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

In this paper, 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 inter-node traffic 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 and passive participation. We analyze the performance and the security of our scheme under various attack models and show our schemes are very efficient in defending against many attacks.

Categories and Subject Descriptors

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

General Terms

Design, Security

CCS'03, October 27–31, 2003, Washington, DC, USA.

Keywords

Security Mechanism, Sensor Networks, Key Management, In-network Processing

1. INTRODUCTION

Many sensor systems are deployed in unattended and often adversarial environments. 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 (4 MHz) under-powered processor with only 8 KB of memory [22]. Thus, key management protocols for sensor networks are based upon symmetric key algorithms.

A fundamental issue that must be addressed for using key management protocols based on symmetric shared keys is the mechanism 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 [7] and Kerberos [15] developed 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, 2], approaches in which every node shares a unique key with the base station [22], and approaches based on random key sharing [9, 6].

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(s).

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 [16]. In many applications, sensors in the network are organized into a data fusion or aggregation hierarchy for efficiency. 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 [13]. Passive participation is another form of in-network processing in which a sensor node can take certain actions based on overheard messages [14, 20], 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 be able to decrypt or authenticate 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. On the other hand, if a pairwise shared key is used for encrypting or authenticating a message, it effectively precludes passive participation in the sensor network.

In this paper, 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 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, our protocol 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 inter-node traffic 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).

The rest of this paper is organized as follows. We discuss our design goals and assumptions in Section 2, before describing the LEAP protocol in detail in Section 3. The inter-node traffic authentication protocol is described in Section 3.3. In Section 4 and 5, we analyze the performance and security of our protocol. We discuss related work in Section 6 before concluding the paper in Section 7.

2. ASSUMPTIONS AND DESIGN GOALS

We describe below our assumptions regarding the sensor network scenarios in which our keying protocols will be used, before discussing the design goals of our protocol.

2.1 Network and Security Assumptions

We assume that the sensor network is static, i.e., sensor nodes are not mobile. The base station, acting as a controller (or 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 current generation sensor nodes, e.g. the Berkeley MICA motes [12]. 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. However, we assume 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 also be compromised. However, we assume the base station will not be compromised.

2.2 Design Goals

The main goal of LEAP is to design efficient security mechanisms for supporting various communication models in sensor networks. The security requirements not only include authentication and confidentiality but also robustness and survivability. In other words, the sensor network should be robust against various security attacks, and if an attack succeeds, its impact should be minimized. For example, the compromise of a single node should not break the security of the entire network.

The protocol should also support sensor network optimization mechanisms such as in-network processing. Since the resources of a sensor node are very constrained, the key establishment protocols should be lightweight and minimize communication and energy consumption. It should be possible to add new sensor nodes incrementally to the sensor network. The keying protocols should be scalable, i.e., the size of the sensor network should not be limited by the per-node storage and energy resources.

3. LEAP: LOCALIZED ENCRYPTION AND AUTHENTICATION PROTOCOL

As discussed in the introduction, LEAP provides multiple keying mechanisms that can be used 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 protocols used by LEAP for establishing these keys. The inter-node traffic authentication mechanism that is part of LEAP is discussed separately in Section 3.3.

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 type 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 transmitted by a sensor node and the queries sent by the base station may need confidentiality.

We argue that *no single* keying mechanism is appropriate for all the secure communication that is 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 pairwise with the base station. This key is used for secure communication between a node and the base station. For example, a node may send an alert to the base station if it observes any abnormal or unexpected behavior by 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 [13, 14, 20]. 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. For passive participation to be feasible, neighboring nodes should be able to decrypt and authenticate some classes of messages, e.g., sensor readings, transmitted by their neighbors. This means that such messages should be encrypted or authenticated by a *locally shared* key. Therefore, in LEAP each node possesses a unique cluster key that it uses for securing its messages, while its immediate neighbors use the same key for decryption or authentication of 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 transmissions of its sensor readings to an aggregation node. Note that the use of pairwise keys precludes passive participation.

3.2 Key Establishment

In this section, we describe the schemes provided by LEAP for sensor nodes to establish and update individual keys, pairwise shared keys, cluster keys, and group keys for each node. We note that 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 [10].
- $\{s\}_k$ 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 $K0 = f_K(0)$ for encryption and use $K1 = f_K(1)$ for authentication. For ease of presentation, in the following discussion, we simply say that a message is encrypted or authenticated with key K, although the message is really encrypted with $K0$ and authenticated with $K1$ respectively.

