor networks. As such, LEAP+ supports the establishment of four types of keys for each sensor node – an individual key shared with the base station, a pairwise key shared with another sensor node, a cluster key shared with multiple neighboring nodes, and a global key that is shared by all the nodes in the network. We now discuss each of these keys in turn and describe our reasons for including it in our protocol.

Individual Key. Every node has a unique key that it shares with the base station. This key is used for secure communication between the node and the base station. For example, a node can use its individual key to compute message authentication codes (MACs) for its sensed readings if the readings are to be verified by the base station. A node may also send an alert to the base station if it observes any abnormal or unexpected behavior of a neighboring node. Similarly, the base station can use this key to encrypt any sensitive information, e.g., keying material or special instructions, that it sends to an individual node.

Global Key. This is a secret key shared by all nodes in the network and the base station. It is mainly used by the base station for encrypting messages that are broadcast to the whole network. For example, the base station may broadcast queries or commands to the entire network. Note that from the confidentiality point of view there is no advantage to separately encrypting a broadcast message using the individual key of each node. However, since the global key is shared among all the nodes in the network, an efficient rekeying mechanism is necessary for updating this key after a compromised node is revoked.

Cluster Key. A cluster key is a key shared by a node and all its neighbors, and it is used for securing locally broadcast messages, e.g., routing control information, or securing sensor messages which can benefit from passive participation. LEAP+ provides each node a unique cluster key shared with all its neighbors for securing its messages. Its neighbors use the same key for decrypting or verifying its messages; they have their own cluster keys for securing their own messages.

Pairwise Shared Key. Every node shares a pairwise key with each of its immediate neighbors. In LEAP+, pairwise keys are used for securing communications that require privacy or source authentication. For example, a node can use its pairwise keys to secure the distribution of its cluster key to its neighbors, or to secure the transmission of its sensor readings to an aggregation node. Note that the use of pairwise keys precludes passive participation.

In the following subsections, we describe the schemes provided by LEAP+ to establish and update these keys for each node. The key establishment (and re-keying) protocol for the global key uses cluster keys, whereas cluster keys are established (and re-keyed) using pairwise shared keys.

Notation We list below notations which appear in the rest of this discussion.

—N is the number of nodes in the network.

—u, v (letters in lower case) are principals such as communicating nodes.

—$\{f_k\}$ is a family of pseudo-random functions.

—$\{s\}_k$ means encrypting message s with key k.

—$MAC(k, s)$ is the message authentication code (MAC) of message s using a symmetric key k.

From a key K a node can derive other keys for various security purposes. For example, a node can use $K' = f_K(0)$ for encryption and use $K'' = f_K(1)$ for authentication. For ease of exposition, in the following discussion we simply say that a message is encrypted or authenticated with key K, although the message is really encrypted with K' or authenticated with K''.

3.2 Establishing Individual Node Keys

Every node has an individual key that is only shared with the base station. This key is generated and pre-loaded into each node prior to its deployment.

The individual key IK_u for a node u (each node has a unique id) is generated as follows: $IK_u = f_{K^m}(u)$. Here f is a pseudo-random function and K^m is a master key known only to the controller. In this scheme the controller might only keep its master key to save the storage needed to keep all the individual keys. When it needs to communicate with an individual node u, it computes IK_u on the fly. Due to the computational efficiency of pseudo random functions, the computational overhead of this step is negligible.

3.3 Establishing Pairwise Shared Keys

In this paper, we focus on establishing pairwise keys that are shared only between nodes and their *immediate* neighbors (i.e., one-hop neighbors). A sensor node will need to communicate with its immediate neighbors in any sensor network application. Therefore, this type of pairwise key is most commonly used. If a sensor network application has special requirements of establishing pairwise keys for two nodes that are multiple hops away, a multiple-path scheme [Zhu et al. 2003] or a probabilistic key sharing scheme [Du et al. 2003; Liu and Ning 2003a; Zhu et al. 2003] may be employed. This section presents two schemes for establishing pairwise keys - a basic scheme, and then an extended scheme that provides stronger security at the expense of greater implementation complexity and larger storage requirements.

3.3.1 *The Basic Scheme.* For nodes whose neighbors are predetermined (e.g., when nodes are deployed via physical installation), they can be simply preloaded with the pairwise keys shared with their neighbors. However, here we are interested in establishing pairwise keys for sensor nodes that are unaware of their neighbors until they have been deployed (e.g., when nodes are deployed via aerial scattering).

Our approach exploits the special property of sensor networks consisting of stationary nodes that the set of neighbors of a node is relatively static, and that a sensor node that is being added to the network will discover most of its neighbors at the time of its initial deployment. Second, it is our belief that a sensor node deployed in a security critical environment must be manufactured to sustain possible break-in attacks at least for a short time interval (say several seconds) when captured by an adversary; otherwise, the adversary could easily compromise all the sensor nodes in a sensor network and then take over the network. Hence, instead of assuming that sensor nodes are tamper-proof which often turns out not to be true [Anderson and Kuhn 1996], we assume there exists a lower bound on the time interval T_{min} that is necessary for an adversary to compromise a sensor node, and that the time T_{est} for a newly deployed sensor node to discover its immediate neighbors is smaller than T_{min}. Based on their real experiments, Deng et al. [Deng et al. 2005] showed that it is possible to obtain copies of all the memory and data of a Mica2 mote in tens of seconds or minutes after a node is captured, given the proper level of experience and tools. Our experiment in Section 5.4 shows that in practice T_{est} is less than 10 seconds for a network of reasonable node density (up to 20 neighbors). Therefore, we believe that in practice it is a reasonable assumption that $T_{min} > T_{est}$.

Below we describe our scheme in detail, given this lower bound T_{min}. Note that the approach used for pairwise key establishment by nodes that are incrementally added to the network is identical to the approach used at the time of initial network deployment. The four steps described below demonstrate the way a newly added node u establishes a pairwise key with each of its neighbors that were deployed earlier.

Key Pre-distribution. The controller generates an initial key K_{IN} and loads each node with this key. Each node u derives a master key $K_u = f_{K_{IN}}(u)$.

Neighbor Discovery. Immediately after it is deployed, node u tries to discover its neighbors by broadcasting a HELLO message that contains its id. It also starts a timer that will fire after time T_{min} and trigger key erasure as discussed below. Node u waits for each neighbor v to respond to the HELLO message with an ACK message that includes its id, v. The ACK from a neighbor v is authenticated using its master key K_v, which was derived as $K_v = f_{K_{IN}}(v)$. Since node u knows K_{IN}, it can also derive K_v and then verify node v's identity.

$$\begin{aligned} u \longrightarrow * :&\quad u. \\ v \longrightarrow u :&\quad v, MAC(K_v, u|v). \end{aligned}$$

Pairwise Key Establishment. Node u computes its pairwise key with v, K_{uv}, as $K_{uv} = f_{K_v}(u)$. Node v can also compute K_{uv} in the same way. K_{uv} serves as their pairwise key. No message is exchanged between u and v in this step. Note that node u does not have to authenticate itself to node v by sending a special message, because any future messages from node u authenticated with K_{uv} will prove node u's identity.

Key Erasure. When its timer expires after T_{min}, node u erases K_{IN} and all the master keys (K_v) of its neighbors, which it computed in the neighbor discovery phase. Note that node u does not erase its own master key K_u. Every node keeps its own master key.

After the above steps, node u will have established a pairwise shared key with each of its neighbors and the pairwise key is used for securing data exchanged between them. There is no need for two nodes to use one pairwise key in one direction and another key in the reverse direction during their secure communication. Further, no nodes in the network possess K_{IN}. An adversary may have eavesdropped on all the traffic in this phase, but without K_{IN} it cannot inject erroneous information or decrypt any of the messages. An adversary compromising a sensor node T_{min} after the node was deployed only obtains the keying material of the compromised node, not that of any other nodes. When a compromised node is detected, its neighbors simply delete the keys that were shared with this node.

