LiSP: A Lightweight Security Protocol for Wireless Sensor Networks

TAEJOON PARK and KANG G. SHIN
The University of Michigan

Small low-cost sensor devices with limited resources are being used widely to build a self-organizing wireless network for various applications, such as situation monitoring and asset surveillance. Making such a sensor network secure is crucial to their intended applications, yet challenging due to the severe resource constraints in each sensor device. We present a *lightweight security protocol* (LiSP) that makes a tradeoff between security and resource consumption via efficient rekeying. The heart of the protocol is the novel rekeying mechanism that offers (1) efficient key broadcast without requiring retransmission/ACKs, (2) authentication for each key-disclosure without incurring additional overhead, (3) the ability of detecting/recovering lost keys, (4) seamless key refreshment without disrupting ongoing data encryption/decryption, and (5) robustness to inter-node clock skews. Furthermore, these benefits are preserved in conventional contention-based medium access control protocols that do not support reliable broadcast. Our performance evaluation shows that LiSP reduces resource consumption significantly, while requiring only three hash computations, on average, and a storage space for eight keys.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Security and Protection*; K.6.m [**Management of Computing and Information Systems**]: Miscellaneous—*Security*

General Terms: Design, Security

Additional Key Words and Phrases: Authentication, key management, lightweight security, sensor networks

1. INTRODUCTION

Sensor networks are usually built with a large number of small, inexpensive, battery-powered devices that have limited residual energy, computation, memory, and communication capacities. Such sensor networks can be used for various applications typified by the well-known pursuit-evasion game (PEG)

The work reported in this paper is supported in part by the ONR under Grant No. N00014-99-1-0465, the DARPA administered by AFRL under contract F33615-02-C-4031, and NSF under Grant CCR-0329629.
Authors' address: T. Park and K. G. Shin, Real-Time Computing Laboratory (RTCL), Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109-2122; email: {taejoonp,kgshin}@eecs.umich.edu.

[Hespanha et al. 1999; Vidal et al. 2002], in which a group of *pursuers* attempts to track and capture moving *evaders*. A sensor network is integrated into the PEG to augment the "sensibility" of pursuers.

Sensor networks must meet several operational challenges, such as energy efficiency in terms of maximizing the lifetime of sensor networks; scalability to a large number (thousands to millions) of nodes; survivability in certain environments where sensors are subject to compromise, capture, and manipulation by adversaries; support for dynamic addition/removal of sensors;[1] and robustness to spontaneous interferences, collisions, and packet losses. Moreover, they are vulnerable to serious security attacks. Unlike the wired counterparts, sensor networks do not require any physical contact for communication, and hence, an adversary with a simple radio receiver/transmitter can easily eavesdrop conversations, inject/modify packets, and mount denial-of-service (DoS) attacks. These challenges, along with the severe resource constraints in each sensor node, limit both the security and the performance of a sensor network. Confidentiality, data integrity, and authentication services must be supported to prevent adversaries from compromising the sensor network, but advanced cryptography cannot be used by resource-poor sensor nodes. It is, therefore, important to make a good tradeoff between the levels of security and resource consumption.

A sensor device usually cannot use nontrivial cryptography like public-key algorithms due mainly to its insufficient resources. The symmetric-key ciphers and cryptographic hash functions, which are orders-of-magnitude cheaper and faster, would be a better choice for sensor nodes. Moreover, data packets in sensor networks are generally small. A desirable property in this environment is that the size of the ciphertext should be the same as that of the plaintext. These requirements suggest the use of a stream cipher as the underlying encryption engine. For example, SPINS [Perrig et al. 2001] realizes the stream cipher by running the RC5 block cipher in the counter mode, Burnside et al. [2002] use RC5 in the output feedback mode, and the IEEE 802.11 Wired Equivalent Privacy (WEP) [IEEE 1997] uses the RC4 stream cipher.

However, it is well known that stream ciphers are vulnerable to keystream[2] reuse. This weakness allows attacks against stream ciphers that succeed irrespective of the symmetric-key size. For example, WEP prefixes each encrypted packet with a per-packet IV, but, due to the limited IV space (24 bits), it is vulnerable to a number of practical attacks as reported in Borisov et al. [2001], Walker [2000], and McGrew and Fluhrer [2000]. To remove the keystream-reuse problem, SPINS forces both communicating parties to maintain the IV separately, instead of including IV in data packets, while updating the key after the IV wraps around. Unfortunately, this design choice creates the following new problems:

—Lossy wireless links may cause IVs loss of synchronization, and in such a case, communication will remain disabled until IVs are resynchronized.

[1]This is necessary to expand the network coverage area or replace faulty/subverted nodes.
[2]The keystream is generated as a function of the symmetric key and the initialization vector (IV), and is XORed with the plaintext to produce the ciphertext.

— It cannot protect the network against replay attacks and incurs an additional overhead of maintaining IV states of the other sensor devices.
— The rekeying overhead increases rapidly as the network size grows.

Problems with the schemes in WEP and SPINS emanate from the fact that they solely control IVs without refreshing the key, or use implicit IVs and triggered rekeying. It is, therefore, important to refresh the symmetric key as often as needed while keeping small-length IVs, not only to remove keystream collisions but also to improve performance.

The security for sensor networks hinges on a *group communication model*: authorized sensors in the network share a symmetric key that is used to encrypt communication messages; new sensors *join* the network after their deployment; and the compromised sensors are forced to *leave* the network. In this model, forward and backward confidentiality[3] [Wallner et al. 1999] should be provided via rekeying, in which the shared key is changed/redistributed whenever a sensor joins or leaves the network. For its proper operation, rekeying must be protected by the following mechanisms. First, the master secret, used for securing the shared-key updates, must be predeployed to each sensor using tamper-proofing techniques [Carman et al. 2000] or a ring of keys [Eschenauer and Gilgor 2002]. Second, intrusion detection systems (IDSs) [Kumar and Spafford 1995; Ilgun et al. 1995; Bass 2000; Zhang and Lee 2000; Zhang et al. 2001] must be used as sensors are likely to be compromised because (i) use of tamper-proofing techniques is limited by the low-cost requirement for sensor devices, and (ii) only computationally inexpensive cryptography can be employed.