3.2.1 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: $K_u^m = f_{K_s^m}(u)$. Here f is a pseudo-random function and K_s^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 for keeping 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.

3.2.2 Establishing Pairwise Shared Keys

In this paper, unless otherwise specified, a pairwise shared key belonging to a node refers to a key shared only between the node and one of its *direct* neighbors (i.e. one-hop neighbors).

For nodes whose neighborhood relationships are predetermined (e.g., via physical installation), pairwise key establishment is simply done by preloading the sensor nodes with

the corresponding keys. Here we are interested in establishing pairwise keys for sensor nodes unaware of their neighbors until their deployment (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 designed to sustain possible break-in attacks at least for a short interval (say several seconds) when captured by the adversary; otherwise, the adversary could easily compromise all the sensor nodes in a sensor network and then take over the network. Therefore, instead of assuming that sensor nodes are tamper resistant which often turns out not to be true [1], 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 $T_{min} > T_{est}$.

We now 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. Below, we describe the four steps for adding a new node u to the sensor network.

Key Pre-distribution The controller generates an initial key K_I and loads each node with this key. Each sensor 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 a nonce, and waits for each neighbor v to respond with its identity. The reply from each neighbor v is authenticated using its master key K_v. Since node u can compute K_v using K_I, it is able to verify node v's identity independently.

$$u \longrightarrow * : \quad u, Nonce_u.$$
$$v \longrightarrow u : \quad v, MAC(K_v, Nonce_u|v).$$

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} independently. No message is exchanged between u and v in this step. Note that node u does not have to authenticate itself to node v explicitly and immediately, because any future messages authenticated with K_{uv} by node u will prove node u's identity. No other nodes can compute K_{uv} after the *key erasure* phase below.

Key Erasure When its timer expires, node u erases K_I and all the keys K_v it computed in the neighbor discovery phase.

After the steps above, node u will have established a pairwise shared key with each of its neighbors. Further, no node in the network possesses 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 after T_{min} has expired will only obtain the keying material of the compromised node, and not that of any other node. Thus, this scheme localizes the security impact of a node compromise. 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 step 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 $u < v$.

Discussion 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 most sensor networks and adversaries. The current generation of sensor nodes can transmit at the rate of 19.2 Kbps [24] whereas the size of an id announcement message is very small (12 bytes if an id is 4 bytes and the MAC size is 8 bytes). The probability of collision is quite small when a non-persistent CSMA protocol is used for medium access control [25]. Moreover, a node can broadcast its id multiple times to increase the probability that it is received by all its neighbors. We expect the total time in this phase should be of the order of several seconds.

Furthermore, to increase the difficulty for an adversary to recover K_I after it has physically capturing 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 disk, we believe that this is true for keys stored in memory. Another implicit assumption is that is a node u will not compute and then 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.

Node Addition For a very dense sensor network, researchers suggest that maintaining only a necessary set of working nodes while turning off redundant ones would extend the lifetime of a network [3, 27]. 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 sleep mode during the initial T_{min}. To address this issue, we let node u obtain neighbor lists from the working nodes which will include most of the nodes within two hops of node u. Node u can then proceed to compute its pairwise keys for the sleeping nodes and then erase K_I and other intermediate keys. Alternatively, the multi-path scheme in Section 3.2.4 can be used to establish a pairwise key on the fly for two nodes even though they have not established one within T_{min}.

3.2.3 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 of each neighbor, and then transmits the encrypted key to each neighbor v_i.

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

Node v_i decrypts the key K_u^c and stores it in a table. When one of the neighbors is revoked, node u generates a new cluster key and transmits to all the remaining neighbors in a same way.

3.2.4 Establishing Multi-hop Pairwise Shared Keys

When a node wants to send its sensor readings to an aggregation node (or its cluster head) that is *multiple hops* away, to achieve privacy or strong source authentication, it has to use a pairwise key that is shared only between itself and the aggregation point. We call such keys *multi-hop* pairwise shared keys to distinguish them from pairwise shared keys established with immediate neighbors.

We can simply extend the (one-hop) pairwise shared key establishment scheme discussed above for the establishment of two-hop pairwise keys. Specifically, once a node discovers its neighbors in the *neighbor discovery* phase, it then broadcasts their ids. As a result, a node discovers all the nodes that can be reached in two hops. It can then establish a pairwise shared key with all the nodes that are two hops away using the same scheme it used for one-hop pairwise key establishment. This scheme works well if (i) two-hop pairwise shared keys can be established within T_{min} and (ii) a node has memory space to store these two-hop pairwise shared keys, in addition to all the other keys it needs to store.