The above scheme can be further simplified when two neighboring nodes, say u and v, are added at the same time. For example, if u receives v's response to u's HELLO before u responds to v's HELLO, u will suppress its own response. However, if u and v finish their neighbor discovery phase separately, in the pairwise key establishment step they will have two different pairwise keys, K_{uv} and K_{vu}. In this case, they may choose K_{uv} as their pairwise key if $u < v$.

Performance Analysis Our pairwise key establishment scheme incurs the following computational overhead. The joining node needs to verify a MAC from every neighbor and evaluate a pseudo random function to generate their pairwise key. Every neighbor node computes a MAC and generate a pairwise key. The communication overhead for establishing a pairwise key includes an ACK message, which has two fields: a node id and a MAC. A HELLO message only includes a node id. Both these messages easily fit into one packet. Moreover, storage space is required for only one key, K_{IN}. Thus, the computational, communication, and storage overhead of our scheme for pairwise key establishment is very small.

Security Analysis A critical assumption made by our scheme is that the actual time T_{est} to complete the *neighbor discovery* phase is smaller than T_{min}. We believe that this is a reasonable assumption for many sensor networks and adversaries. The current generation of sensor nodes can transmit at the rate of 19.2 Kbps [xbow 2005] whereas the size of an ACK message is very small (12 bytes if the size of an id is 4 bytes and the MAC size is 8 bytes). Packet losses, due to unreliable channel and collisions, do pose a challenge to our scheme (as well as to any other protocol for sensor networks). In Section 5.4, we discuss several techniques to reduce packet losses so that a sensor node can establish pairwise keys with most, if not all, of its neighbors within T_{min}.

In the scheme described above, a HELLO message is not authenticated. An adversary may exploit this to launch resource consumption attacks by injecting a large number of HELLO messages. For every HELLO message it receives from a new neighbor, a node has to compute a MAC, send back an ACK message, and also maintain an entry in a neighbor table. This wastes CPU cycles and also consumes communication and memory resources. The message overhead is a much bigger concern than the computation overhead. It has been shown that the energy for computing one MAC is equivalent to that for transmitting only a single byte [Liu and Ning 2003a]. In addition, this attack could cause buffer overflow not only because there is only very limited RAM (e.g., 4 KB), but also because TinyOS only supports static memory allocation.

We consider two solutions to mitigate this attack. Although these solutions involve the computation of a MAC for each HELLO message, they can prevent communication and memory resources from being consumed due to this attack. First, the network controller can pre-load a new sensor node with the current global key (in addition to the initial network key K_{IN}). A new node can authenticate its HELLO message to its neighbors using the current global key. A false HELLO message will be detected and dropped immediately; thus, no ACK message will be sent and no state will be kept for the claimed neighbor. This approach however can only prevent outsider attacks. An insider adversary might know the current global key if it has compromised a sensor node that was deployed earlier. Our second solution adds some randomness into the ids of the newly added nodes such that false ids will be detected and dropped. This solution will be described in more detail in Section 3.3.2.

Furthermore, to increase the difficulty for an adversary to recover K_{IN} after it has physically captured a sensor node, the node can copy K_{IN} from non-volatile memory into volatile memory as soon as it is powered on, while erasing the copy of the key in the non-volatile memory. An implicit assumption here is that a sensor node is able to erase a key completely. Another implicit assumption is that node u will not keep the master key of another node v. We believe as long as the program loaded in a sensor node is executed correctly, this situation will not occur.

Our scheme has a unique security property that is not possessed by other pairwise key establishment schemes [Chan et al. 2003; Du et al. 2003; Eschenauer and Gligor 2002; Zhu et al. 2003; Liu and Ning 2003a]. In LEAP+, only newly deployed nodes possess the credentials, i.e., K_{IN}, to be able to establish keys with their neighbors. Once a node has erased K_{IN}, it will not be able to establish a pairwise key with any other nodes that have also erased K_{IN} (note that nodes deployed in the future will be able to establish a pairwise key with it). This helps prevent *node cloning attacks* or *node replication attacks* [Parno et al. 2005]. In a cloning attack, an attacker loads its own nodes with the keys of a compromised node, then deploys these cloned nodes in different locations of the sensor network. These cloned nodes then try to establish pairwise keys with their neighbors. Once they are accepted by their neighbors, they can launch various insider attacks such as injecting false data packets. Consequently, an attacker might only need to compromise a few sensor nodes to bring down the entire network due to the unattended nature of a sensor network. The reason that our scheme is robust to this attack is that a cloned node will not possess K_{IN} and thus cannot establish pairwise keys with nodes that are not the neighbors of the compromised nodes. Thus, our scheme can localize the security impact of a node compromise. On the other hand, we note that in the scenario where a node could be compromised before deployment, our scheme cannot prevent node cloning attacks. An attacker may introduce new nodes with the same or different node ids. Under the same assumption, however, it is harder for the attacker to make nodes with different ids in the other pairwise key establishment schemes.

3.3.2 *An Extended Scheme.* In the basic scheme, a newly added sensor node uses the same initial network key K_{IN} to derive its master key and its pairwise keys shared with its neighbors. Given the assumption that a sensor node cannot be compromised within T_{min}, the basic scheme is secure because K_{IN} is not compromised. Next we consider more powerful attacks by assuming that K_{IN} can be

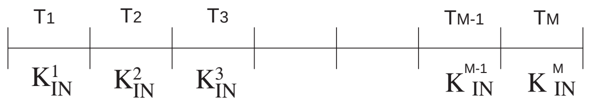


Fig. 2. The lifetime of a sensor network is divided into M intervals, and each interval has its own initial key.

compromised (either by compromising a sensor node within T_{min} or via a security breach at the mission authority). As a result, all the pairwise keys in the network will be compromised. Moreover, an attacker will be able to add its own new nodes into the sensor network since it knows K_{IN}. Below we discuss several countermeasures that can be used to mitigate these attacks.

First, we describe a simple scheme to prevent an attacker from deriving the master keys and pairwise keys of the nodes that were deployed earlier or will be deployed later, even if the attacker has managed to obtain the initial key K_{IN}. The basic idea behind this scheme is to remove the dependence on a single initial key K_{IN}; instead, there is a sequence of initial keys used for deriving the master keys of individual nodes.

Assume that there are at most M (batched) node addition events for a sensor network application and that these M events occur in M intervals $T_1, T_2, ..., T_M$, respectively, where these intervals could be of different lengths. The network controller randomly generates M keys: $K_{IN}^1, K_{IN}^2, ..., K_{IN}^M$. These keys serve as initial keys. Fig. 2 illustrates the mapping between keys and time intervals. As we did in the basic scheme, we now describe how a new node u is initialized and added to the network during the time interval T_i.

Key Pre-distribution. A node u to be deployed during time interval T_i is loaded with the initial key K_{IN}^i, from which it derives its master key $K_u^i = f_{K_{IN}^i}(u)$ for the current time interval T_i. It is also loaded with master keys $K_u^j = f_{K_{IN}^j}(u)$ for all future intervals $i < j \leq M$ but not any initial keys K_{IN}^j corresponding to those time intervals.

Neighbor Discovery. When it is deployed, node u broadcasts a HELLO message and initializes a timer to fire after time T_{min}. The HELLO message contains its id and the interval id i. Each neighbor v responds with an ACK message including its identity. The ACK from every neighbor v is authenticated using the current master key K_v^i of node v. Since node u knows K_{IN}^i, it can derive K_v^i and then verify the ACK message.

$$\begin{aligned} u \longrightarrow * : &\quad u, i \\ v \longrightarrow u : &\quad v, MAC(K_v^i, u|v) \end{aligned}$$

Pairwise Key Establishment. Node u computes its pairwise key with v, K_{uv}, as $K_{uv} = f_{K_v^i}(u)$. Node v can also compute K_{uv} in the same way.

Key Erasure. When its timer expires after time T_{min}, node u erases K_{IN}^i and all the master keys (K_v^i) of its neighbors, which it computed in the neighbor discovery phase. Node u does not erase its own master key K_u^i or any other preloaded master keys K_u^j where $i < j \leq M$.