For scalability, the entire network is typically divided into multiple groups, each with its own symmetric key [Mittra 1997] or with a key shared among all groups [Setia et al. 2000]. Carman et al. [2000] conducted a broad survey of group-determination algorithms and the associated group rekeying protocols. Group rekeying protocols can be either *reactive* or *periodic*. The reactive protocol [Harney and Muchenhirn 1997; Wallner et al. 1999; Wong et al. 1998; Chang et al. 1999] renews the key upon a member's join/leave. This approach, however, does not attempt to reduce the frequency of rekeying that causes high rekeying overhead in large and/or dynamic groups. By contrast, the periodic protocol [Setia et al. 2000, 2002; Li et al. 2001] refreshes keys periodically to decouple the frequency of rekeying from the group size and dynamics, and hence, scales well to large groups. Yang et al. [2001] have shown that periodic rekeying reduces both processing and communication overheads of the key-server (KS)[4] and improves the scalability and performance over the reactive rekeying. Moreover, severe resource constraints in each sensor node and the requirement for a large number of sensors in the network make it necessary to limit the frequency of rekeying so as to reduce its overhead. In such a case, periodic rekeying might be preferable to reactive protocols [Setia et al. 2000; Basagni et al. 2001].

[3]New members joining the network should not be able to access the packets transmitted before their joining and those having left the network should not be able to access the packets communicated after their departure.
[4]The KS is responsible for distributing a new key within its group.

The rekeying must ensure reliable distribution of keys. The Time Division Multiple Access (TDMA) protocol can provide a reliability service to the key-management layer. However, Ye et al. [2002] argued that TDMA is unsuitable for sensor networks as it is difficult to manage dynamic groups and control inter-group communications and interferences. Most protocols used/proposed for sensor networks [Singh and Raghavendra 1998; Woo and Culler 2001; Ye et al. 2002] are essentially the Carrier Sense Multiple Access (CSMA) protocol. To achieve reliable key distribution in the CSMA protocol, one may perform multiple unicasts with handshakes (RTS/CTS/Data/ACK), but this suffers from high latency and excessive control traffic. Broadcasting keys eliminates these problems, but, since the CSMA protocol does not provide any means of recovering lost frames, the broadcast reliability is degraded due to the increased probability of frame losses as a result of frame collisions. Several protocols [Pagani and Rossi 1997; Tang and Gerla 2000; Tourrilhes 1998; Sun et al. 2002] have been proposed to improve the CSMA's broadcast reliability. Unfortunately, they still introduce significant control traffic, without guaranteeing 100% reliability. We, therefore, need a key-management protocol that reliably coordinates the key-distribution service.

In this paper, we propose a *lightweight security protocol* (LiSP) that is equipped with key renewability and makes a tradeoff between security and resource consumption. The heart of LiSP is a novel rekeying protocol that (1) periodically renews the shared key to solve the keystream-reuse problem and maximize scalability/energy efficiency. and (2) supports reliable key distribution. The rekeying protocol has the following salient features:

— efficient key broadcasting without retransmission/ACKs;
— implicit authentication for new keys without incurring additional overhead;
— ability of detecting/recovering lost keys;
— seamless key refreshment without disrupting ongoing data transmission; and
— robustness to clock skews among nodes.

These features make LiSP very flexible in that it only requires very loose time synchronization, and does not stress the underlying network/link layers, that is, not requiring reliable broadcast at the link layer. LiSP is also energy efficient and robust to DoS attacks, since it does not require any retransmissions or other control packets. To our best knowledge, there is no previous work that effectively handles all of these issues.

We propose a joint authentication and recovery algorithm for rekeying, in which the KS periodically broadcasts a new key well before its use for encryption/decryption, and a client node first authenticates the received key and then recovers all previously missed keys, if any. The proposed algorithm relies on the unique properties of the cryptographic one-way function. It is efficient in that each node buffers keys only, as compared to TESLA [Perrig et al. 2001], which buffers all the received data packets until the node receives an error-free key. LiSP also uses double buffering of keys for their seamless,

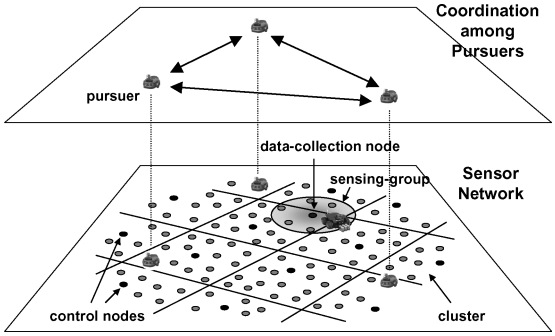

Fig. 1. The PEG framework consisting of two wireless networks, one for sensors and the other for pursuers.

robust refreshment: while the key in one slot is being used for data encryption/decryption, the next key will be written to the other slot.

The rest of the paper is organized as follows. Section 2 presents a system architecture for PEG-like applications. Section 3 summarizes possible security attacks and an intrusion model, while Section 4 describes the details of LiSP. Section 5 presents the results of our performance evaluation. Finally, the paper concludes with Section 6.

2. SYSTEM ARCHITECTURE

Prototypical applications of a sensor network include: (i) PEG [Hespanha et al. 1999; Vidal et al. 2002], in which a group of pursuers track and capture moving evaders based on the information collected and processed by the sensor network; (ii) a common reference grid, in which the sensor network collects and maintains information for a discovery service or a distributed directory service to locate critical services; (iii) a shooter localization system [Duckworth et al. 1996], in which sensors detect the acoustic shock wave of a bullet as it travels through the air; and (iv) habitat and environmental monitoring systems [Mainwaring et al. 2002], in which sensors are deployed to collect data without incurring disturbance effects (e.g., by human).

PEG-like applications are realized on two wireless networks, one for connecting sensors and the other for connecting pursuers, as shown in Figure 1. The sensor network typically covers a wide area, requiring thousands or even millions of sensors, each of which is capable of detecting (part of) an object moving nearby. On top of the sensor network, a separate wireless network of pursuers is formed, for example, to build a terrain map and cooperate with one another to capture/kill evaders based on the information collected from sensors.

