LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks [*]

Sencun Zhu
Center for Secure Information Systems
George Mason University
Fairfax, VA 22030
szhu1@gmu.edu

Sanjeev Setia
Center for Secure Information Systems
George Mason University
Fairfax, VA 22030
setia@gmu.edu

Sushil Jajodia
Center for Secure Information Systems
George Mason University
Fairfax, VA 22030
jajodia@gmu.edu

ABSTRACT

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 group key that is shared by all the nodes in the network. The protocol used for establishing and updating these keys is communication- and energy-efficient, and minimizes the involvement of the base station. LEAP also includes an efficient protocol for 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 without precluding in-network processing. Our performance analysis shows that LEAP is very efficient in computation, communication, and storage. 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 attack, Sybil attack, and Wormhole attack. A prototype implementation of LEAP in a sensor network testbed is also reported.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

General Terms

Security,Algorithm,Design

Keywords

In-network Processing, Key Erasure, Key Management, Pairwise Key, Sensor Networks

[*]This is a technique report as of Aug. 2004. A conference version was appeared in CCS'03.

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 space (Mica2 Motes [10]). Thus, 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 constrained energy budgets and the limited computational and communication capacities of sensor nodes make protocols such as TLS [13] and Kerberos [23] 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 [5, 8], approaches in which every node shares a unique key with the base station [36], and approaches based on random key sharing [9, 15, 16, 28].

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 [24]. 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 [21] (see Figure. 1(a)). Passive participation (Figure. 1(b)) is another form of in-network processing in which a sensor node can take certain actions based on overheard messages [22, 30], 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. 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 of secure messages 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), 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 group 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 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 Section 4 and 5, we analyze the performance and security of the LEAP protocol. A prototype implementation of LEAP is reported in Section 6. We discuss related work in Section 7 before concluding the paper in Section 8.

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 the 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 [10]. We assume that every node has space for storing up to 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 will not be known in advance.

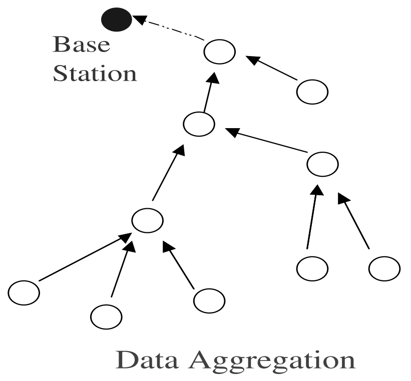
Because wireless communication is not secure, we assume an adversary can eavesdrop on all traffic, inject packets, or replay older messages. We assume that if a node is compromised, all the information it holds will be known to the attacker. However, we assume that the base station will not be compromised.

We assume that the physical layer of a wireless sensor network could use techniques such as spread spectrum [34] to prevent physical jamming attack if necessary. Techniques such as ALOHA and Slotted ALOHA [1] may be used to relieve attacks on the underlying Media Access Control protocol. In this paper, we do not address these attacks.

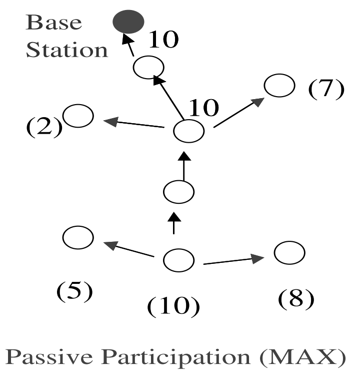
2.2 Design Goal

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 to meet several security and performance requirements that are considerably more challenging 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.



(a) Data aggregation: a sensor node aggregates the reported values from its children and forwards the aggregated value to its parent



(b) Passive Participation: a sensor node decides whether or not sending its own readings based on the overheard values

Figure 1: In-network Processing

- **Supporting In-network Processing** Security mechanisms should permit in-network processing operations such as data aggregation and passive participation. In-network processing could 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. Therefore, a sensor network should be robust against security attacks, and if an attack succeeds, its impact should be minimized. For example, the compromise a single sensor node should not break the security of the entire network.

- **Energy Efficiency** Due to the limited battery lifetime, security mechanisms for sensor networks must be energy efficient. Especially, the number of message transmissions and the number of expensive computations should be as few as possible. Moreover, the size of a sensor network should not be limited by the per-node storage and energy resources.

- **Avoiding Message Fragmentation** A unique challenge in sensor networks is due to small packet size. In TinyOS [20], the operating system for Mica series sensors [10], the default supported packet size is only 36 bytes for increasing the reliability of packet delivery. Thus, messages in a security protocol have to be small enough to fit in one packet to avoid message fragmentation. Message fragmentation is very undesirable for sensor networks because it increases the implementation complexity and difficulty. High packet loss in a sensor network and limited buffer space of a sensor node 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 [36].

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 group 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 instruction, that it sends to an individual node.