For example, a node u that is deployed in the first time interval T_1 is loaded with K_{IN}^1 and K_u^j for all j, where $1 < j \leq M$. It derives its master key K_u^1 and uses K_{IN}^1 as the initial network key to establish pairwise keys with its neighbors, and then erases K_{IN}^1 after T_{min}. Note that the network controller or the mission authority should also erase the initial key K_{IN}^1, since it is no longer needed. When adding a sensor node v during time interval T_2, the network controller loads node v with initial key K_{IN}^2, from which the node derives its master key K_v^2, and $m-2$ master keys $K_v^3, ..., K_v^M$. Node v uses K_{IN}^2 to establish pairwise keys with the neighboring nodes that were deployed in T_1 or T_2, because v can derive the master keys of its neighbors for the interval T_2. When its timer expires, node v erases K_{IN}^2.

Security Analysis In this scheme, an attacker compromising a new node u deployed in T_i within T_{min} obtains K_{IN}^i and $M-i+1$ master keys of node u. The attacker cannot know any pairwise keys that the other nodes established in previous time intervals because node u does not know any initial keys K_{IN}^j $(1 \leq j < i)$ or the master keys of any other nodes for time intervals before T_i. Therefore, this scheme provides *weak backward key confidentiality.* Furthermore, the attacker cannot derive any initial keys K_{IN}^j $(i < j \leq M)$ corresponding to future time intervals either, so it cannot know any pairwise keys that other nodes will establish in the following time intervals. Hence, this scheme also provides *weak forward key confidentiality.*

We note if an attacker can compromise a sensor node within T_{min}, it can launch a *node addition attack* in which it introduces new nodes into the network by loading them with correct master keys. These new nodes can then be used to launch a variety of attacks that defeat the mission of the sensor network. A simple way to address this attack is that the network controller broadcasts the ids of the new nodes after T_{min}, using for example μTESLA [Perrig et al. 2001] for broadcast source authentication. A node checks the id of each new neighbor (if any) to see if the neighbor is legitimate. If not, it will discard its pairwise key shared with the claimed neighbor. This approach however has two potential problems. First, an attacker may easily guess new node ids if for example node ids are always linearly increasing. Second, a large number of ids (packets) may have to be broadcast if many nodes are to be added, consuming much energy. We adopt the following simple approach to solve the problem. Suppose that the network controller wants to add N_i nodes into a network at the time interval T_I^i. It generates N_i ids for these nodes based on a random seed s_i and a pseudo-random number generator (e.g., RC5 [Rivest 1994]); each of the N_i nodes is loaded with one unique id. Immediately after it is deployed, each node establishes a pairwise key with each neighbor. Time T_{min} later, the network controller broadcasts N_i and s_i in the network. A node deployed earlier thus is able to verify if the id of a new neighbor is one of the ids that are derived from s_i and N_i. We can see that if the size of a node id is large enough, it is very hard for an attacker to forge valid ids.

3.3.3 *Comparison with Other Pairwise Key Establishment Schemes.* Next we compare our basic scheme with several other pairwise key establishment schemes based on security and performance. The results are shown in Table I. The KDC scheme [Perrig et al. 2001] is a Kerberos-like scheme in which the base station helps two nodes to establish a pairwise key. In addition to MAC computations, the

Table I. Comparison with Other Schemes (§ means after deployment)

	Performance			Security	
Scheme	Computation	Communication(hops)	Pre-storage (keys)	Deterministic	Clone Prevention
KDC	MACs+Enc	≥ 2	One	Yes	No
PKS-Key	MACs+Enc	> 1	hundreds	No	No
PKS-MP	Modular+MACs+Enc	> 1	hundreds	No	No
LEAP+ (basic)	MACs	1	One	Yes	Yes§

KDC scheme also involves encryption/decryption because the base station needs to encrypt pairwise keys for transmission. An encrypted key is transmitted with at least two hops (two hops when two nodes are direct neighbors of the base station). The KDC scheme provides deterministic security in the sense that a node cannot know a pairwise key shared between two other nodes, although the base station knows all the pairwise keys in the system and hence is a single point of failure from the standpoint of security. Moreover, the KDC scheme cannot prevent the node cloning attacks unless the base station knows the topology of the network and hence only help those neighboring nodes establish pairwise keys.

There are two types of probabilistic key sharing schemes: those based on preloaded keys [Chan et al. 2003; Eschenauer and Gligor 2002; Zhu et al. 2003] and those based on preloaded matrices or polynomials [Du et al. 2003; Liu and Ning 2003a]. We denote them as PKS-Key and PKS-MP, respectively. In these schemes, with some probability, a node can establish a pairwise key with other nodes directly. If two nodes have to resort to a third node that might be one or multiple hops away for helping establishing a pairwise key, larger computational (e.g., encryptions) and communication overhead will be incurred. On the other hand, except for the random pairwise scheme in [Chan et al. 2003], all the these schemes only provide probabilistic security in the sense that a coalition of attackers can learn the pairwise keys shared between other nodes with some probability.

The table shows that overall our scheme has lower performance overhead than other schemes. Second, our scheme provides deterministic (as opposed to probabilistic) security. Third, unlike the other schemes, our scheme can also prevent node cloning attacks after node deployment (we shall show how it defends against several other sophisticated attacks in Section 4.2). Furthermore, although not shown in the table, our scheme scales up to arbitrarily large sensor networks whereas in both the PKS-Key and PKS-MP schemes there is an upper bound on the supported network size [Chan et al. 2003; Liu and Ning 2003a]. We note, however, that in our scheme there is a small time window (T_{min}) during which a newly deployed node is vulnerable to compromise, although our extended scheme helps mitigate the attack. We show in Section 5.4 that this window is of the order of several seconds; consequently, we believe this will not be a limitation in practice for most applications. Finally, we note that our pairwise key establishment protocol assumes that nodes are stationary, and is therefore particularly suitable for large-scale static sensor networks. The other schemes do not have this restriction and may be applied to mobile ad hoc networks as well.

3.4 Establishing Cluster Keys

Every node shares a unique key, called cluster key, with all its neighbors for encrypting its broadcast messages. Cluster key establishment follows the pairwise key establishment phase, and the process is very straightforward. Consider the case that node u wants to establish a cluster key with all its immediate neighbors $v_1, v_2, ..., v_m$. Node u first generates a random key K^c_u, then encrypts this key with the pairwise key shared with each neighbor, and then transmits the encrypted key to each neighbor v_i $1 \leq i \leq m$.

$$u \longrightarrow v_i : (K^c_u)_{K_{uv_i}}. \qquad (1)$$

Each node v_i decrypts the key K^c_u, stores it in a table, and then sends its own cluster key to node u, encrypted with their pairwise key. When node u is revoked, every neighbor node generates a new cluster key and transmits it to all the other neighbors in the same way.

3.5 Establishing the Global Key

A global key is a key shared by all the nodes in the network, and it is necessary when the controller distributes a confidential message, e.g., a query for some event of interest or an instruction, to all the nodes in the network.

A simple way to bootstrap a global key for a sensor network is to pre-load every node with the global key. An important issue that arises immediately is the need to securely update this key when a compromised node is detected. In other words, the global key must be changed and distributed to all the remaining nodes in a secure, reliable, and timely fashion. Note that this is equivalent to a *group rekeying* operation with the all the nodes in the network belonging to the group.

Unicast-based group rekeying, in which the key server sends the new group key to every individual node, has the communication complexity of $O(N)$ keys, where N is the group size. Recently proposed group rekeying schemes [Perrig et al. 2001; Wong et al. 1998] use logical key trees to reduce the complexity of a group rekeying operation from $O(N)$ to $O(\log N)$. Note that in all these schemes, the key server includes keys for *all* the member nodes when distributing its rekeying message, and every member receives the entire message although it is only interested in a small fraction of the content. A recent effort [Lazos and Poovendran 2003] attempts to reduce the unnecessary keys a node has to receive by mapping the physical locations of the nodes to the logical key tree in LKH [Wong et al. 1998]. This scheme however has only marginal improvement over the original LKH scheme. Moreover, for this scheme to work, the key server requires a global knowledge of the network topology.