If these two conditions cannot be satisfied, we can use the following scheme to establish a two-hop pairwise shared key on the fly. A node u broadcasts a QUERY message which contains its own id u and the id of the cluster head c. All the nodes that are a neighbor to both the node u and the cluster head c sends back a REPLY, authenticated with their pairwise keys shared with node u. We call the intermediate nodes *proxies*.

Using this procedure, node u may find multiple (say m) proxies, denoted as $v_1, v_2, ..., v_m$. To establish a pairwise key S with node c, node u first splits S into m shares, i.e., $sk_1, sk_2, ..., sk_m$ such that $S = sk_1 \oplus sk_2 ... \oplus sk_m$; it then forwards each share sk_i to the cluster head c through the i^{th} proxy v_i. Namely,

$$u \longrightarrow v_i : \{sk_i\}_{K_{uv_i}}, f_{sk_i}(0).$$
$$v_i \longrightarrow c : \{sk_i\}_{K_{v_ic}}, f_{sk_i}(0).$$

Here $f_{sk_i}(0)$ is called the *verification* key of key sk_i because it allows the cluster head c to verify immediately if sk_i is valid. A compromised node v_i may change sk_i to sk_i' and attach $f_{sk_i'}(0)$, but node u will detect this attack by overhearing node v_i's transmission. Node v_i is required to erase sk_i after it has done the forwarding for node u, which is important to prevent sk_i from being disclosed if node v_i is compromised later. After the cluster head c receives all shares, it re-constructs S. The cluster head can send back a DONE message authenticated with S, which allows node u to detect if the established pairwise key is correct. Clearly, this scheme is secure when up to $m-1$ proxies are compromised.

The scheme described above can be extended for establishing a pairwise key on the fly for two nodes that are more than two hops away. Its performance overhead will depend on the desired security level m. An alternative approach for establishing multi-hop pairwise keys is to use the base station as a helper [22]. However, it is unclear if this approach will outperform our multiple path scheme. Designing a more efficient scheme for this purpose without the involvement of the base station is an interesting research problem.

3.2.5 Establishing Group Keys

A group key is a key shared by all the nodes in the network, and it is necessary when the controller is distributing a secure message, e.g., a query on some event of interest or a confidential 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, thus consuming a non-trivial amount of energy on computation. Therefore, using a group key for encrypting a broadcast message is preferable from the performance point of view.

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. The naive approach in which the base station encrypts the updated group key using the individual key of each node and then sends the encrypted key to each node separately is not scalable because its communication and computational costs increase linearly with the size of the network.

Below we propose an efficient key updating scheme based on cluster keys. We first discuss *authentic node revocation* that is a prerequisite for group keying, then describe the *secure key distribution* mechanism in detail.

3.2.5.1 Authentic Node Revocation.

In a sensor network, all the messages the base station broadcasts to the sensors should be authenticated; otherwise, an outsider adversary or a compromised node may impersonate the base station. Therefore, a node revocation announcement must be authenticated when distributed.

We employ μTESLA [22], a broadcast authentication protocol proposed by Perrig *et al*, due to 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 key server are loosely time synchronized, i.e., a node knows the upper bound on the time synchronization error with the key server.

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. The use of μTESLA allows the key server to broad-

cast authenticated packets (including keying materials and data messages) efficiently. 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. Thus, 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:

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

Again, 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. In addition, if node v is a neighbor of node u, v will remove its pairwise key with u and updates its cluster key.

3.2.5.2 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 [12, 17]. 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 immediate 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 necessary for secure key distribution below and for defending against various attacks 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 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 the hop-by-hop encryption and decryption overhead for translating broadcast messages is non-trivial for sensor nodes, we believe it is still affordable for distributing a new group key. This is because the distributed message contains only one key. Moreover, group rekeying events in most sensor networks can be expected to be relatively infrequent.

Finally, we note that it is desirable that the group key be updated more frequently even when no revocation event occurs. This is important to defend against cryptanalysis and to prevent the adversary from decrypting all the previously distributed messages by 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. Every node generates a new group key $K'_g = f_{K_g}(0)$ and then erases the old key K_g.