Usually, there exists significant heterogeneity between sensors and pursuers. Sensors range from *Motes* [Hill et al. 2000; Crossbow 2003] with an 8-bit CPU running at 4 MHz, 128 KB of program memory, 4 KB of RAM, 512 KB of serial flash memory and two AA batteries, to those with more powerful CPUs like MIPS R4400 and larger memory capacity. Sensors in this range usually have limited battery energy, computation, memory, and communication capabilities.

In contrast, pursuers, such as unmanned aerial vehicles and unmanned ground vehicles, do not have such resource limitations. Each pursuer is equipped with the same radio receiver/transmitter as sensors, as well as a more powerful RF interface to communicate with other pursuers.

The sensor network includes (i) *data-collection* nodes, which collect/store sensed data, and process and make it available to pursuers, and/or (ii) *control* nodes, which coordinate (multihop) routing among sensors and broadcast commands to sensors. Clusters in this two-tier routing system rely on control nodes, called *cluster-heads*, for managing cluster topology, routing information, pursuers' locations, and so on. Clusters may be statically formed according to geographic grids [Li et al. 2000]. Sensing groups, which are formed around evaders to aggregate/disseminate sensor data about evaders, dynamically elect data-collection nodes, as in Estrin et al. [1999], Madden et al. [2002], Ye et al. [2002], and Bonnet et al. [2001]. In light of the group communication model, we will henceforth refer to each cluster or sensing group as a *group*, and each control or data-collection node as a *group-head* (GH).

Figure 1 illustrates the role of GHs in the sensor network: (i) each GH collects data about evaders, and (ii) all GHs cooperate in sending/receiving data to/from the pursuer (intergroup communication) as well as communicating with sensors within their own group (intragroup communication). So, the communication between a sensor and the pursuer is made in three steps: (1) a source s sends data to its GH h_1; (2) h_1 relays the data to another GH h_2 that knows the location of the receiver d; and finally, (3) h_2 forwards the data to d.

There exist two types of intragroup communication, one from GH to sensors and the other from a sensor to GH. GH either unicasts specific commands to a sensor or broadcasts control packets, such as beacons, queries, and requests, to all of its sensors, while each sensor unicasts data to its GH. Since sensors are assumed immobile, it suffices for them to use a table-driven routing protocol, under which each GH acts as a coordinator, maintaining the routing topology, and each sensor within a cluster stores only one entry, the next-hop information, in its routing table to reach its GH.

3. SECURITY ATTACKS

Adversaries can be classified as *passive* or *active*: passive attackers only eavesdrop conversations on the network, while active attackers inject packets into the network in addition to eavesdropping. Since sensor networks may be deployed in a hostile environment, we should assume more powerful adversaries. An adversary's attempt to disrupt, subvert, or destroy the sensor network, belongs to a broad category of DoS attack that diminishes or eliminates the network's ability to perform its normal function. Wood and Stankovic [2002] summarized plausible tools for DoS attacks as follows:

—*jamming* that interferes with the operating radio frequencies;
—*collisions* that are induced on ongoing packet transmissions;
—*exhaustion* that forces the link layer to repeat packet retransmission; and
—*vulnerabilities* of existing protocols.

A captured sensor may be scrutinized and modified by the adversary. The tamper-resistance technology [Carman et al. 2000; Anderson and Kuhn 1996] is widely used as a countermeasure for this. But, as stated in Anderson and Kuhn [1996], tamper-resistant hardware is not always absolutely safe and there exist various tampering schemes, such as microprobing, glitch attacks, and cipher instruction search attacks.

Possible security attacks we assume are very general: an attacker can eavesdrop, forge, modify, and delete any information. It can also mount offline dictionary attacks for future break-ins, man-in-the-middle attacks, replay attacks, resource-consumption (or DoS) attacks. We also assume that an attacker can take over any sensor node within the network, because perfect tamper-proofing is too expensive to be built into low-cost sensor devices. It is, therefore, reasonable to assume that any secret can be securely preserved from attackers only for a certain period of time.

Compromising a single node means that all nodes within its communication range can be blocked/denied from receiving and/or sending/relaying any information. So, we must minimize the effects of a compromised node on the rest of the network, that is, the single compromised node should not be allowed to enable subversion of the entire network.

4. LIGHTWEIGHT SECURITY PROTOCOL

LiSP aims to offer a lightweight security solution for a large-scale network of resource-limited sensor devices. For scalability to a large number of sensors, LiSP decomposes the entire network into clusters and/or sensing groups and selects a GH for each of them, as described in Section 2.[5] LiSP addresses the following two main questions associated with the device's resource constraints:

Q1. How to combine security with other services, such as routing, sensor data aggregation/dissemination, and location services?

Q2. How to make a tradeoff between security and resource consumption?

To address the first question, LiSP introduces the notion of KS, which controls the security of a group. For a sensor network that consists of multiple groups, LiSP designates GHs as KSs. The wireless networks for connecting pursuers would also be partitioned into groups, each of which elects the KS among its members. So, without loss of generality, we can assume the existence of one KS per group. LiSP also uses a KS for the network (KSN) that coordinates KSs in rekeying for intergroup communications.

For the security tradeoff, LiSP (i) uses a stream cipher for its cheap and fast processing, and (ii) supports periodic renewal of keys with inexpensive cryptographic hash algorithms. It is reliable, and works well with the conventional CSMA protocols that do not support reliable broadcast. Moreover, LiSP requires only very loose time synchronization among group members.

[5]Each group (cluster) will be reasonably sized. Accordingly, the larger the network gets, the more groups (clusters) LiSP creates.

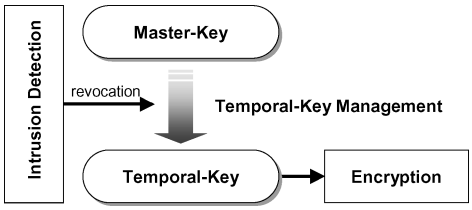

Fig. 2. The key hierarchy for LiSP.

LiSP achieves the following goals in protecting security-critical information from attackers.