Below we propose an efficient scheme for rekeying the global key based on cluster keys. We first discuss *authenticated node revocation*, which is a prerequisite for the rekeying operation, then describe the *secure key distribution* mechanism in detail.

3.5.1 *Authenticated Node Revocation.* In a sensor network, all the messages broadcast by the controller must be authenticated; otherwise, an outsider adversary may impersonate the controller. Therefore, a node revocation announcement must be authenticated during its distribution.

We employ the μTESLA [Perrig et al. 2001] protocol for broadcast authentication because of its efficiency and tolerance to packet loss. Let u be the node to be revoked, and k'_g the new global key. Let the to-be-disclosed μTESLA key be k_i^T. The controller broadcasts the following message M:

$$M : Controller \longrightarrow * : u, f_{k'_g}(0), MAC(k_i^T, u|f_{k'_g}(0)). \tag{2}$$

Here we refer to $f_{k'_g}(0)$ as the *verification* key because it enables a node to verify the authenticity of the global key k'_g that it will receive later. The controller then distributes the MAC key k_i^T after one μTESLA interval. After a node v receives message M and the MAC key that arrives one μTESLA interval later, it verifies the authenticity of M using μTESLA. If the verification is successful, node v will store the verification key $f_{k'_g}(0)$ temporarily. Finally, if a node v is a neighbor of node u, v will remove its pairwise key shared with u and update its cluster key.

3.5.2 *Secure Key Distribution.* Secure global key distribution does not require the use of a specific routing protocol. However, for concreteness, in this paper we assume the use of a routing protocol similar to the TinyOS beaconing protocol [Hill et al. 2000; Karlof and Wagner 2003]. In this protocol, the nodes in the network are organized into a breadth first spanning tree on the basis of routing updates that are periodically broadcast by the base station and recursively propagated to the rest of the network. The new global key k'_g is distributed to all the legitimate sensor nodes via a recursive process over the spanning tree. The base station (controller) initiates the process by sending k'_g to each of its children in the spanning tree using its cluster key for encryption. A node v that receives k'_g can verify the authenticity of k'_g immediately by computing and checking if $f_{k'_g}(0)$ is the same as the *verification key* it received earlier in the node revocation message. The algorithm continues recursively down the spanning tree, i.e., each node v that has received k'_g transmits k'_g to its children in the spanning tree, using its own cluster key for encryption.

Although hop-by-hop translation of encrypted broadcast messages involves non-trivial computational overhead for sensor nodes, it is not an issue for broadcasting a global key because only one key needs to be translated and rekeying is a relatively infrequent event. We note that our scheme for securely distributing the new global key is similar to the Iolus scheme for secure group communication [Mittra 1997].

Finally, we note that it is desirable that the global key be updated periodically even when no node revocation events occur. This is important for defending against cryptanalysis and to prevent an adversary from decrypting all the previously broadcast messages after compromising a sensor node. In our scheme, the controller can periodically broadcast an authenticated key update instruction. Alternatively, if the nodes in the network are loosely time synchronized, at fixed time intervals each node can update the global key by applying a one-way function to it. Specifically, each node can derive a new global key $K'_g = f_{K_g}(1)$ to replace the old key K_g.

3.6 Local Broadcast Authentication

For a secure sensor network, every message in the network must be authenticated before it is forwarded or processed; otherwise, an adversary could deplete the energy of the sensor nodes by injecting many spurious packets into the network. On the other hand, this requires that the authentication scheme be very lightweight; otherwise, a sensor node can be kept busy just verifying the injected false packets.

Previous work [Perrig et al. 2001] has studied unicast authentication (used when a node authenticates a packet to another node) and global broadcast authentication (used when the base station authenticates a packet to all the nodes in the network). Efficient local broadcast authentication however has not received much attention. We note that local broadcast is common operation in sensor networks, and that locally broadcast messages are usually event- or time-driven; for example, routing control messages or aggregated sensor readings. Typically a node will not know in advance the contents of the next packet that will be transmitted. This prevents us from directly applying μTESLA [Perrig et al. 2001] for local broadcast authentication as μTESLA uses delayed key disclosure. In Section 3.5 we employed μTESLA for providing global broadcast authentication of a node revocation message, assuming that the application can tolerate some revocation delay. Pairwise keys are not suitable for local broadcast authentication either, because a node would need to compute and attach m MACs to a locally broadcast message if it has m neighbors.

To support local broadcast authentication, a node needs to use a key that is known by all its neighbors so that the node only has to compute and attach one MAC to a message. The cluster key established in Section 3.4 meets this requirement. A cluster-key-based scheme, however, is subject to node impersonation attacks. If an adversary has compromised a sensor node, it can inject spurious packets into the network while impersonating a neighbor by using the cluster key of that neighbor for authenticating messages. Due to the symmetric nature of cluster keys, this impersonation attack cannot be defeated. Therefore, our goal is to design a scheme to thwart this attack to the extent possible. As such, unlike digital signature or μTESLA which provide strong broadcast source authentication, our scheme may be considered as a weak source authentication scheme.

The main idea in our approach is to use one-time authentication keys. More specifically, we propose to use one-way key chains [Lamport 1981] for weak one-hop broadcast authentication. Unlike μTESLA, this technique does not use delayed key disclosure and does not require time synchronization between neighboring nodes. Basically, every node generates a one-way key chain of certain length, and then transmits the commitment (i.e., the first key) of the key chain to each neighbor in an authenticated way. We refer to a key in a node's one-way key chain as its AUTH key. Whenever a node has a message to send, it attaches to the message the next AUTH key in the key chain. The AUTH keys are disclosed in an order reverse to their generation. A receiving neighbor can verify the message based on the commitment or an AUTH key it received from the sending node most recently.

The design of this authentication scheme was motivated by two observations. First, a node only needs to authenticate a packet (e.g., a routing control packet) to its *immediate* neighbors. Second, when a node sends a packet, a neighbor will *normally* receive the packet before it receives a copy forwarded by any other nodes. This is true due to the *triangle inequality* among the distances of the involved nodes, which is illustrated in Fig. 3(a). When node u sends a packet that contains the content M and an AUTH key K, node v will receive the packet before it receives a forwarded copy from node x because $|uv| < |ux| + |xv|$. Thus, once node v receives the message M, the adversary x cannot reuse the AUTH key K to inject another packet while impersonating node u.

The above authentication scheme provides weak source authentication while al-

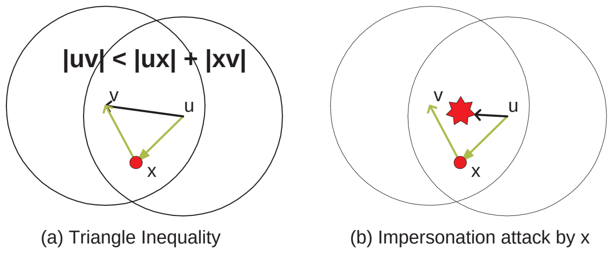


Fig. 3. Triangle inequality and impersonation attacks

lowing passive participation. However, we note that an attacker can overcome our scheme by preventing node v from receiving the packet from u directly. For example, as shown in Fig. 3(b), the adversary might shield node v or jam node v by letting another node w transmit to v at the same time when node u is transmitting. Later the adversary sends a modified packet to node v while impersonating node u. Because node v has not received a packet with the same AUTH key, it will accept the modified packet.

We can prevent an outsider attacker from launching the above attack by encrypting AUTH keys with cluster keys. Without knowing node u's cluster key, the outsider attacker is unable to impersonate node u. Unfortunately, we do not have a lightweight countermeasure to prevent the attack by an insider attacker that knows node u's cluster key. Nevertheless, we note that (i) the maximum number of false packets that a compromised node x can inject into the network, while impersonating node u, is bounded by the number of packets node u has transmitted, due to the one-wayness property of hash functions (ii) the compromise of a sensor node x only allows the adversary to launch such attacks in a two-hop zone centered at node x, because node x only has the cluster keys of its one-hop neighbors.