3.3 Inter-node Traffic Authentication

We now consider the issue of inter-node traffic authentication that is critical in defending against various attacks such as DoS. A mandatory requirement for a secure sensor network is 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 computationally very lightweight, otherwise, a sensor node may be engaged in verifying a large number of packets from the adversary leading to its energy being depleted.

We have discussed the issue of authentic node revocation in Section 3.2.5. We employ μTESLA for broadcast authentication for the controller. Given the loose time synchronization condition, μTESLA assures the authenticity of a broadcast message by using one-way key chain and delayed key disclosure. Unfortunately, μTESLA is not suitable for inter-node traffic authentication because it does not provide immediate authentication. This is because a node receiving a packet has to wait for one μTESLA interval to receive the delayed disclosed MAC key; as a result, a message traversing l hops will take at least l μTESLA intervals to arrive at the destination. Moreover, a sensor node has to buffer all the unverified packets. Both the latency and the storage requirements of this scheme make it unsuitable for authenticating all traffic, although it suffices when authenticating infrequent messages (e.g., rekeying messages) broadcast by a base station. Therefore, we need authentication mechanisms other than μTESLA for immediate traffic authentication.

One solution to this problem is to use pairwise keys for authentication. Using pairwise keys provide source authentication, but it precludes passive participation. To enable passive participation, it is necessary to use cluster keys for authentication. The basic scheme is as follows. Every node authenticates a packet it transmits using its own cluster key as the MAC key. A receiving node first verifies the packet using the same cluster key it obtained from the sending node in the cluster key establishment phase, then authenticates the packet to its own neighbors with its own cluster key. Thus, a message gets authenticated repeatedly in a *hop-by-hop* fashion if it traverses multiple hops.

The above approach provides immediate authentication, and its communication overhead is small because a node only adds one MAC to each packet. However, although this approach defends against outsider attacks in which the adversary does not hold any keys, insider attacks are possible after the adversary compromises a sensor node. The adversary could inject spurious packets into the network, authenticated with the cluster key of the compromised node or with the cluster keys of the neighbors of the compromised node. The former attack exists in both pairwise key based and cluster key based authentication schemes, and it is very hard to prevent and detect. We do not address this attack in this work. However, the latter impersonation attack only exists in the cluster key based authentication scheme because a cluster key is shared between a node and all its neighbors.

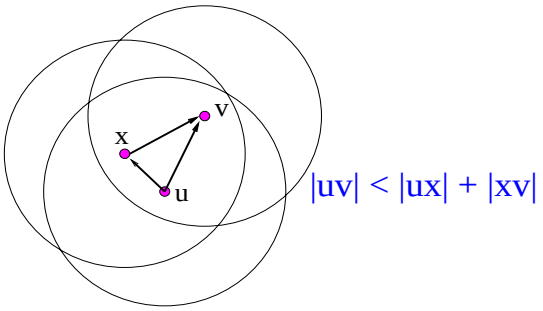

Figure 1: Triangular Inequality

Below we propose a scheme to thwart this impersonation attack.

3.3.1 One-way Key Chain Based Authentication

To address the impersonation attack described above, we propose to use one-way hash key chain [18] 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 a 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 neighbor receiving the message can verify its authenticity based on the commitment or a recently disclosed AUTH key of the sending node.

Our authentication scheme is motivated by two observations. First, since packets are authenticated hop-by-hop, a node only needs to authenticate a 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 demonstrated in Fig. 1. 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 $|uv| < |ux| + |xv|$. This means, the adversary x cannot reuse node u's AUTH keys to impersonate node u.

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 there is a possible impersonation attack on this scheme. For example, an adversary can 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 impersonating node u. Because node v has not received a packet with the same AUTH key, it will accept the modified packet.

If this attack is launched by an outsider adversary, we can simply prevent it as follows. Node u combines its AUTH keys with its cluster key (e.g., XORing them together), then authenticates the packets it sends using the combined keys as the MAC keys. The outsider adversary does not know node u's cluster key thus is unable to launch this attack. Unfortunately, we do not have a lightweight countermeasure to prevent the attack by an insider adversary. However, we note that (i) The maximum number of erroneous 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 only allows the adversary to launch such attack in a two-hop zone of the compromised node x, because node x only has the cluster keys of its one-hop direct neighbors. To further deter this attack, we present below a probabilistic scheme for detecting it.