—*Confidentiality*: keeps data from being eavesdropped, and ensures that an attacker will not acquire any information about the plaintext, even if it sees multiple encrypted versions of the same plaintext.
—*Data integrity*: prevents tampering with the transmitted data.
—*Access control*: protects and controls access to the network.
—*Availability*: protects the network from interruptions in service.
—*Key renewability and revocability*: protects the network from compromised nodes, if any.

In this section, we detail the architecture of LiSP, the rekeying protocol, the message encryption/authentication layer, and describe algorithms and tradeoffs.

4.1 The LiSP Architecture

For key renewability, LiSP implements a key hierarchy as shown in Figure 2. This hierarchy defines two keys: (i) a *temporal key* (TK) for encrypting/decrypting data packets; and (ii) a sensor-specific *master key* (MK) that is used by KS to unicast TK to an individual sensor. Under the symmetric-key cryptography, a TK should be shared by all group members (for intragroup communications) and refreshed periodically to ensure forward and backward confidentiality as well as elimination of keystream collisions. Using its group-based architecture, LiSP achieves scalable and distributed rekeying, since membership changes[6] in a group do not affect the other groups in the network.

The KS executes *entity authentication*[7] with a new sensor joining the group, and if successful, grants a membership to the sensor by storing the sensor's MK in its database and then transmitting the current TK. MKs for sensors will be stored in tamper-resistant hardware, but we assume limited tamper-resistance built in low-cost sensor devices. This means that an attacker may access MKs of the subverted sensors.

[6]The group membership changes if a new sensor joins the group or if an existing member leaves the group. The latter event occurs when the member is compromised.
[7]The entity authentication between two nodes verifies each other's identity/authenticity. It typically relies on trusted third parties such as distributed certificate authorities [Shamir 1979; Zhou and Haas 1998; Kong et al. 2001; Zhou et al. 2002].

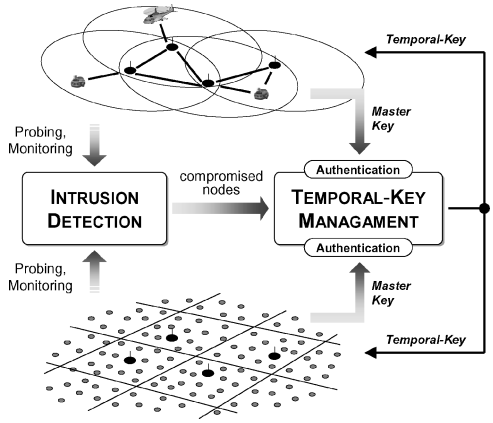

Fig. 3. The LiSP architecture.

As shown in Figures 2 and 3, there are two main components associated with the key hierarchy: intrusion detection, which probes/monitors network activities to uncover compromised nodes, and TK management, which protects network traffic from attacks by rekeying TK periodically. Since it is almost impossible to safeguard the network against all possible attacks, it is important to introduce a second line of defense, that is, LiSP uses an IDS [Kumar and Spafford 1995; Ilgun et al. 1995; Bass 2000; Zhang and Lee 2000; Zhang et al. 2001] to probe/monitor for anomalies in the network. Since each GH that serves as KS, is a traffic concentration point of the corresponding group, it will be equipped with an IDS to monitor the ongoing traffic. However, in sensor networks, all useful information is local at some point of time, to the group(s) near the evader. Due to this distinct feature, a single point of failure at the KS might cripple the entire network. To avoid this problem, the IDS is organized hierarchically: each KS is in charge of monitoring sensors within a group, while a more powerful IDS running, for example, on pursuers, watch out for possible KS compromises. In our framework depicted in Figure 1, both KSs and pursuers are more capable and have more resources than usual sensors, and hence, the IDSs resource consumption, unless it is excessive, should not be an issue.

Once a compromised sensor is identified by the KS, the TK manager disables the sensor and renews the TK in the next update cycle. If a KS is found to have been compromised, LiSP either (i) elects a new KS for the group, or (ii) redistributes member sensors to the neighboring groups.

The TK manager, running on the KS, renews TK for the group. Due to the severe resource limitation in each sensor device, TK rekeying should be lightweight and conserve resources as much as possible. Our approach to meeting this requirement is to renew TK periodically using (not necessarily reliable) broadcast, instead of using triggered and unicast/retransmission-based renewal. Periodic rekeying of the TK is crucial to counter keystream-reuse attacks and improve scalability/energy efficiency of group rekeying. The proposed TK management has the following salient properties.

— efficient TK broadcasting without relying on retransmissions/ACKs;
— implicit authentication of TKs;
— fault tolerance by recovering lost TKs;
— seamless TK rekeying without disrupting ongoing data transmissions; and
— robustness to internode clock skews.

These properties of LiSP have yielded high-performance TK rekeying: it minimizes the number of control packets generated in the network and reduces the size of each control packet. The TK management will be detailed next.

4.2 TK Management

The challenges in TK management are how to enable all nodes to (i) acquire a new TK efficiently, securely, and reliably, and (ii) switch to the new TK without disrupting the ongoing data transmission. Note that, with the symmetric-key ciphers, it is difficult to refresh TK seamlessly, as it requires the same key to be possessed by both communicating parties. To address the first challenge, TK distribution must be secure and fault tolerant: the "secure" part relates to confidentiality and authenticity of TKs, and the "fault tolerant" part means the ability to restore lost TKs. The second challenge requires seamlessness and weak internode time synchronization.

The proposed TK management meets the above challenges/requirements while incurring low overhead. The main ideas of the proposed protocol are to: (1) generate a sequence of TKs by utilizing the cryptographic one-way function, similarly to that of S/KEY [Haller 1995]; (2) distribute each TK well before it is used for encryption/decryption; (3) perform TK buffering in all sensors in the group; and (4) verify the authenticity of the received TK and detect/recover missing TKs using the other stored TKs.

To ensure secure TK distribution, the KS initiates TK management by encrypting, authenticating, and transmitting a control packet that includes the length t of the key-buffer (for TKs), an initial TK, and the TK-refreshment interval, T_{refresh}.[8] Then, once every T_{refresh}, the KS encrypts and broadcasts a control packet that contains a future TK. Note that the latter does not include a message authentication code (MAC) for TK. Thanks to the cryptographic one-way property of TK sequence, receivers can determine whether or not the received TK belongs to the same key sequence as those stored in the buffer, thus verifying the TKs authenticity. This procedure, called *implicit authentication*, reduces the size of control packet significantly, because the size of MAC (e.g., 128-bit in MD4) is as large as that of fields to be protected.