4. SECURITY ANALYSIS

In this section, we analyze the security of the keying mechanisms in LEAP+. We first discuss the survivability of the network when undetected compromises occur, and then study the robustness of our scheme in defending against various attacks on routing protocols.

4.1 Survivability

When a sensor node u is compromised, the adversary can launch attacks by utilizing node u's keying materials. If the compromise event is detected, our rekeying scheme can efficiently revoke node u from the network. After the revocation, the adversary cannot launch further attacks.

However, compromise detection in sensor systems is more difficult than in other systems because sensor systems are often deployed in unattended environments. Thus, we believe *survivability* in the face of *undetected* node compromises is one of the most critical security requirements for sensor networks. Below we first consider in general what the adversary can accomplish after it has compromised a sensor node. We then discuss some detailed attacks on routing protocols in Section 4.2.

First, if every node reports its readings to the base station directly using its individual key, obtaining the individual key allows the compromised node to inject a false sensor reading. Second, possessing the pairwise keys and cluster keys of a compromised node allows the adversary to establish trust with all the neighboring nodes. Thus the adversary can inject some malicious routing control information or erroneous sensor readings into the network. However, in our scheme the adversary usually has to launch such attacks by using the identity of the compromised node due to our one-time key based local broadcast authentication scheme. We note a salient feature of our protocol is its ability in *localizing* the possible damage, because after the network deployment, every node keeps a list of trusted neighboring nodes. Thus, a compromised node cannot establish trust relationships with any nodes other than its neighbors, nor can it jeopardize the secure links between other nodes.

Third, possessing the global key allows the adversary to decrypt the messages broadcast by the base station. Since a broadcast message, by its nature, is intended to be known by every node, compromising one single node is enough to reveal the message, irrespective of the security mechanisms used for securing message distribution. Moreover, possessing the global key does *not* enable the adversary to flood the entire network with malicious packets while impersonating the base station, because any messages sent by the base station are authenticated using μTESLA. Finally, because the global key is updated periodically, the adversary can decrypt only the messages being encrypted using the current global key.

4.2 Defending against Various Attacks on Secure Routing

Karlof and Wagner [Karlof and Wagner 2003] have studied various attacks on the security of routing protocols for wireless sensor network. We now discuss how our schemes can defend against the attacks they described. In LEAP+ routing control information is authenticated by the local broadcast authentication scheme, which prevents most outsider attacks (with the exception of the wormhole attack which we discuss in Section 4.2.1). Therefore, in the discussion below we mainly consider attacks launched by an *insider* adversary that has compromised one or more sensor nodes.

An insider adversary may attempt to *spoof, alter or replay* routing information, in the hope of creating routing loops, attracting or repelling network traffic, or generating error messages. The adversary may also launch the *selective forwarding* attack in which a compromised node suppresses the routing packets originating from a select few nodes while reliably forwarding the remaining packets. Our scheme cannot prevent the adversary from launching these attacks. However, our scheme can thwart or minimize the consequences of these attacks. First, our local broadcast authentication scheme makes these attacks only possible within a one-hop zone of the compromised node. Second, because the attacks are localized in such a small zone, the adversary takes a high risk of being detected in launching these attacks. The altering attack is also likely to be detected because the sending node may overhear its message being altered while being forwarded by the compromised node. Third, once a compromised node is detected, our rekeying scheme can revoke the node from the network very efficiently.

Our scheme can largely prevent the following attacks. The adversary may try to launch a *HELLO Flood Attack* in which it sends a HELLO message to all the nodes with transmission power high enough to convince all the nodes that it is their neighbor. It has been shown that many routing protocols (e.g., TinyOS Beaconing, directed diffusion [Intanagonwiwat et al. 2000], LEACH [Heinzelman et al. 2000], SPAN [Chen et al. 2002].) are subject to HELLO flood attacks. If this attack succeeds, all the nodes may send their readings or other packets into oblivion. However, this attack will not succeed in LEAP+ beyond the neighbor discovery phase because every node only accepts packets from its authenticated neighbors. As we discussed earlier, our scheme can also prevent the *node cloning attacks* beyond the neighbor discovery phase.

4.2.1 *Dealing with Wormhole and Sinkhole Attacks.* The attack that is most difficult to detect or prevent is one that combines the *sinkhole* and *wormhole* attacks. In a *sinkhole* attack, a compromised node may try to attract packets (e.g., sensor readings) from its neighbors and then drop them, by advertising information such as high remaining energy or high end-to-end reliability. This information is hard to verify. In a *wormhole* attack, typically two distant malicious nodes, which have an out-of-band low latency link that is invisible to the underlying sensor network, forward to each other the packets overheard from their own neighbors. By placing one such node close to the base station and the other close to the target of interest, the adversary could convince the nodes near the target who would normally be multiple hops away from the base station that they are only one or two hops away, thereby creating a *sinkhole*. Similarly, nodes that are multiple hops away from each other may believe they are neighbors due to the wormhole. The wormhole attack is very powerful because the adversary does not have to compromise any sensor nodes to be able to launch it. In the literature, Hu, Perrig, and Johnson [Hu et al. 2003] propose two schemes to detect wormhole attacks for ad hoc networks. The first scheme requires every node to know its geographic coordinate (using GPS). The second scheme requires an extremely tight time synchronization between nodes and is thus infeasible for most sensor networks. More importantly, these schemes can only mitigate such attacks against two honest nodes; they do not prevent a malicious node from deceiving another node by, for example, lying about its coordinates.

In LEAP+, an outsider adversary cannot succeed in launching wormhole attacks at any time other than during the *neighbor discovery* phase of the pairwise key establishment process. After that phase, a node knows all its neighbors. Thus the adversary cannot later convince two distant nodes that they are neighbors. Since the time for neighbor discovery is very small (of the order of seconds) compared to the lifetime of the network, the probability that the adversary succeeds in such attacks will also be very small. We note that *authenticated* neighborhood knowledge is critical to defend against wormhole attacks.

An insider adversary needs to compromise at least two sensor nodes to create a wormhole in LEAP+. Even so, it still cannot convince two distant nodes that they are neighbors after they have completed their *neighbor discovery* phase. However, if the adversary compromises one node u that is located close to the base station with the other node v in the area of interest, it may succeed in making node v a *sinkhole*. This is because the number of hops between the node v and the base station becomes smaller, making node v especially attractive to surrounding nodes. In applications where the location of a base station is static, a node will know the approximate number of hops to the base station after the network topology is constructed. Thus it is difficult for the adversary to create a very attractive sinkhole without being detected.

5. PERFORMANCE EVALUATION

We only consider the overhead for updating cluster keys and the global key in the case of a node revocation. The performance of our pairwise key establishment has been analyzed in Section 3.3. We assume that the rekeying protocol uses a spanning tree for delivering the new global key to the nodes in the system.

5.1 Computational Cost

While updating a cluster key, a node that is a neighbor of the node being revoked needs to encrypt its new cluster key using the pairwise key shared with each neighbor. Therefore, the number of such encryptions is determined by the number of neighbors, which depends on the density of the sensor network. Let s be the number of neighbors of the node being revoked, and d_i, $1 \leq i \leq s$ the number of legitimate neighbors of each of these s neighbors. The total number of encryptions is simply $X_e = \sum_{i=1}^{s} d_i$. The total number of decryptions is the same, although the number of decryptions for an individual node that is a neighbor to any of these s nodes depends on its location. In the worst case where a node is a neighbor to all these s nodes, it needs to decrypt s keys. For an individual node, the total number of symmetric key operations it performs is bounded by $\max_i(d_i) + s - 1$. For a network of size N (including the node being revoked), the average number of symmetric key operations a node performs while updating a cluster key is $\frac{2X_e}{N-1} = 2\sum_{i=1}^{s} d_i/(N-1)$.