Group Key This is a globally shared key that is used by the base station for encrypting messages that are broadcast to the whole group. For example, the base station issues missions, sends queries and interests. 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 group 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 mainly used for securing locally broadcast messages, e.g., routing control information, or securing sensor messages which can benefit from passive participation. Researchers have shown that in-network processing techniques, including data aggregation and passive participation are very important for saving energy consumption in sensor networks [21, 22, 30]. For example, a node that overhears a neighboring sensor node transmitting the same reading as its own current reading can elect to not transmit the same. In responding to aggregation operations such as MAX, a node can also suppress its own reading if its reading is not larger than an overheard one. Clearly, for passive participation to be feasible, sensor nodes should be able to decrypt or verify some classes of messages, e.g., sensor readings, transmitted by their neighbors. This requires that such messages be encrypted or authenticated by a *locally shared* key. As such, 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.

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 individual keys, pairwise shared keys, cluster keys, and group keys for each node. The key establishment (and re-keying) protocol for the group 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 (in lower case) are principals such as communicating nodes.
- $\{f_k\}$ is a family of pseudo-random functions [17].
- $\{s\}_k$ 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 $K_0 = f_K(0)$ for encryption and use $K_1 = f_K(1)$ 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_0 or authenticated with K_1.

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 K_u^m for a node u (each node has a unique id) is generated as follows: $K_u^m = f_{K^m}(u)$. 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 K_u^m on the fly. Due to the computational efficiency of pseudo random functions, the computational overhead 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 always communicates with its immediate neighbors in any sensor network applications. Therefore, this type of pairwise keys is most commonly used. If a sensor network application has special requirements of establishing pairwise keys for two nodes that are multiple hops away, the multiple-path scheme [44] or a probabilistic key sharing scheme [15, 28, 47] may be employed.

For nodes whose neighbor relationship are predetermined, e.g., via physical installation, pairwise key establishment can be simply done by pre-loading the sensor nodes with the corresponding pairwise keys. Here we are interested in establishing pairwise keys for sensor nodes that are unaware of their neighbors until they have been deployed, e.g., 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. To this end, instead of assuming that sensor nodes are tamper resistant which often turns out not to be true [2], 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}. In practice, we expect T_{est} to be of the order of several seconds, so we believe 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 by 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 establishing a pairwise key with each of its neighbors that were deployed earlier.

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

Neighbor Discovery When it is deployed, node u first initializes a timer to fire after time T_{min}. It then tries to discover its neighbors. It broadcasts a HELLO message which contains its id, and waits for each neighbor v to respond with an ACK message including the identity of node v. The ACK from every neighbor v is authenticated using the master key K_v of node v, which was derived as $K_v = f_{K_I}(v)$. Since node u knows K_I, it can 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 authenticated with K_{uv} by node u will prove node u's identity.

Key Erasure When its timer expires, node u erases K_I and all the master keys K_v's 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_I. An adversary may have eavesdropped on all the traffic in this phase, but without K_I 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 $u < v$.

Handling Sleeping Nodes For a high density sensor network, it is suggested that maintaining only a necessary set of working nodes while turning off redundant ones would extend the lifetime of a sensor network [7, 43]. To employ our scheme in these applications, a new node u can establish pairwise keys with the working nodes. However, node u will not be able to establish pairwise keys with nodes that are in sleeping mode during the initial T_{min}. To address this issue, we let node u obtain neighbor lists from the working nodes. These lists will include most of the nodes within the two-hop range of node u. Node u can then proceed to compute the pairwise keys with the sleeping nodes and then erase K_I and other intermediate keys. Alternatively, two nodes can establish a pairwise key on the fly with the help of one or multiple neighboring nodes even if they have not established one yet within T_{min} [44].

3.3.1 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 mainly 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 can be easily fit into one packet. Moreover, the required space for storing preloaded keys is only one key, which is K_I. Thus, the computational, communication, and storage overhead of our scheme for pairwise key establishment is very small.

3.3.2 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 [10] 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 collision, do impose a challenge to our scheme (as well as to any other practical protocols for sensor networks). In Section 6 we discuss several techniques to reduce packet losses so that a sensor node can establish pairwise keys with most of its neighbors, if not all, within T_{min}.

In the previous description, 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, if not dropped due to buffer overflow, a neighbor node computes a MAC and sends back an ACK message. There are two solutions to mitigate this attack. First, the network controller can in addition pre-load a new sensor node with the current group key (not the initial network key K_I). A new node can authenticate its HELLO message to its neighbors using the current group key. Thus, a false HELLO message will be detected and dropped, and no ACK message will be sent. This approach however can only prevent outsider attacks. An insider adversary might know the current group 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 shortly.

Furthermore, to increase the difficulty for an adversary to recover K_I after it has physically captured a sensor node, the node can copy K_I 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. While this may not be true for keys stored on a disk, this is true for keys stored in memory because the keys are not cached in any other spaces. 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.

We stress that this scheme has a unique security property that is not possessed by other pairwise key establishment schemes [9, 15, 16, 47, 28]. Once a node has erased K_I, it will not be able to establish a pairwise key with any other nodes that have also erased K_I (but a newly added node can still establish a pairwise key with it anyway). This helps prevent node *clone attacks* or *sybil attacks* [14]. In a clone 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 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. We shall discuss the capability of our scheme in defending against several very sophisticated attacks such as sinkhole attacks [25] and wormhole attacks [19] in Section 5.2.1.

Increasing the Security of the Scheme In the described scheme, a newly added sensor node uses the same initial network key K_I 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 scheme is secure because K_I is secure. Next we consider more powerful attacks by assuming that K_I can be 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 due to K_I. 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_I. The basic idea behind this scheme is to remove the dependence on a single initial key K_I; instead, there is a sequence of initial keys used for deriving the master keys of individual nodes.