3.3.2 Probabilistic Challenge Scheme

In this scheme, a node challenges the authenticity of a received packet with a certain probability. More specifically, when node v receives a packet P (we assume every packet includes a count C for message freshness [22]) with AUTH key K from (claimed) node u, it challenges node u for the authenticity of packet P with probability p_c, using their pairwise key K_{uv} as the MAC key. The process is as follows.

$$\begin{aligned} v \xrightarrow{p_c} u: &\quad C, N_v, MAC(K_{uv}, C|N_v) \\ u \longrightarrow v: &\quad N_u, MAC(K_{uv}, C|N_v|N_u) \end{aligned} \tag{1}$$

Here N_u and N_v are the nonces generated by node u and node v respectively. The compromised node x cannot forge the response impersonating node u because it does not have the pairwise key K_{uv}. Thus, the insider adversary takes the risk of being detected when it launches the above impersonation attack, subject to the challenge probability p_c. We note the choice of p_c should make a tradeoff between security and performance, because a larger p_c leads to a stronger security, but incurs a larger overhead for exchanging the challenges and responses.

In our scheme, it is desirable to control the probability p_r that a node receives a challenge. This can be achieved if every node broadcasts its degree of connection d to its neighbors after the neighbor discovery phase and every neighbor then sets its probability to challenge this node as $p_c = p_r/d$.

After a node detects an impersonation attack, it knows one of its neighbors is compromised, although it does not know which one. In this case, it increases its challenge probability p_c. It can also send a notice encrypted with its pairwise keys to each neighbor, so that every node will be aware of the compromise. Finally, it sends a notice to the controller, encrypted with its individual key. The controller can then take any necessary actions to detect and recover from the compromise.

4. PERFORMANCE EVALUATION

In this section we analyze the computational and communication cost of our key establishment and key updating schemes. We note that the individual key of a node and the pairwise shared keys are usually not updated after the neighbor discovery phase, whereas the cluster key is updated if this node is a neighbor of the node being revoked and the group key is updated in every group rekeying. Therefore, we only consider the cost involved in updating the cluster keys and the group key.

Our rekeying protocol does not require the use of a specific routing protocol; however, its communication costs will

depend upon the routing protocol. In our analysis below, we assume that the rekeying protocol uses a spanning tree constructed as discussed in Section 3.2.5 for delivering the new group key to the nodes in the system.

4.1 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 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 while updating a cluster key is $\frac{2S_e}{N}$.

The number of decryption operations in the secure distribution of a group key is equal to the network size N because every node needs to decrypt once. Recall that we are using 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 average cost is at most 2 symmetric key operations per node.

From the above analysis, we know that the computational cost in a cluster and group rekeying 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. A larger N will further reduce this cost.

4.2 Communication Cost

The analysis of communication cost for a group rekeying event is similar to that of computational cost. For updating a cluster key, the average number of keys a node transmits and receives is equal to $(d-1)^2/(N-1)$ for a network of connection degree d and size N. For 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 [26], 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, for our inter-node authentication scheme, a node also keeps the commitment or the most recent AUTH key of each neighbor and its own one-way key chain. In a sensor network the packet transmission rate is usually very small. 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 reasonable length of key chain. After the keys in the key chain are used up, it can generate and bootstrap a new key chain. To avoid storing the entire key chain, we can deploy the optimization algorithm by Coppersmith and Jakobsson [4] to trade storage and computation cost. Their algorithm performs $O(\log_2 \sqrt{n})$ hashes per output element, and uses $O(\log_2 n)$ memory cells, where the size of each cell is slightly larger than that of a key and n is the length of the 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 SRAM 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 scheme can revoke node u from the group efficiently. 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 any sensor networks. Below we first consider in general what the adversary can accomplish after it compromises a sensor node. We then discuss some detailed attacks on routing protocols in Section 5.2.