TK refreshment must tolerate TK losses caused by a noisy channel. A retransmission-based reliability mechanism cannot be used because it will generate too many control packets and/or result in very high latencies. It would be more efficient and more desirable if nodes could automatically restore TKs,

[8]The larger t, the more fault-tolerant LiSP becomes at the expense of larger key-buffer space. T_{refresh} is a design parameter for making a tradeoff: the shorter T_{refresh}, the higher overhead and the smaller rekeying latency. T_{refresh} should be shorter than the interval that ensures collision-free keystreams at sensors' maximum packet-generation rate.

rather than asking the KS for retransmission of the lost TKs. Thus, we need a lightweight mechanism to detect the loss of TKs and restore up to t lost TKs in a cryptographically secure way. LiSP achieves this based on a one-way key sequence.

The last two requirements—seamlessness and weak/loose time synchronization—are met by equipping each node with two key-slots to be used concurrently. While the TK in one key-slot is being used for data encryption, the next TK is written into the other key-slot. Then, at the midpoint of the refresh interval, the node switches the active key-slot to the one with the new TK. In summary, TK management stores/utilizes $t + 2$ TKs, that is, t TKs in the key-buffer are for authentication and recovery of lost TKs and 2 TKs in the key-slots for encryption/decryption.

4.2.1 *Control Packets.* LiSP uses three different control packets: `InitKey`, `UpdateKey`, and `RequestKey`. `InitKey` is used by the KS to initiate TK refreshment, and contains, t, the number of lost TKs that can be recovered; an initial TK; T_{refresh}, TK refreshment interval; and MAC. The KS unicasts this packet to each group member whenever it wishes to (re)configure TK management with a given set of parameters. `UpdateKey` is used by the KS to periodically broadcast the next TK in the key-sequence, and contains a new TK. `RequestKey` is used by individual nodes to explicitly request the current TK in the key-sequence. This packet is generated when a node failed to receive TKs over t key-refresh intervals.

`UpdateKey` is broadcast to all group members, while `InitKey` and `RequestKey` are unicast between KS and an individual member. Therefore, we use notation `InitKey`(m) and `RequestKey`(m) to specify the unicast between KS and node m. `InitKey`(m) uses node m's master secret, MK_m, for encryption and message authentication, where MK_m is shared only between the KS and node m via entity authentication. By contrast, `UpdateKey` uses currently active TK for encryption.

4.2.2 *Initial Setup.* The KS precomputes a one-way sequence of keys, $\{\text{TK}_i \mid i = 1, \ldots, n\}$, where n is chosen to be reasonably large (e.g., 100). It picks the last key, TK_n, randomly and computes the entire key sequence using the cryptographic one-way function H, where each TK_i is derived as $\text{TK}_i = H(\text{TK}_{i+1})$, $i < n$, or equivalently, $\text{TK}_i = H^{n-i}(\text{TK}_n)$, $H^j(x) = H^{j-1}(H(x))$ and $H^0(x) = x$. Then, at time t_{start}, the KS starts TK management by unicasting to every node m in its group, `InitKey` along with t, TK_{t+2}, T_{refresh} and MAC:[9]

$$KS \rightarrow m\text{: } E_{MK_m}(t \mid \text{TK}_{t+2} \mid T_{\text{refresh}}) \mid MAC(t \mid \text{TK}_{t+2} \mid T_{\text{refresh}}),$$

where $E_K(x)$ is the encryption of x with key K, and $MAC(y)$ generates a message digest for y using the cryptographic hash function.

On receiving the `InitKey`(k) packet, node k (i) clears all previous TKs; (ii) allocates a key buffer of length t (`kb[t]`, ..., `kb[1]`), and two key-slots; (iii) computes keys, $\text{TK}_{t+1}, \ldots, \text{TK}_1$, from TK_{t+2}; (iv) stores $\{\text{TK}_{t+2}, \ldots, \text{TK}_3\}$ and $\{\text{TK}_2, \text{TK}_1\}$

[9]The initial TK is not TK_1 but TK_{t+2}. t and T_{refresh} are network-wide parameters shared by all groups. Since receivers cannot recover from `InitKey` loss, the KS should use external reliability services like retransmissions and handshakes (RTS/CTS/Data/ACK).

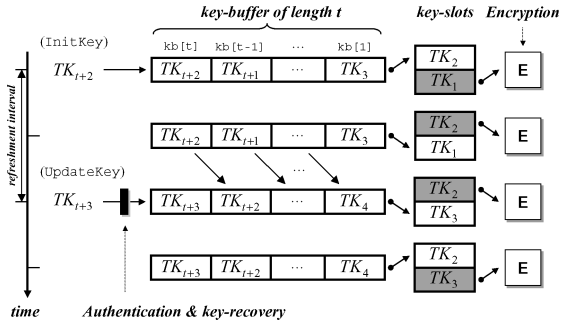

Fig. 4. TK Management: initial setup and rekeying.

in the key-buffer and key-slots, respectively; (v) activates TK_1 for data encryption; and finally (vi) sets `ReKeyingTimer` that expires after $T_{\text{refresh}}/2$. When the timer expires, the node (1) switches the active key to TK_2, thus making the key-slot (used to store TK_1) available for the next encryption key TK_3, and (2) sets `ReKeyingTimer` to expire after T_{refresh} for future key switching. Figure 4 shows how the node copies TKs into the key-buffer and key-slots, and switches the active-key after receiving TK_{t+2}.