Assume that there are at most m node addition events for a sensor network 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: $K_1, K_2, ..., K_m$. These keys serve as initial keys. A node u deployed in time interval T_i is loaded with the initial key K_i, from which it derives its master key $K_i(u) = f_{K_i}(u)$ for T_i. It is also loaded with master keys $K_j(u) = f_{K_j}(u)$s for all $i < j \leq m$, and it will use a master key $K_j(u)$ $(i < j \leq m)$ for establishing pairwise keys with nodes that will be deployed in T_j. When deployed, node u uses K_i to establish pairwise keys with its neighbors and then erases K_i, as in our original scheme.

For example, a node u deployed in the first time interval T_1 is loaded with K_1 and $K_j(u)$s for all $1 < j \leq m$. It derives its master key $K_1(u)$ and uses K_1 as the initial network key to establish pairwise keys with its neighbors, and then erases K_1 after T_{min}. Note that the network controller or the mission authority should also erase the initial key K_1, since it is no longer needed. When adding a sensor node v in T_2, the network controller loads node v with initial key K_2, from which the node derives its master key $K_2(v)$, and $m-1$ master keys $K_2(v), K_3(v), ..., K_m(v)$. Node v uses K_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 used in T_2. When its timer expires, node v erases K_2.

As a result, we can see that an attacker compromising a new node u deployed in T_i within T_{min} obtains K_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 K_j $(1 \leq j < i)$ and the master keys of any other nodes for time intervals before T_i. Therefore, this scheme provides *backward confidentiality.* Furthermore, the attacker cannot derive any K_j $(i < j \leq m)$ either, so it cannot know any pairwise keys that other nodes will establish in the following time intervals. To this end, this scheme also provides *forward confidentiality.*

We note however that this scheme does not provide confidentiality for the time interval when a node is compromised in T_{min}, because an attacker can derive the master keys of the nodes that are deployed in the same time interval. This attack can be mitigated in practice if two nodes negotiate a new pairwise key through multiple round message exchanges instead of deriving their pairwise key from their master keys as discussed in our original scheme. In this case, an attacker has to intercept all the messages exchanged between the two sensor nodes to be able to recover their pairwise key.

If an attacker can compromise a sensor node within T_{min}, it can also 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. This attack can be addressed as follows. Suppose that the network controller wants to add N_i nodes into a network at the time interval T_i. It generates N_i ids for these nodes based on a random seed s_i and a well-known pseudo-random number generator such as LFSR; each of the N_i nodes is loaded with one unique id. When deployed, each node establishes a pairwise key with each neighbor based on our original scheme. T_{min} later, the network control broadcasts N_i and s_i in the network, using for example μTESLA [36] for broadcast source authentication. A neighbor node deployed earlier thus is able to verify if the id of the new node is valid based on s_i and N_i. If not, it will discard its pairwise key shared with this new node. We see that if the size of a node id is large enough, it is very hard for an attacker to forge valid ids. Thus this defeats the node addition attack.

3.3.3 Comparison with Other Pairwise Key Establishment Schemes

Next we compare our scheme with several other pairwise key establishment schemes based on security and performance. The results are shown in Table. 1. The KDC scheme [36] is a Kerberos-like scheme in which the base station helps two nodes to establish a pairwise key. In addition to MAC computations, the 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 Sybil attack unless the base station knows the topology of the network and only help those neighboring nodes establish pairwise keys.

There are two types of probabilistic key sharing schemes: those based on symmetric key cryptography [9, 16, 47] and those based on threshold cryptography [15, 28]. We denote them as Prob(1) and Prob(2), respectively. In these schemes, a node only has a certain probability to 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. Especially, those threshold cryptography based schemes involve relatively expensive operations such as modular multiplications. Moreover, these schemes only provide probabilistic security in the sense that a coalition of nodes can know the pairwise keys shared between other nodes with some probability.

The table shows that overall our scheme has much smaller performance overhead than other schemes, and it provides deterministic security although it has a small vulnerability window. Our scheme can also prevent Sybil attacks and other attacks (discussed in Section 5).

3.4 Establishing Cluster Keys

The cluster key establishment phase 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_u^c, then encrypts this key with the pairwise key shared with each neighbor, and then transmits the encrypted key to each neighbor v_i, $1 \leq i \leq m$.

$$u \longrightarrow v_i : (K_u^c)_{K_{uv_i}}. \tag{1}$$

Node v_i decrypts the key K_u^c, stores it in a table, and then sends back its own cluster key to node u. When one of the neighbors is revoked, node u generates a new cluster key and transmits it to all the remaining neighbors in the same way. It is easy to see that for a new node the cost of establishing a cluster key is $O(d)$ keys, where d is the number of neighbor nodes.

3.5 Establishing Group Keys

A group 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 on some event of interest or an instruction, to all the nodes in the network.

One way for the base station to distribute a message M securely to all the nodes is using hop-by-hop translation. Specifically, the base station encrypts M with its cluster key and then broadcasts the message. Each neighbor receiving the message decrypts it to obtain M, re-encrypts M with its own cluster key, and then re-broadcasts the message. The process is repeated until all the nodes receive M. However, this approach has a major drawback, that is, each intermediate node needs to encrypt and decrypt the message, consuming a nontrivial amount of energy on computation. Therefore, using a group key for encrypting a broadcast message is more preferable that using cluster keys.