First, because the individual key of a node is only shared between the node and the base station, obtaining this key usually does not help the adversary to launch attacks. Second, possessing the pairwise shared 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 the use of our inter-node authentication scheme. We note a salient feature of our protocol is its ability in *localizing* the possible damage, because after a node joins the network, it keeps a list of trusted neighboring nodes. Thus, the compromised node cannot establish trust relationship with any nodes except its neighbors, which means the adversary cannot jeopardize the secure links among any 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 secure message distribution. Moreover, possessing the group key does *not* enable the adversary to flood the entire network with malicious packets 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 [17] have studied various possible attacks on security of routing protocols for wireless sensor network. We now show how our schemes can defend against the attacks they described. Recall that in our scheme routing control information is authenticated by the inter-node 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 inter-node 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. For example, our probabilistic challenge scheme make the spoofing attacks difficult to go undetected. 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 *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 our schemes because the adversary does not have a network-wide authentication key (note that the group key in our scheme is only used for secure distribution of messages from the base station, not for authentication purpose). Our scheme can also prevent *Sybil attacks* [8]. In this attack, the adversary may replicate the compromised node and add multiple replicas into the network. A replica node then tries to establish pairwise keys with nodes that are not the neighbors of the compromised node with the help of the base station, if the base station does not know precisely the topology of the network. This attack may work in schemes [22] that use a Kerberos-like protocol for pairwise key establishment, but it will not work in our scheme because in our scheme every node knows its neighbors and we do not use the base station for pairwise key establishment.

5.2.1 Dealing with Wormhole and Sinkhole Attacks

The attack that is most difficult to detect or prevent is one that combines the *Sinkhole* and 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 [11] 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.

In our scheme, 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 (usually 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 the *authenticated* and *initial* 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 our schemes. Even so, it still cannot convince two distant nodes that they are neighbors after they have completed their *neighbor discovery* phases. 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. RELATED WORK

Stajano and Anderson discuss the various issues that arise for secure devices consisting of "peanut nodes" [23]. In particular, they propose that nodes bootstrap trust relationship through physical contact. Zhu et al [28] propose an efficient scheme for bootstrapping trust among mobile nodes based on the combination of TESLA [21] and one-way hash chain. Carman, Kruus and Matt have analyzed several approaches for key management and distribution in sensor networks [5]. 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* [2] discuss a rekeying scheme for periodically updating the group-wide traffic encryption key in a sensor network. However, they assume that the sensor nodes are tamper-free 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 [9] present a key management scheme for sensor networks based on probabilistic key predeployment. Chan et al [6] extend this scheme and present three new mechanisms for key establishment based on the framework of probabilistic key predeployment. Zhu et al [29] 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. We note most of these schemes only provide probabilistic security in the sense that the compromise of a fraction of sensor nodes may expose other nodes' keys with some probability. In contrast, our schemes provide deterministic security because the compromise of a fraction of sensor nodes does not reveal the pairwise keys between other nodes.

Perrig *et al* present security protocols for sensor networks [22]. 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 make it subject to Sybil attacks [8]. In contrast, in our scheme pairwise keys are established in a distributed fashion without the involvement of the base station.

Liu and Ning [19] present a multi-level key chain scheme for μTESLA. Karlof *et al* [16] describe TinySec, the link layer security mechanism that is part of the TinyOS platform. They also discuss the impact of different keying mechanisms on the effectiveness of in-network processing in sensor networks. In a recent paper [17], Karlof and Wagner discussed 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.

7. CONCLUSIONS

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

- 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. Consequently, 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 inter-node traffic 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 at the same time restricting the security impact of a node compromise to the immediate network neighborhood of the compromised node.
- The key establishment and key updating procedures used by LEAP are efficient and the storage requirements per node are small.
- LEAP can prevent or increase the difficulty of launching many security attacks on sensor networks.

8. ACKNOWLEDGMENTS

We would like to thank Fan Ye (UCLA) for helpful comments and discussions. We also thank the anonymous reviewers for their valuable comments.

9. REFERENCES

[1] R. Anderson, M. Kuhn. Tamper Resistance – a Cautionary Note. The Second USENIX Workshop on Electronic Com-merce Proceedings, Oakland, California, November, 1996.

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

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

[4] D. Coppersmith, M. Jakobsson. Almost Optimal Hash Sequence Traversal. In Finanical Cryptography (FC) 02.

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

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

[7] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246, January 1999.

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

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

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

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

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

[24] TinyOs. http://www.cs.berkeley.edu/ jhill/spec/index.htm.

[25] A. Woo and D. Culler. A Transmission Control Scheme for Media Access in Sensor Networks. In Proc. of MOBICOM '01, Rome, July 2001.

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

[27] 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.

[28] 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.

[29] S. Zhu, S. Xu, S. Setia, and S. Jajodia. Establishing Pair-wise Keys For Secure Communication in Ad Hoc Networks: A Probabilistic Approach. To appear in the 11th IEEE International Conference on Network Protocols (ICNP'03), Atlanta, Georgia, November 4-7, 2003.