4.2.3 *Re-keying.* After the initial setup, the KS periodically discloses TKs, starting with TK_{t+3} to all nodes in the group. That is, at time $t_{\text{start}} + i \cdot T_{\text{refresh}}$, the KS broadcasts `UpdateKey` packets containing TK_{i+t+2}, $i = 1, \ldots, n-t-2$:

$$\text{KS} \Rightarrow \text{ group: } E_{TK_{i+1}}(TK_{i+t+2})$$

where TK_{i+1} is the active encryption key at the time when `UpdateKey` is broadcast.[10]

Upon receiving the `UpdateKey` packet, a node processes it as follows. If it had received the same packet previously or the packet is not from its own KS, the packet is discarded. Otherwise, it rebroadcasts the packet to all of its neighbors and

(1) shifts the stored TKs, that is, `kb[1]` to the inactive key-slot and `kb[i]` to `kb[i-1]`, for $2 \le i \le t$;
(2) executes TK authentication and recovery on the received TK, as described in Section 4.2.4; and
(3) if successful, copies the received TK to `kb[t]` else discards TK.

Whenever `ReKeyingTimer` expires, the node (i) switches the active-key to the TK in the other key-slot, and (ii) sets `ReKeyingTimer` to expire after T_{refresh} elapses. Figure 4 illustrates how the key-buffer and key-slots are updated after reception of TK_{t+3} and expiration of `ReKeyingTimer`.

4.2.4 *Authentication and Recovery of Lost TKs.* After receiving the `UpdateKey` packet, each node verifies if the received TK is authentic, and, if

[10]The group members may reserve an IV value for `UpdateKey` packets to protect the `UpdateKey` packet from keystream collisions.

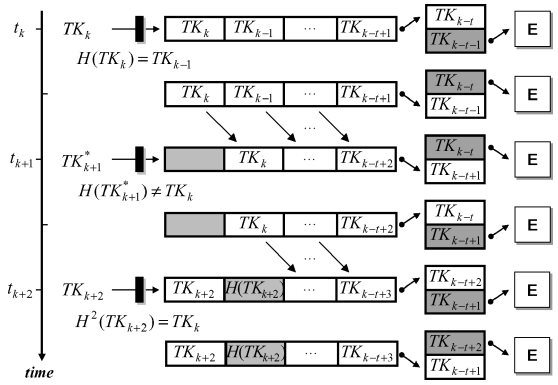

Fig. 5. TK management: authentication and recovery of lost TKs.

so, recovers lost TKs, if any, using the received TK. To recover from TK losses, the key-buffer stores $\leq t$ TKs. Let $\mathrm{TK}^\dagger_r, \ldots, \mathrm{TK}^\dagger_1$ denote r ($\leq t$) TKs in the key-buffer `kb[r]`, ..., `kb[1]`, respectively, and TK_k is the received TK. Then, there are $e = t - r$ empty slots in the key-buffer. It follows from the property of the one-way key sequence that $H(\mathrm{TK}^\dagger_r) = \mathrm{TK}^\dagger_{r-1}$, ..., $H(\mathrm{TK}^\dagger_2) = \mathrm{TK}^\dagger_1$. Since TK_k belongs to the same one-way key sequence, it should meet the following two conditions.

—*Authenticity condition*: TK_k is authentic if $H^{e+1}(\mathrm{TK}_k) = \mathrm{TK}^\dagger_r$, where $r = t - e$.

—*Fault-tolerance condition*: if $1 \leq e \leq t$, there are e lost TKs, $\{\mathrm{TK}^\dagger_{t-i+1} = H^i(\mathrm{TK}_k) \mid 1 \leq i \leq e\}$.

TK authentication and recovery uses these two conditions, and works as follows:

— Compute e and $\{ H^i(\mathrm{TK}_k) \mid i = 1, \ldots, e+1 \}$.
— If $H^{e+1}(\mathrm{TK}_k) \neq \mathrm{TK}^\dagger_r$, discard TK_k.
— Otherwise, if $e \geq 1$ then copy $\{H^i(\mathrm{TK}_k) \mid i = 1, \ldots, e\}$ to the key-buffer.

Figure 5 illustrates how authentication and key-recovery works. On receiving TK_k, the receiver computes $e = 0$ and $H(\mathrm{TK}_k) = \mathrm{TK}_{k-1}$, and hence, TK_k is authentic and there are no TK losses. At time t_{k+1}, the node receives $\mathrm{TK}^*_{k+1} (\neq \mathrm{TK}_{k+1})$, verifies that $H(\mathrm{TK}^*_{k+1}) \neq \mathrm{TK}_k$ ($e = 0$), and drops TK^*_{k+1}. The key-buffer thus stores $(t-1)$ TKs (or $e = 1$). At a later time t_{k+2}, the node receives TK_{k+2} and verifies that the received TK is the correct key by computing $H^2(\mathrm{TK}_{k+2}) = \mathrm{TK}_k$, and recovers $\mathrm{TK}_{k+1} = H(\mathrm{TK}_{k+2})$. Likewise, other TK arrivals will be processed.

LiSP can also detect and correct the situation where a receiver misses TK-disclosures. Consider the case when the node failed to receive TK at time t_{k+1} in Figure 5. This can be handled by the `ReKeyingTimer` event triggered at time $t_{k+1} + T_{\mathrm{refresh}}/2$: the event handler checks if TKs in the key-buffer has been right-shifted because the last `ReKeyingTimer` event, and, if not, shifts `kb[1]` to the inactive key-slot and `kb[i]` to `kb[i-1]`, for $2 \leq i \leq t$.

Use of the one-way key sequence for recovery of the lost keys in LiSP is similar to that of TESLA. However, the two protocols differ significantly in the

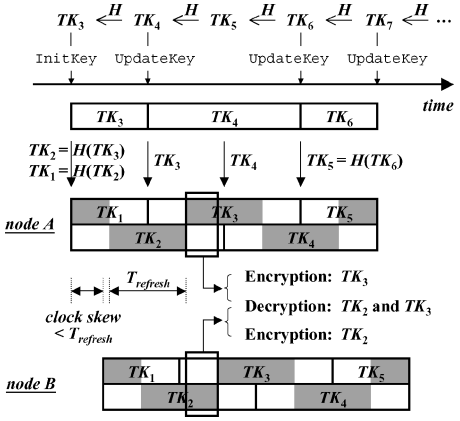

Fig. 6. TK Management ($t = 1$): robustness to clock skews.

way the one-way key sequence is applied. TESLA buffers all of the received packets until it receives an error-free key, and hence, if it misses several key disclosures, TESLA suffers from high latency and large buffer size. In contrast, LiSP buffers only $t + 2$ TKs, so the buffer size is small and fixed. Moreover, the KS distributes TKs well before its use for data encryption, and thus, the missed TKs will not disrupt the ongoing data transmission.