A simple way to bootstrap a group key for a sensor network is to pre-load every node with the group 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 group key must be changed and distributed to all the remaining nodes in a secure, reliable, and timely fashion. This is referred to as *group rekeying.*

Unicast-based group rekeying, in which the key server sends a 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 [4, 31, 33, 41, 39] use logical key trees to reduce the complexity of a group rekeying operation from $O(N)$ to $O(logN)$. 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. Recent effort [29] attempts to reduce the unnecessary keys a node has to receive by mapping the physical locations of the member nodes to the logical key tree in LKH [41] for a static sensor network. However, the result is not very promising. Compared to the original LKH scheme, it is possible to reduce the energy cost of a group rekeying by 15% $\sim$ 37%, but it may incur a larger overhead in some scenarios. Moreover, for this scheme to work, the key server must have the global knowledge on the network topology.

The above schemes, when employed in sensor networks, raise a practical issue due to the small packet size [20]. Consider a group of $N = 1024$ nodes. To revoke a node, the number of keys to be distributed is 10 (assuming a binary key tree). Assume that every key is 10 bytes (8 bytes plus a 2-byte key id field), and that the payload of a packet is 29 bytes. The key server needs to broadcast 5 packets to avoid the fragmentation of keys. These packets must be *reliably* forwarded to all the nodes in a hop-by-hop fashion. This is a non-trivial task because of the unreliable transmission links and hidden terminal problems in wireless sensor networks.

Below we propose an efficient group rekeying scheme based on cluster keys. In our scheme the amortized transmission cost is *one key*, thus it can be easily fit into one packet. We first discuss *authenticated node revocation*, which is a prerequisite for group keying, then show the *secure key distribution* mechanism in detail.

3.5.1 Authenticated Node Revocation

In a sensor network, all the messages broadcast by the

Table 1: Comparison with Other Schemes

	Performance			Security	
Scheme	Computation	Communic-ation(hops)	Pre-storage (keys)	Deterministic	Sybil Prevention
KDC	MACs+Enc	≥ 2	One	Yes	No
Prob(1)	MACs+Enc	> 1	hundreds	No	No
Prob(2)	Modular+MACs+Enc	> 1	hundreds	No	No
Our	MACs	1	One	Yes	Yes

base station must be authenticated; otherwise, an outsider adversary may impersonate the base station. Therefore, a node revocation announcement must be authenticated during its distribution.

We employ the μTESLA [36] protocol for broadcast authentication because of its efficiency and tolerance to packet loss. μTESLA is based on the use of a one-way key chain along with delayed key disclosure. To use μTESLA, we assume that all the sensor nodes and the base station are loosely time synchronized, i.e., a node knows the upper bound on the time synchronization error with the base station.

To bootstrap its μTESLA key chain, the controller preloads every node with the commitment (i.e., the first key) of the key chain prior to the deployment of the network. The base station then discloses the keys in the key chain periodically in the order reverse to the generation of these keys. Since μTESLA uses delayed key disclosure, a node needs to buffer a received message until it receives the μTESLA key used for authenticating this message. Hence, there is a one μTESLA interval latency for node revocation.

Let u be the node to be revoked, and k'_g the new group key. Let the to-be-disclosed μTESLA key be k_i^T. The controller broadcasts the following message M:

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

Here we refer to $f_{k'_g}(0)$ as the *verification* key because it enables a node to verify the authenticity of the group key k'_g that it will receive later. The key server 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 updates its cluster key.

3.5.2 *Secure Key Distribution*

Secure 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 [20, 25]. 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. Each node keeps track of not only its parent and its children in the spanning tree, and also other neighbors. Note that in the TinyOS beaconing protocol a node does not maintain any information regarding any non-parent nodes in the spanning tree; however, this information is needed for secure key distribution and is critical for defending against various security attacks introduced in Section 5.

The new group key k'_g is distributed to all the legitimate sensor nodes via a recursive process over the spanning tree set up by the routing protocol. 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. Note that a node v that receives k'_g can verify the authenticity of k'_g 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.

Note that although we pointed out that hop-by-hop translation of regular broadcast messages involves non-trivial overhead for sensor nodes, it is not an issue for broadcasting a group key because only one key needs to be translated and group rekeying is a relatively less frequent event.

Finally, we note that it is desirable that the group key be updated more frequently even when no node revocation events occur. This is important to defend 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 updating instruction. Alternatively, every node can update the group key periodically on its own. For example, every node generates a new group key $K'_g = f_{K_g}(0)$ and then erases the old key K_g.

3.6 Local Broadcast Authentication

A mandatory requirement for a secure sensor network is that every message in the network must be authenticated before it is forwarded or processed; otherwise, an adversary can simply deplete the energy of the sensor nodes by injecting spurious packets into the network, even without compromising a single node. Moreover, the authentication scheme must be very lightweight in computation; otherwise, a sensor node may be engaged in verifying the false packets injected by an adversary.