During the secure distribution of a global key, the number of decryptions is $N-1$ because every node needs to decrypt one key. Recall that we use cluster keys for secure forwarding of the global key, which means a parent node only needs to encrypt once for all its children. Thus the total number of encryptions depends on the network topology and is at most $N-1$. Therefore, the total number of symmetric key operations is at most $2(N-1)$ and the amortized cost is at most 2 symmetric key operations per node.

The above analysis shows that the computational cost of a node revocation operation is determined by the network density. In a network of size N where every node has a connection degree d, the average number of symmetric key operations for every node is approximately $2\sum_{i=1}^{d-1}(d-1)/(N-1) + 2 = 2(d-1)^2/(N-1) + 2$ when summing up all the computational costs. For a network of reasonable density, we believe that computational overhead will not become a performance bottleneck in our schemes. For example, for a network of size $N = 1000$ and connection degree 20, the average computational cost is 2.7 symmetric key operations per node per revocation. This cost becomes smaller for a larger N.

5.2 Communication Cost

The analysis of communication cost for rekeying the global key is similar to that of the computational cost. For updating cluster keys due to a node revocation, the average number of keys a node transmits and receives is equal to $(d-1)^2/(N-1)$ for a network of degree d and size N. During the secure distribution of a global key, the average number of keys a node transmits and receives is equal to one. For example, for a network of size $N = 1000$ and connection degree $d = 20$, the average transmission and receiving costs are both 1.4 keys per node per revocation.

The average communication cost increases with the connection degree of a sensor network, but decrease with the network size N. Note that in a group rekeying scheme based on logical key tree such as LKH [Wong et al. 1998], the communication cost of a group rekeying is $O(logN)$. Thus, our scheme is more scalable than LKH if LKH were to be used for rekeying the global key.

5.3 Storage Requirements

In our schemes, a node needs to keep four types of keys. If a node has d neighbors, it needs to store one individual key, d pairwise keys, d cluster keys, and one global key. In addition, in our local broadcast authentication scheme, a node also keeps its own one-way key chain as well as the commitment or the most recent AUTH key of each neighbor. In a sensor network the packet transmission rate is usually low. For example, the readings may be generated and forwarded periodically, and the routing control information may be exchanged less often. Thus, a node could store a key chain of a reasonable length. After the keys in the key chain are used up, it can generate and bootstrap a new key chain. On the other hand, if network traffic load is heavy, it is better to use a long key chain. To avoid storing the entire key chain of size n, we can employ Coppersmith and Jakobsson's optimization algorithm [Coppersmith and Jakobsson 2002] that computes $O(\log_2 \sqrt{n})$ hashes for each AUTH key and uses $O(\log_2 n)$ keys. (We note that Hu et al. [Hu et al. 2005] have proposed an algorithm that further reduces the computation cost of key chain traversal to $O(1)$ but at the expense of additional transmissions.) Let L be the number of keys a node stores for its key chain. Thus, the total number of keys a node stores is $3d + 2 + L$. For example, when $d = 20$ and $L = 30$, a node stores 92 keys taking up 736 bytes if the key size is 8 bytes.

Overall, we conclude our scheme is scalable and efficient in terms of its computational, communication, and storage requirements.

5.4 The Prototype Implementation of LEAP+

We have implemented a prototype of LEAP+ on the TinyOS platform [Hill et al. 2000]. Our goal was to evaluate the feasibility of implementing LEAP+ on the current generation of sensor nodes, and to evaluate the storage requirements for the protocol code, and to experimentally measure the time taken by a newly deployed node to establish pairwise and cluster keys with its neighbors. Our current implementation includes the protocols for pairwise key establishment, cluster key establishment, and one-way key chain generation. It does not include the scheme for rekeying the global key and the broadcast source authentication scheme based on μTESLA.

The programs were written in nesC, a C-like language for developing applications in TinyOS. LEAP+ uses several Timer interfaces (provided by a Timer component) for providing its key erasure time threshold and for handling message retransmission during pairwise key establishment phases. LEAP+ uses the RC5 block cipher [Rivest 1994] to provide both encryption and CBC-MAC. The pseudo random functions, which are used in deriving master keys and pairwise keys, and the one-way functions, which are used in constructing one-way key chains, are also replaced by MAC with RC5. That is, LEAP+ uses RC5 to provide all the security primitives. This code reuse saves code space in ROM as well as data space in RAM because fewer variables and cipher contexts have to be defined.

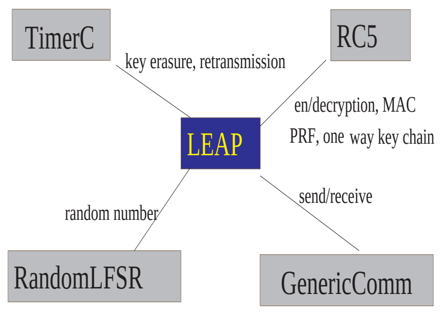


Fig. 4. The components and the interfaces of TinyOS used in the implementation of LEAP+

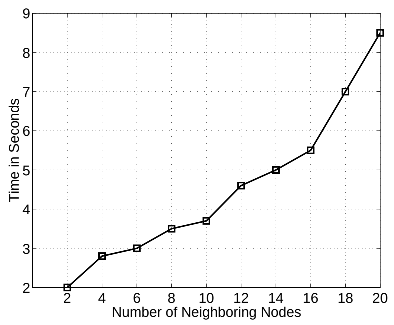


Fig. 5. Time for a sensor node to establish pairwise keys with all of its neighbors

In LEAP+, a newly added node needs to exchange one message with every neighbor node to establish a pairwise key and exchange another message to establish a cluster key. Clearly, reliable transmission mechanisms are needed to ensure that a new node establishes keys with the majority of its neighbors, if not all. Providing reliability is a non-trivial task for sensor networks. Recall that in the neighbor discovery phase every neighbor sends back an ACK message to a new node after receiving a HELLO message from the new node. This leads to feedback implosion, especially when the network node density d is high (e.g., $d > 20$ neighbors). As a result, a new node is likely to miss one or more of the ACKS due to packet collisions. We note that although the media access control component (ChannelMonM) in TinyOS has a random backoff mechanism to address the packet collision issue, the parameters for the random function are fixed and do not adapt to network node density d. Our experiments shows that a node often misses at least one ACK once d grows larger than 5. This indicates that the random backoff mechanism does not scale with d when feedback implosion occurs. To address this issue, LEAP+ adds a random backoff mechanism of its own. That is, after a node receives a HELLO message, it waits for a random time in the interval $(0, T_1)$ before sending its ACK message, where T_1 is chosen based on d. This greatly reduces the probability of packet collision.

We added the following mechanisms in the implementation of LEAP+ to further increase the probability of two nodes successfully establishing pairwise/cluster keys. First, a new node broadcasts its first HELLO message at a random time in the interval $(0, T_2)$ after it is deployed and it broadcasts the HELLO message multiple times. For example, we can set $T_2 = 3$ seconds, and the HELLO broadcast can be repeated 3 times with an interval of 3 seconds if $T_{min} > 12$ seconds. Second, we integrate the pairwise key establishment phase with the cluster establishment phase by using a three-way handshake process. Assuming that node u is a new node, and v is a neighbor node. The messages exchanged in the pairwise/cluster key establishment phases are rewritten as follows.

$$u \longrightarrow * : \quad u.$$

Table II. The required RAM space as a function of the number of neighbors d

d	1	5	10	15	20	25	30
RAM (bytes)	600	736	906	1076	1246	1416	1586

$$
\begin{aligned}
v \longrightarrow u: &\quad v, \{K_v^c\}_{K_v}, MAC(K_v, u|v|\{K_v^c\}_{K_v}).\\
u \longrightarrow v: &\quad u, \{K_u^c\}_{K_{uv}}, MAC(K_v, u|\{K_u^c\}_{K_{uv}}).
\end{aligned}
$$

Here node v includes its cluster key, encrypted with its master key, in its ACK message to avoid sending a separate message for establishing a cluster key. Upon receiving an ACK message from v, node u immediately replies with an ACK2 message, which includes its cluster key, encrypted with K_{uv}. The advantage of this integration is that both nodes u and v can infer if their messages are lost and schedule retransmissions if necessary. For example, if node v does not receive an ACK2 message from node u within a threshold time, it retransmits its ACK message. If node u receives a retransmitted ACK message, it infers that its ACK2 message is lost, so it can retransmit it. These mechanisms help a new node establish keys with all its neighbors.