4.2.5 *Robustness to Clock Skews.* The proposed TK management is robust to clock skews among group members. As an example, Figure 6 illustrates, in the time domain, key-slots of two nodes, A and B, when there exists a clock skew between the two nodes. Thanks to the authentication and key-recovery, loss of up to t TK losses will not affect the key-slot activation. Let c_j be the mapping from clock time to real time at node j, where $c_j(T) = t$ means that at clock time T, the real time is t. Then, the clock skew between A and B is given by $\delta = |c_A(T) - c_B(T)|$. The figure shows that seamless TK rekeying is preserved, if $\delta < T_{\text{refresh}}/2$. During the marked period in Figure 6, A uses TK_3 for encryption, while B still uses TK_2 due to the clock skew between A and B. However, since both A and B have the same decryption key pair, $\{\text{TK}_2, \text{TK}_3\}$, they can communicate with each other during this period. In general, LiSP can sustain the worst-case clock skew of $T_{\text{refresh}}/2$ (i.e., $\max\{\, |c_A(T) - c_B(T)| : \forall A, B \,\} < T_{\text{refresh}}/2$).

Moreover, TK distribution from KS to group members also tolerates clock skews of up to $T_{\text{refresh}}/2$. That is, TK_k will be processed correctly if it arrives at the node during the time interval $[t_k - T_{\text{refresh}}/4,\ t_k + T_{\text{refresh}}/4]$, where t_k is the scheduled time when TK_k is disclosed.

4.2.6 *Reconfiguration.* The KS will reconfigure the TK management at the time of next rekeying, if (1) existing group members have been compromised; (2) all n TKs have been disclosed; (3) a new node has joined the group; or (4) a member has explicitly requested TK, because it missed more than t TK-disclosures. The first two events force all group members to be reconfigured, whereas the third and fourth events allow reconfiguration of the requesting

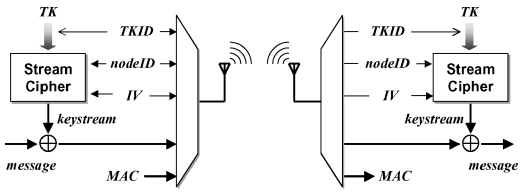

Fig. 7. Message transmission and reception in LiSP.

node only. The required actions for each event are summarized as follows:

(1) The KS revokes compromised nodes, and if TK_{k-1} has been disclosed previously, discloses TK_{k+t+2}, instead of TK_k, using `InitKey`. This makes all previous TK-disclosures (up to TK_{k-1}) futile.
(2) KS computes a new key-sequence $\{TK'_i \mid i = 1, \ldots, n\}$, and unicasts `InitKey` with TK'_{t+2} to all members.
(3) The KS performs entity authentication with the new node, and if successful, sends the current configuration via an `InitKey` packet.
(4) The KS sends the requesting node an `InitKey` packet containing the current configuration.

4.2.7 *Tradeoffs.* The performance of the proposed rekeying scheme in terms of communication overhead, depends on the size of the group. As the group size increases, each group will have more chances of getting compromised per TK-disclosure, hence increasing the rate of reconfigurations. Since LiSP achieves significant performance gain over conventional rekeying by broadcasting TK-disclosures with no retransmissions, more frequent reconfigurations effectively mean poorer performance. In contrast, larger groups will have more efficient broadcasting of a single TK-disclosure, improving the performance. Therefore, we can make a tradeoff between these two, and there exists an optimal group size that maximizes the overall performance.

4.3 Message Encryption/Decryption

For intragroup communications, LiSP encrypts messages with the stream cipher and the currently active TK. Since each node has two (even and odd) key-slots, LiSP uses a 1-bit *keyID* to tell which of the two TKs to use. LiSP also includes an ID of the sender and a per-packet IV to counter the keystream-reuse problem.

Figure 7 shows the encryption/decryption and authentication of messages from the sender, s, to the destination, d. The message transmission at the sender side proceeds as follows. First, based on the TK referenced by *keyID*, *nodeID* of the sender s, and the current IV, s generates a keystream, *keystream*(TK, *nodeID*, *IV*), and then XORs the keystream with plaintext, P, to build a ciphertext, C. Second, s computes a MAC, *mac*, to protect *keyID*, *nodeID*, IV, and P, where $mac = MAC(keyID|nodeID|IV|P)$. Finally, s transmits to d the following information over the wireless link:

$$s \rightarrow d : \quad keyID \,|\, nodeID \,|\, IV \,|\, C \,|\, mac.$$

The data message/packet reception at the destination simply reverses the above process. First, the keystream, $keystream(\text{TK}, nodeID, IV)$, is regenerated using the TK pointed to by *keyID*, and then XORed with C to recover the plaintext P'. Second, d checks the integrity of P' by computing $mac' = MAC(keyID|nodeID|IV|P')$ and comparing mac' with the received mac. Only a match will lead to the acceptance of P'.

LiSP ensures that keystreams will never be reused, with the following operations. First, a sender blends its own *nodeID* into the generation of the keystream to ensure that the various parties sharing TK use different keystreams. Second, a sender increments its own IV by 1 for each message it transmits to avoid any repetition of the keystream. Finally, the KS updates TK at an appropriately chosen interval, T_{refresh}, guaranteeing that none of its members (including itself) starts to reuse IV. The length of the IV field can be made small, thanks to TK rekeying.

The way LiSP manages TK and IVs differs significantly from that of SPINS. In SPINS, IVs are not included in messages/packets, but maintained internally by two communicating peers. SPINS also updates TK whenever IV wraps around. As a result, SPINS has shorter packets than LiSP by the length of the IV field (plus *keyID*). However, SPINS performs poorer than LiSP in the network of a large number of sensors for the following reasons. First, SPINS incurs overheads of (triggered) TK rekeying and IV resynchronization, which increases rapidly as the network size grows. Second, SPINS requires each sensor to allocate memory for maintaining IV states of all other sensors, which also increases with the network size. By contrast, LiSP can control the overhead of TK re-keying regardless of the network size, and reduces the generation of control packets, improving the performance in contention-based networks.