Previous work [36] 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). A missing link is local broadcast authentication, which is needed for authenticating local broadcast messages (e.g., routing control packets) or supporting passive participation. We note that locally broadcast messages are usually event- or time-driven; for example, a node generates and broadcasts routing control messages periodically, or broadcasts aggregated sensor readings. Often, a node does not know in advance what is the next packet to transmit.

μTESLA [36] is not suitable for local broadcast authentication, although in Section 3.5 we employed μTESLA for providing global broadcast authentication of a node revocation message. This is because μTESLA does not provide immediate authentication. For every received packet, a node has to wait for one μTESLA interval to receive the MAC key used in computing the MAC for the packet. As a result, if μTESLA is used for local broadcast authentication, a message traversing l hops will take at least l μTESLA intervals to arrive at the destination. In addition, a sensor node has to buffer all the unverified packets. Both the latency and the storage requirement limit the scheme for authenticating infrequent messages (e.g., rekeying messages) broadcast by the base station. Nor are pairwise keys suitable for local broadcast authentication, because otherwise a node has to compute and attach m MACs for a 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. Figure 2 depicts this attack. Here node u has two neighbors x and v, and x is compromised. Since x knows u's cluster key K_u^c, it can send packets to node v while impersonating node u by using K_u^c in computing MACs.

Due to the symmetric nature of cluster keys, this impersonation attack cannot be completely defeated. Therefore, our goal is to design a scheme to thwart this attack at the maximum. The main idea is to use one-time authentication key.

3.6.1 One-way Key Chain Based Authentication

We propose to use one-way key chain [26] for 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, then transmits the commitment (i.e., the first key) of the key chain to each neighbor, encrypted with their pairwise shared key. 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 more recently.

The design of our authentication scheme is 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 *triangular inequality* among the distances of the involved nodes, which is illustrated in Figure. 2. 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.

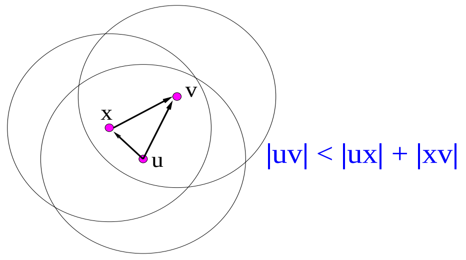


Figure 2: Triangular Inequality

The above authentication scheme provides source authentication (like an authentication scheme based on pairwise shared keys) while not precluding passive participation (unlike an authentication scheme based on pairwise shared keys). However, we note that an attacker can overcome our scheme by preventing node v from receiving the packet from u directly. For example, the adversary might shield node v or jam node v by letting another node w transmitting 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 combining AUTH keys with cluster keys. For example, node u can compute a new AUTH key by XORing an AUTH key with its cluster key, and use the new AUTH key for packet authentication. 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 has only the cluster keys of its one-hop neighbors.

4. PERFORMANCE EVALUATION

We only consider the overhead for updating cluster keys and group keys 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 new group keys to the nodes in the system.

4.1 Computational Cost

When 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 d_0 be the number of neighbors of the node being revoked, and d_i, $i = 1, 2, ..., d_0$ the number of legitimate neighbors of each of these d_0 neighbors. The total number of encryptions is simply $S_e = \sum_{i=1}^{d_0} d_i$. 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 d_0 nodes depends on its location. In the worse case where a node is a neighbor to all these d_0 nodes, it needs to decrypt d_0 keys. For an individual node, the total number of symmetric key operations it performs is bounded by $(\max(d_i) + d_0 - 1)$. For a network of size N, the average number of symmetric key operations a node performs during cluster key updating is $\frac{2S_e}{N}$.

During the secure distribution of a group key, the number of decryptions is equal to the network size N because every node needs to decrypt once. Recall that we use cluster keys for secure forwarding of the group 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. Therefore, the total number of symmetric key operations is at most $2N$ and the amortized cost is at most 2 symmetric key operations per node.

The above analysis shows that the computational cost in a node revocation 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 about $2(d-1)^2/(N-1)+2$. 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.

4.2 Communication Cost

The analysis of communication cost for a group rekeying event is similar to that of 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 group 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 [41], the communication cost of a group rekeying is $O(logN)$. Thus, our scheme is more scalable than LKH if LKH is used for group rekeying in sensor networks.

4.3 Storage Requirement

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 group 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. Let L be the number of keys a node stores for its key chain. Thus, the total number of keys a node stores is $3d + 2 + L$.

Although memory space is a very scarce resource for the current generation of sensor nodes (4 KB RAM in a Berkeley Mica Mote), for a reasonable degree d, storage is not an issue in our scheme. For example, when $d = 20$ and $L = 20$, a node stores 82 keys (totally 656 bytes when the key size is 8 bytes).

Overall, we conclude our scheme is scalable and efficient in computation, communication and storage.

5. 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.

5.1 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 somehow, our group rekeying scheme can efficiently revoke node u from the group. Basically, every neighbor of node u deletes its pairwise key shared with u and updates its cluster key. The group key is also updated efficiently. 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* under *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 5.2.

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 (in case of in-network processing) 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, neither can a compromised node establish trust relationship with any nodes other than its neighbors, nor can it jeopardize the secure links between other nodes.