On Mica2 Motes [xbow 2005], our code occupies 17.9KB (out of 128 KB available) of the program memory. The memory used for data depends on the number of neighbors d since this determines the number of pairwise keys and cluster keys that need to be stored. Table II shows the usage of RAM as a function of d. Here the size of a key is 8 bytes; the length of a key chain is 20 keys. We can see that the memory requirements for LEAP+ are reasonable given that the Mica2 motes have 4 KB of RAM available for data storage.

One parameter of particular interest for our pairwise key establishment scheme is the threshold T_{min}, which determines the earliest moment a node could erase the preloaded initial network key K_{IN} without missing a neighbor. Recall that in our scheme a newly added node only needs to know the id of a neighbor to establish their pairwise key; therefore, T_{min} could be the time the new node receives ACKs from all neighbors after it broadcasts a Hello message. Once a new node receives an ACK from a neighbor, it can verify the ACK and derive their pairwise key immediately. After it receives the ACKs from all its neighbors, it can safely erase K_{IN}. The ACK2 message only requires a pairwise key, which the node has already computed. In our experiments, however, we consider T_{min} as the time for a node to finish the three-way handshake described above with all neighbors. Figure 5 shows that a node can complete the three-way handshake with 20 neighbors in about 8.5 seconds. Therefore, we consider $T_{min} = 10$ seconds to be a good threshold in practice for a network of moderate density (e.g., $d = 20$). We further note that these results are implementation dependent. For example, the setting of retransmission timers and the number of packet buffers can affect the results.

Finally, we should mention that packet loss and feedback implosion are not unique issues to LEAP+. Most security protocols for sensor networks face the same issues. Therefore, although the techniques for addressing packet losses are discussed in the context of LEAP+, we believe that our experience could benefit the implementation of other protocols [Du et al. 2003; Eschenauer and Gligor 2002; Liu and Ning 2003a].

6. RELATED WORK

Carman, Kruus and Matt have analyzed several approaches for key management and distribution in sensor networks [Carman et al. 2000]. In particular, they discuss the energy consumption of three different approaches for key establishment – pre-deployed keying protocols, arbitrated protocols involving a trusted server, and autonomous key agreement protocols. Basagni *et al* [Basagni et al. 2001] discussed a cluster-based rekeying scheme for periodically updating the group-wide traffic encryption key in a sensor network. However, they assume that sensor nodes are tamper-proof and can trust each other. In contrast, our pairwise key establishment scheme only requires that a sensor node not be compromised for a short time interval at the time of its deployment.

Anderson et al.[Anderson et al. 2004] described an interesting scheme that can be used by sensor nodes to establish pairwise keys in environments with a partially present, passive adversary. A node wishing to communicate securely with other nodes simply generates a symmetric key and sends it in the clear to its neighbors. Mechanisms including multipath secrecy amplification and multihop key propagation are further used to enhance the strength of the pairwise keys. These mechanisms may also enhance our scheme when the initial network key could be broken. Indeed, also based on the assumption of partially present, passive adversary, Deng et al. [Deng et al. 2005] improved our pairwise key establishment scheme by using the initial key to securely exchange pairwise keys that are randomly generated on the fly instead of deriving pairwise keys from the initial key. Thus, even if the adversary can compromise the initial key, he may not know all the pairwise keys.

Eschenauer and Gligor [Eschenauer and Gligor 2002] presented a key management scheme for sensor networks based on probabilistic key predeployment. Chan et al. [Chan et al. 2003] extended this scheme and present three mechanisms for key establishment. Zhu et al. [Zhu et al. 2003] presented an approach for establishing a pairwise key based on the combination of probabilistic key sharing and threshold secret sharing. Using peer trusted intermediaries and a virtual ID space, Chan and Perrig [Chan and Perrig 2005] proposed the PIKE protocol, which has been shown to greatly increase the scalability of pairwise key establishment as well as the security of pairwise keys. Du et al. [Du et al. 2003], Liu and Ning [Liu and Ning 2003a] combined the technique of probabilistic key predeployment with the Blom's [Blom 1985] or the Blundo's [Blundo et al. 1993] key management schemes for establishing pairwise keys in sensor networks. In addition, deployment knowledge has been explored for pairwise key establishment [Du et al. 2004; Liu and Ning 2003b]. We have compared in detail both the performance and the security of our pairwise key establishment scheme with some of these schemes in Section 3.3. Note that our key management framework does not specifically rely on our own pairwise key establishment scheme; indeed, the other pairwise key establishment schemes can be substituted for our scheme in the framework if necessary.

Perrig et al. presented security protocols for sensor networks [Perrig et al. 2001]. In particular, they describe SNEP, a protocol for data confidentiality and two-party data authentication, and μTESLA, a protocol for broadcast data authentication. We note that their scheme uses the base station to help establish a pairwise key between two nodes, which limits its scalability and leaves it subject to node cloning attacks. In contrast, in our scheme pairwise keys are established in a distributed fashion without the involvement of the base station.

Karlof et al. [Karlof et al. 2004] described TinySec, a link layer security mechanism using a single preloaded fixed global key for both encryption and authentication, assuming no node compromises. They also discuss the impact of different keying mechanisms on the effectiveness of in-network processing in sensor networks. Deng et al. [Deng et al. 2003] discussed several security mechanisms for supporting in-network processing in hierarchical sensor networks.

Some of the other related works on sensor network security include secure data aggregation [Hu and Evans 2003; Przydatek et al. 2003; Yang et al. 2006], preventing false data injection attacks [Ye et al. 2004; Zhu et al. 2004], detecting node replication attacks [Parno et al. 2005] and Sybil attacks [Newsome et al. 2004], tolerating mobile sink compromises [Zhang et al. 2005], and broadcast source authentication [Liu et al. 2005; Perrig et al. 2001].

7. CONCLUSIONS

We have presented LEAP+ (Localized Encryption and Authentication Protocol), a key management protocol for sensor networks. LEAP+ has the following properties:

—LEAP+ includes support for establishing four types of keys per sensor node – individual keys shared with the base station, pairwise keys shared with individual neighboring nodes, cluster keys shared with a set of neighbors, and a global key shared by all the nodes in the network. These keys can be used to increase the security of many protocols.

—LEAP+ includes an efficient protocol for weak local broadcast authentication based on the use of one-way key chains.

—A distinguishing feature of LEAP+ is that its key sharing approach supports in-network processing, while restricting the security impact of a node compromise to the immediate network neighborhood of the compromised node.

—LEAP+ can prevent or increase the difficulty of launching many security attacks on sensor networks.

—The key establishment and key updating procedures used by LEAP+ are efficient and the storage requirements per node are small.

We have implemented a prototype of LEAP+ on the TinyOS platform. Our experience indicates that LEAP+ is a practical and resource-efficient protocol that can be used for securing a wide variety of sensor network applications and protocols. Our future work includes implementing the full functionality of LEAP+ and contributing the source code to the TinyOS community.

Acknowledgment The authors are grateful to our shepherd Adrian Perrig for his effort and suggestions in improving this paper. We would also like to thank the anonymous reviewers for their valuable comments. This work was supported by US Army Research Office (ARO) under grant W911NF-05-1-0270, and by National Science Foundation (NSF) under grant NSF-0524156.

REFERENCES

2005. Crossbow technology inc. Url: http://www.xbow.com.

Anderson, R., Chan, H., and Perrig, A. 2004. Key infection: Smart trust for smart dust. In *Proceedings of IEEE International Conference on Network Protocols (ICNP 2004)*.

Anderson, R. and Kuhn, M. 1996. Tamper resistance – a cautionary note. In *Proceedings of The Second USENIX Workshop on Electronic Commerce'96*. 1–11.