Third, possessing the group 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, no matter what security mechanisms are used for securing message distribution. Moreover, possessing the group 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 we deploy a periodic group rekeying scheme, the adversary can decrypt only the messages being encrypted using the current group key.

5.2 Defending against Various Attacks on Secure Routing

Karlof and Wagner [25] have studied various attacks on the security of routing protocols for wireless sensor network. We now show 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 introduce in Section 5.2.1). 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, generating false error messages. The adversary may also launch the *Selective Forwarding* attack in which the 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 two-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 group rekeying scheme can revoke the node from the network very efficiently.

Our scheme can *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. If this attack succeeds, all the nodes may send their readings or other packets into oblivion. However, this attack will not succeed in LEAP because every node only accepts packets from its authenticated neighbors. As we discussed earlier, our scheme can also prevent the *Sybil attack*.

5.2.1 *Dealing with the Wormhole and Sinkhole Attacks*

The attack that is most difficult to detect or prevent is one that combines the *Sinkhole* and the *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 the *wormhole* attack, typically two distant malicious nodes, which have an out-of-band low latency link that is invisible to the underlying sensor network, collude to understate their distance form each other. When 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. Thus this creates a *sinkhole*. Similarly, nodes that are multiple hops away from each other may believe they are neighbors via 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 [19] 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 coordinate.

In LEAP, an outsider adversary cannot succeed in launching wormhole attacks in any time other than 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 (in 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 close to the base station, the other one v in the area of interest, it may succeed in creating node v as a *sinkhole* 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 it is away from 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.

6. THE IMPLEMENTATION OF LEAP

We have implemented a prototype of LEAP in the TinyOS platform [20]. The programs were written in nesC, a C-like language for developing applications in TinyOS. Figure. 3 depicts the components and the interfaces of TinyOS used by LEAP. Basically, LEAP uses several Timer interfaces (provided by a Timer component) for providing its key erasure time threshold and for handling message retransmission as a result of packet losses during key establishment phases. It uses a linear-feedback shift register (LFSR) component to generate pseudo-random numbers. LEAP uses the RC5 block cipher [37] 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 both replaced with MACs. As such, 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 less variables and cipher contexts have to be defined.

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 en-

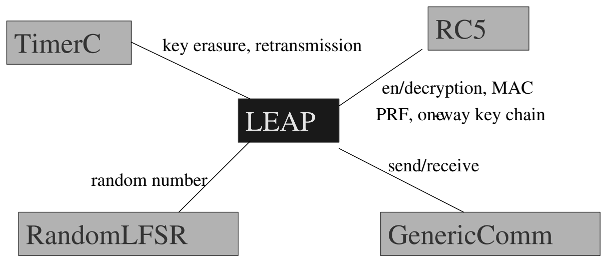


Figure 3: The components and the interfaces of TinyOS used in the implementation of LEAP

sure that a new node establishes keys with the majority of its neighbors, if not all. Providing perfect 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., > 20 neighbors). As a result, a new node probably misses one or several of the ACKS due to high collision probability. We note that although the media access control component in TinyOS, ChannelMonM, 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 on its own. That is, after a node receives a HELLO message, it delays a random time between $0 \sim T_1$ before sending its ACK message, where T_1 is chosen based on d. This greatly reduces the probability of packet collision.

We add the following mechanisms in the implementation of LEAP to further increase the probability of two nodes establishing pairwise/cluster keys. First, a new node broadcasts its first HELLO message at a random time between $0 \sim T_2$ (e.g., $T_2 = 3$) seconds after it is deployed and it broadcasts the HELLO message multiple times (e.g., 3 times with an interval of 3 seconds if $T_{min} > 12$ seconds). Second, motivated by the TCP protocol, we integrate the pairwise key establishment phase with the cluster establishment phase to mimic the TCP 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.

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

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 an ACK2 message, which includes its cluster key, encrypted also with K_v (or K_{uv}. Recall that node u can derive K_v and K_{uv} from K_I). The advantage of this integration is that both nodes u and v can infer if their messages are lost and schedule retransmission 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 messages, it infers that its ACK2 message is lost, so it can schedule to retransmit it. These mechanisms help a new node establish keys with all its neighbors. We note that the number of neighbors a new node can establish keys with highly depends on the number of maximum retransmissions chosen in the implementation.

We upload and run the code[1] in Mica2 Motes [10]. The code space is 17.9KB (out of 128 KB) in ROM. The usage of data space in RAM (totally 4 KB) depends on the number of neighbors d a node has. If a node has more neighbors, it has to store more pairwise keys and cluster keys, thus more RAM space is needed. Table 2 shows the usage of RAM space 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 requirement on memory space is feasible for the Mica2 motes.

Finally, we should mention that packet loss and feedback implosion are not unique issues to LEAP. Most practical security and non-security protocols for sensor networks face the same issues, although they have rarely touched these 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 [15, 16, 28].

7. RELATED WORK

Stajano and Anderson discuss various issues that arise for secure devices consisting of "peanut nodes" [38]. In particular, they propose that nodes bootstrap trust relationship through physical contact. Zhu et al [46] propose an efficient scheme for bootstrapping trust among mobile nodes based on the combination of TESLA [32] and one-way hash chain. Carman, Kruus and Matt have analyzed several approaches for key management and distribution in sensor networks [8]. 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* [5] discuss 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.

Eschenauer and Gligor [16] present a key management scheme for sensor networks based on probabilistic key predeployment. Chan et al [9] extend this scheme and present three mechanisms for key establishment. Zhu et al [47] present an approach for establishing a pairwise key that is exclusively known to a pair of nodes with overwhelming probability, based on the combination of probabilistic key sharing and threshold secret sharing. Du et al [15], Liu and Ning [28] combine the technique of probabilistic key predeployment with the Blom's [3] or the Blundo's [6] key management schemes for establishing pairwise keys in sensor

[1]The current version of our code only provides pairwise key establishment, cluster key establishment, and one-way key chain generation. It does not include the group rekeying scheme and the broadcast source authentication based on μTESLA.

Table 2: 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

networks. They have shown that their schemes are more robust to node collusion attacks than the previous schemes [9, 16]. We have compared in detail both the performance and the security of our pairwise key establishment scheme with these schemes in Section 3.3.

Perrig *et al* present security protocols for sensor networks [36]. 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 Sybil attacks [14]. In contrast, in our scheme pairwise keys are established in a distributed fashion without the involvement of the base station.

Karlof *et al* [24] describe TinySec, a link layer security mechanism using a single preloaded fixed group 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 [11] discuss several security mechanisms for supporting in-network processing in hierarchical sensor networks. Karlof and Wagner [25] describe several security attacks on routing protocols for sensor networks. As we have shown in Section 5.2, our scheme can prevent or thwart many of these attacks very efficiently.

Liu and Ning [27] present a multi-level key chain scheme for μTESLA. Hu and Evans [18] propose a secure hop-by-hop data aggregation scheme that works if one node is compromised. Ye et al [42] propose a statistical en-route detection scheme called SEF, which allows both the base station and en-route nodes to detect false data with a certain probability. Zhu et al [45] introduce an interleaved hop-by-hop authentication scheme that can filter false data packets injected by a certain number of compromised colluding nodes. Przydatek et al [35] present SIA, a secure information aggregation scheme for sensor networks. Through statistical techniques and interactive proofs, SIA allows an end-user to decide with high probability if an aggregated sensor report from an aggregation point(AP) is authenticated without having to obtain all the original sensor readings from the AP. Wood and Stankovic [40] identify a number of DOS attacks in sensor networks. Deng et al [12] propose a multiple-base station and multiple-path strategy to increase intrusion tolerance, and an anti-traffic analysis strategy to disguise the location of a base station.

8. 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 group key shared with all the nodes in the network. These keys can be used to increase the security of many non-secure protocols.
- LEAP includes an efficient protocol for 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. Our implementation indicates LEAP is feasible for the current generation sensor nodes.

9. REFERENCES

[1] N. Abramson. The aloha system another alternative for computer communications. In Proc. of the Fall 1970 AFIPS Computer Conference, 281–285, 1970.

[2] R. Anderson, M. Kuhn. Tamper Resistance – a Cautionary Note. The Proc. of Second USENIX Workshop on Electronic Commerce, November, 1996.

[3] R. Blom. An optimal class of symmetric key generation systems. In *Advances in Cryptology, Proceedings of EUROCRYPT'84.* LNCS 209. 335–338. 1995

[4] D. Balenson, D. McGrew, and A. Sherman. Key management for large dynamic groups: One-way function trees and amortized initialization. IETF Internet draft (work in progress), August 2000.

[5] S. Basagni, K. Herrin, E. Rosti, D. Bruschi. Secure Pebblenets. In Proc. of MobiHoc 2001.

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

[7] A. Cerpa and D. Estrin. ASCENT: Adaptive selfconfiguring sensor network topologies. In Proc. of INFOCOM'02, June 2002.

[8] D. Carman, P. Kruus and B. Matt, Constraints and approaches for distributed sensor network security, NAI Labs Technical Report No. 00010 (2000).

[9] H. Chan, A. Perrig, D. Song. Random Key Predistribution Schemes for Sensor Networks. In Proc. of the IEEE Security and Privacy Symposim 2003, May 2003.

[10] Crossbow technology inc. URL: http://www.xbow.com.

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

[12] J. Deng, R. Han, and S. Mishra. Intrusion tolerance strategies in wireless sensor networks. In Proc. of IEEE 2004 International Conference on Dependable Systems and Networks (DSN'04), 2004.

[13] T. Dierks and C. Allen. The TLS protocol version 1.0. 1999.

[14] J. Douceur. The Sybil Attack. In First Interntional Workshop on Peer-to-Peer Systems (IPTPS'02), 2002.

[15] W. Du and J. Deng and Y. Han and P. Varshney. A pairwise key pre-distribution scheme for wireless sensor networks. In Proc. of the 10th ACM Conference on

Computer and Communications Security (CCS'03). 42–51, 2003.

[16] L. Eschenauer and V. Gligor. A Key-Management Scheme for Distributed Sensor Networks. In Proc. of ACM CCS 2002

[17] O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. Journal of the ACM, Vol. 33, No. 4, 1986, pp 210-217.

[18] L. Hu and D. Evans. Secure aggregation for wireless networks. In Proc. of Workshop on Security and Assurance in Ad hoc Networks. 2003

[19] Y. Hu, A. Perrig, and D. Johnson. Packet Leashes: A Defense against Wormhole Attacks in Wireless Ad Hoc Networks. Proceedings of INFOCOM 2003, IEEE, San Francisco, CA, April 2003, to appear.