Basagni, S., Herrin, K., Rosti, E., and Bruschi, D. 2001. Secure pebblenets. In *Proceedings of ACM International Symposium on Mobile ad hoc networking and computing (MobiHoc'01)*. 156–163.

Blom, R. 1985. An optimal class of symmetric key generation systems. In *Advances in Cryptology, Proceedings of EUROCRYPT'84*. LNCS 209. 335–338.

Blundo, C., Santis, A. D., Herzberg, A., Kutten, S., Vaccaro, U., and Yung, M. 1993. Perfectly-secure key distribution for dynamic conferences. In *Advances in Cryptology, Proceedings of CRYPTO'92*. LNCS 740. 471–486.

Carman, D., Kruus, P., and Matt, B. 2000. Constraints and approaches for distributed sensor network security. Tech. rep., NAI Labs, Technical Report No. 00010.

Chan, H. and Perrig, A. 2005. PIKE: Peer intermediaries for key establishment in sensor networks. In *Proceedings of IEEE Infocom*.

Chan, H., Perrig, A., and Song, D. 2003. Random key predistribution schemes for sensor networks. In *Proceedings of IEEE Security and Privacy Symposim'03*.

Chen, B., Jamieson, K., Balakrishnan, H., and Morris, R. 2002. Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *ACM Wireless Networks Journal 8(5)*, 481–494.

Coppersmith, D. and Jakobsson, M. 2002. Almost optimal hash sequence traversal. In *Proceedings of Finanical Cryptography (FC) 02*. 102–119.

Deng, J., Han, R., and Mishra, S. 2003. Security support for in-network processing in wireless sensor networks. In *Proceedings of First ACM Workshop on the Security of Ad Hoc and Sensor Networks (SASN'03)*.

Deng, J., Hartung, C., Han, R., and Mishra, S. 2005. A practical study of transitory master key establishment for wireless sensor networks. In *Proceedings of First IEEE/CreateNet Conference on Security and Privacy for Emerging Areas in Communication Networks (SecureComm)*. 289–299.

Dierks, T. and Allen, C. 1999. The tls protocol version 1.0.

Du, W., Deng, J., Han, Y., and Chen, S. 2004. A key management scheme for wireless sensor networks using deployment knowledge. *Proceedings of IEEE INFOCOM'04*.

Du, W., Deng, J., Han, Y., and Varshney, P. 2003. A pairwise key pre-distribution scheme for wireless sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS'03)*. 42–51.

Eschenauer, L. and Gligor, V. 2002. A key-management scheme for distributed sensor networks. In *Proceedings of ACM CCS'02*.

Heinzelman, W., Chandrakasan, A., and Balakrishnan, H. 2000. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*. 3005C3014.

Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D. E., and Pister, K. S. J. 2000. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*. 93–104.

Hu, L. and Evans, D. 2003. Secure aggregation for wireless networks. In *Proceedings of Workshop on Security and Assurance in Ad hoc Networks*.

Hu, Y., Jakobsson, M., and Perrig, A. 2005. Efficient constructions for one-way hash chains. In *Proceeding of Applied Cryptography and Network Security (ACNS)*.

Hu, Y., Perrig, A., and Johnson, D. 2003. Packet leashes: A defense against wormhole attacks in wireless ad hoc networks. In *Proceedings of IEEE Infocom'03*.

INTANAGONWIWAT, C., GOVINDAN, R., AND ESTRIN, D. 2000. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of ACM Mobile Computing and Networking (Mobicom'00)*. 56–67.

KARLOF, C., LI, Y., AND POLASTRE, J. 2003. Arrive: An architecture for robust routing in volatile environments. Tech. rep., Technical Report UCB/CSD-03-1233, University of California at Berkeley.

KARLOF, C., SASTRY, N., AND WAGNER, D. 2004. Tinysec: A link layer security architecture for wireless sensor networks. In *Second ACM Conference on Embedded Networked Sensor Systems (SensSys 2004)*.

KARLOF, C. AND WAGNER, D. 2003. Secure routing in sensor networks: Attacks and countermeasures. In *Proceedings of First IEEE Workshop on Sensor Network Protocols and Applications*.

KOHL, J. AND NEUMAN, B. 1993. The kerberos network authentication service (v5), rfc 1510.

LAMPORT, L. 1981. Password authentication with insecure communication communication. *Communications of the ACM 24(11)*, 770–772.

LAZOS, L. AND POOVENDRAN, R. 2003. Energy-aware secure multicast communication in ad-hoc networks using geographic location information. In *Proceedings of IEEE ICASSP'03*.

LIU, D. AND NING, P. 2003a. Establishing pairwise keys in distributed sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS '03)*. 52–61.

LIU, D. AND NING, P. 2003b. Location-based pairwise key establishment for static sensor networks. *Proceeding of ACM Workshop on Security of Ad Hoc and Sensor Networks*.

LIU, D., NING, P., ZHU, S., AND JAJODIA, S. 2005. Practical broadcast authentication in sensor networks. In *Proceedings of International Conference on Mobile and Ubiquitous Systems (Mobiquitous'05)*.

MADDEN, S., SZEWCZYK, R., FRANKLIN, M., AND CULLER, D. 2002. Supporting aggregate queries over ad-hoc wireless sensor networks. In *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications*.

MITTRA, S. 1997. Iolus: A framework scalable secure multicast. In *Proceedings of ACM SIGCOMM'97*. 277–288.

NEWSOME, J., SHI, R., SONG, D., AND PERRIG, A. 2004. The sybil attack in sensor networks: Analysis and defenses. In *Proceedings of IEEE International Conference on Information Processing in Sensor Networks (IPSN 2004)*.

PARNO, B., PERRIG, A., AND GLIGOR, V. 2005. Distributed detection of node replication attacks in sensor networks. In *Proceedings of IEEE Symposium on Security and Privacy*.

PERRIG, A., SONG, D., AND TYGAR, D. 2001. Elk, a new protocol for efficient large-group key distribution. In *IEEE Symposium on Security and Privacy*.

PERRIG, A., SZEWCZYK, R., WEN, V., CULLER, D. E., AND TYGAR, J. D. 2001. Spins: security protocols for sensor netowrks. In *Proceedings of ACM Mobile Computing and Networking (Mobicom'01)*. 189–199.

PRZYDATEK, B., SONG, D., AND PERRIG, A. 2003. SIA: Secure information aggregation in sensor networks. In *Proceedings of ACM SenSys 2003*.

RIVEST, R. 1994. The rc5 encryption algorithm. In *Proceedings of the 1st International Workshop on Fast Software Encryption*. 86–96.

WONG, C. K., GOUDA, M., AND LAM, S. 1998. Secure group communication using key graphs. In *Proceedings of ACM SIGCOMM 1998*.

WOOD, A. AND STANKOVIC, J. 2002. Denial of service in sensor networks. *IEEE Computer*, 54–62.

YANG, Y., WANG, X., ZHU, S., AND CAO, G. 2006. Sdap: A secure hop-by-hop data aggregation protocol for sensor networks. In *Proceeding of The 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc)*.

YE, F., LUO, H., LU, S., AND ZHANG, L. 2004. Statistical en-route detection and filtering of injected false data in sensor networks. In *Proceedings of IEEE Infocom'04*.

Zhang, W., Song, H., Zhu, S., and Cao, G. 2005. Least privilege and privilege deprivation: Towards tolerating mobile sink compromises in wireless sensor networks. In *Proceedings of ACM Mobihoc*.

Zhu, S., Setia, S., and Jajodia, S. 2003. Leap: Efficient security mechanisms for large-scale distributed sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS '03)*. 62–72.

Zhu, S., Setia, S., Jajodia, S., and Ning, P. 2004. An interleaved hop-by-hop authentication scheme for filtering of injected false data in sensor networks. In *Proceedings of IEEE Symp. on Security and Privacy*. 259–271.

Zhu, S., Xu, S., Setia, S., and Jajodia, S. 2003. Establishing pair-wise keys for secure communication in ad hoc networks: A probabilistic approach. In *Proceedings of 11th IEEE International Conference on Network Protocols (ICNP'03)*.