[20] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In Proc. of ASPLOS IX, 2000.

[21] C. Intanagonwiwat, R. Govindan and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks In Proc. of MobiCOM'00, Boston, Massachussetts, August 2000.

[22] C. Karlof, Y. Li, and J. Polastre. ARRIVE: An Architecture for Robust Routing In Volatile Environments. Technical Report UCB/CSD-03-1233, University of California at Berkeley, Mar. 2003.

[23] J. Kohl and B. Neuman. The Kerberos Network Authentication Service (V5). RFC 1510, Sep. 1993.

[24] C. Karlof, N. Sastry, U. Shankar, and D. Wagner. TinySec: TinyOS Link Layer Security Proposal, version 1.0, Unpublished manuscript, July 2002.

[25] C. Karlof and D. Wagner. Secure Routing in Sensor Networks: Attacks and Countermeasures. To appear in Proc. of First IEEE Workshop on Sensor Network Protocols and Applications, May 2003.

[26] L. Lamport, Password authentication with insecure communication communication. Communications of the ACM, 24(11):770-772, Nov., 1981.

[27] D. Liu and P. Ning, Efficient Distribution of Key Chain Commitments for Broadcast Authentication in Distributed Sensor Networks In Proc. of NDSS'03, Feb. 2003

[28] D. Liu and P. Ning. Establishing pairwise keys in distributed sensor networks. In Proc. of the 10th ACM Conference on Computer and Communications Security (CCS '03). 52-61. 2003

[29] L. Lazos and R. Poovendran. Energy-aware secure multicast communication in ad-hoc networks using geographic location information. In Proc. of IEEE ICASSP'03, 2003.

[30] S. Madden, R. Szewczyk, M. Franklin, and D. Culler. Supporting Aggregate Queries Over Ad-Hoc Wireless Sensor Networks. In 4th IEEE Workshop on Mobile Computing Systems & Applications, June 2002.

[31] D. Naor, M. Naor, and J. Lotspiech. Revocation and tracing schemes for stateless receivers. In *Advances in Cryptology - CRYPTO 2001*. Lecture Notes in Computer Science, vol. 2139. 41–62. 2001

[32] A. Perrig, R. Canetti, J. Tygar, D. Song. Efficient authentication and signing of multicast streams over lossy channels. In IEEE Symposium on Security and Privacy. May 2000.

[33] A. Perrig, D. Song, and D. Tygar. Elk, a new protocol for efficient large-group key distribution. In Proc. of IEEE Symposium on Security and Privacy. 2001.

[34] R. Pickholtz, D. Schilling, and L. Milstein. Theory of spread spectrum communications a tutorial. *IEEE Transactions on Communications 30(5)*, 855–884. 1982

[35] B. Przydatek, D. Song, and A. Perrig. SIA: Secure information aggregation in sensor networks. In Proc. of ACM SenSys 2003.

[36] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. Tygar. SPINS: Security Protocols for Sensor Networks. In Proc. of Seventh Annual ACM International Conference on Mobile Computing and Networks(Mobicom 2001), Rome Italy, July 2001.

[37] R. Rivest. The rc5 encryption algorithm. In Proc. of the 1st International Workshop on Fast Software Encryption. 86–96. 1994

[38] F. Stajano and R. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In Security Protocols, 7th International Workshop. Springer Verlag, 1999.

[39] D. Wallner, E. Harder, and R. Agee. Key management for multicast: Issues and architecture. Internet Draft, draft-wallner-key-arch-01.txt. 1998

[40] A. Wood and J. Stankovic. Denial of service in sensor networks. *IEEE Computer*, 54–62, 2002.

[41] C. Wong, M. Gouda,S. Lam. Secure Group Communication Using Key Graphs. In Proc. Of SIGCOMM'98, 1998.

[42] F. Ye, H. Luo, S. Lu, and L. Zhang. Statistical En-route Detection and Filtering of Injected False Data in Sensor Networks. To appear in Proc. of IEEE INFOCOM 2004.

[43] F. Ye, G. Zhong, S. Lu, L. Zhang. PEAS: A Robust Energy Conserving Protocol for Long-lived Sensor Networks. In Prof. of ICDCS 2003, Providence Rhode Island, May, 2003.

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

[45] S. Zhu, S. Setia, S. Jajodia and P. Ning. An Interleaved Hop-by-hop Authentication Scheme for Filtering of Injected False Data in Sensor Networks. In Proc. of IEEE Security and Privacy, 2004.

[46] S. Zhu, S. Xu, S. Setia, and S. Jajodia. LHAP: A Lightweight Hop-by-Hop Authentication Protocol For Ad-Hoc Networks. In ICDCS 2003 International Workshop on Mobile and Wireless Network (MWN 2003), Providence, Rodhe Island, May 2003.

[47] S. Zhu, S. Xu, S. Setia, and S. Jajodia. Establishing Pair-wise Keys For Secure Communication in Ad Hoc Networks: A Probabilistic Approach. In Proc. of the 11th IEEE International Conference on Network Protocols (ICNP'03), 2003.