4.4 Intergroup Communication

Under LiSP, the entire network is divided into multiple groups, each with a KS. This architecture is scalable in that compromises in one group do not affect the other groups. It also retains high-performance rekeying, since each reconfiguration is confined to a single group, while groups without any compromised node keep broadcasting TKs. This means that TKs for intragroup communications are independently managed by KSs.

For intergroup communications, KSs should coordinate with one another under the control of KSN as follows. First, all KSs agree in advance on n, t, and T_{refresh}, by receiving them from KSN. Also, the time to initiate TK management is loosely synchronized with a clock skew of less than $T_{\text{refresh}}/2$. Second, KSs and KSN use a key-agreement algorithm such as those in Steiner et al. [1998] and Setia et al. [2000] to agree on the initial seed TK_n for the key-sequence, thus ensuring all KSs to have the same key-sequence. Third, for intergroup traffic, the KS prefixes to the encrypted payload the position of the encryption key in the key-sequence.

4.5 Realization of LiSP

We now describe how to realize the proposed TK management protocol. Its implementation is comprised of the server-side and client-side programs. Both

```
constants:  n, t, T_refresh;

initial setup:
    compute {TK_i | i = 1, ..., n};
    k = t + 2;
    for all m,
        unicast to node m InitKey(m) containing E_{MK_m}(t | TK_k | T_refresh) | MAC(t|TK_k|T_refresh);

for every T_refresh:
    k ++;
    broadcast UpdateKey containing E_{TK_{k-t-1}}(TK_k);

if  member(s) compromised:
    k += t + 2;
    for all m,
        unicast to node m InitKey(m) containing E_{MK_m}(t | TK_k | T_refresh) | MAC(t|TK_k|T_refresh);

if  k == n:
    do initial setup;

if  a new node m joins the group || RequestKey(m) received:
    if m is a new node,
        do entity authentication with m;
    unicast to m InitKey(m) containing current configuration;
```

Fig. 8. The pseudocode for the KS.

programs require external modules, such as the IDS, the entity authentication protocol, and the cryptographic one-way function.

The pseudocode for KS is given in Figure 8. After initialization, the KS periodically broadcasts a new TK to all members. It also reconfigures the group security in case of addition/compromise of a node, exhaustion of all TKs, or an explicit request from a member node.

Figure 9 gives the pseudocode for client nodes.[11] The client nodes are *stateless* so that the client-side TK rekeying can be done at a very low cost. Each client maintains two internal variables, e and $\text{TK}^{\dagger}$. e keeps track of the number of TK-disclosures that the node failed to receive correctly, and $\text{TK}^{\dagger}$ points to the most recent TK in the key-buffer. The right-shift operation is done as illustrated in Figures 4 and 5.

5. PERFORMANCE EVALUATION

We are interested in evaluating the resource consumption of the proposed TK management and demonstrating its applicability to the resource-constrained sensor devices. As mentioned earlier, the IDSs resource consumption is not a concern, as it runs on a platform with enough resources. To evaluate the performance of TK management, we first quantify (1) the overheads (in both computation and communication) a node pays to renew TKs, and (2) the performance gain the node makes by adding reliability within LiSP. Then, based on the evaluation results, we analyze how LiSP defends itself against various attacks. In this section, we present computation and communication overheads, efficiency of the built-in reliability mechanism, and analyze the security achieved with LiSP.

[11]The KS should also perform these tasks for proper communication with other clients.

```
if InitKey received:
    Decrypt InitKey to get TK_{t+2};
    Compute TK_{t+1}, ..., TK_1 from TK_{t+2};
    Copy TK_{t+2}, ..., TK_1 to key-buffer/key-slots;
    Activate TK_1 for encryption;
    Set e = 0 and TK^† = TK_{t+2};
    Set ReKeyingTimer to T_{refresh}/2;

if UpdateKey received && if not seen before:
    Decrypt UpdateKey to get TK_k;
    Right-shift the key-buffer/key-slot;
    if H^{e+1}(TK_k) != TK^†,
        e++;
    else,
        if e ≥ 1,
            Copy H(TK_k), ..., H^e(TK_k) to key-buffer;
            e = 0;
        Copy TK_k to key-buffer, TK^† = TK_k;

if ReKeyingTimer triggered:
    if key-buffer not right-shifted,
        Right-shift the key-buffer/key-slot;
        e++;
    Swap active & inactive TKs in key-slots;
    Set ReKeyingTimer to T_{refresh};

if e == t:
    Unicast RequestKey to KS;
```

Fig. 9. The pseudocode for the client.

Table I. Computational Overhead

	Key-Server	Client
Initial setup	$n \cdot C_H$	$(t+1) \cdot C_H$
Rekeying	—	$(e+1) \cdot C_H$

5.1 Computational Overhead

We evaluate the computational overhead of LiSP per group, demonstrating its robustness to losses of, and attacks on, TKs. Since hash computation is the most resource-consuming operation (as compared to data copying/moving), we only consider the cost of computing the cryptographic hash function, H. Let C_H denote the cost of computing a single hash function. Then, computational costs for the KS and the client are given in Table I.

We want to evaluate the average number of hash computations per TK-disclosure. First, the KS, on average, computes $N_{\text{KS}} = \frac{n}{n-t-1} < 1$. Since $N_{\text{KS}} \approx 1$, if $n \gg t$, the KS performs approximately one hash computation per TK-disclosure. Second, the expected number of hash computations per TK-disclosure, $E[N_{\text{client}}]$, of a client is derived in the Appendix, under the assumption that each occurrence of TK loss or failure is random and mutually independent. Given fixed p_L (=Pr{TK is lost}) and p_F (=Pr{TK authentication fails | TK is received}), from Equation (A.12), we have

$$E[N_{\text{client}}] = \sum_{k=0}^{t}(k+1)(1-p_L)(p_L+p_F)^k p(0,0)+(t+1)(p_L+p_F)^{t+1}p(0,0) \qquad (1)$$

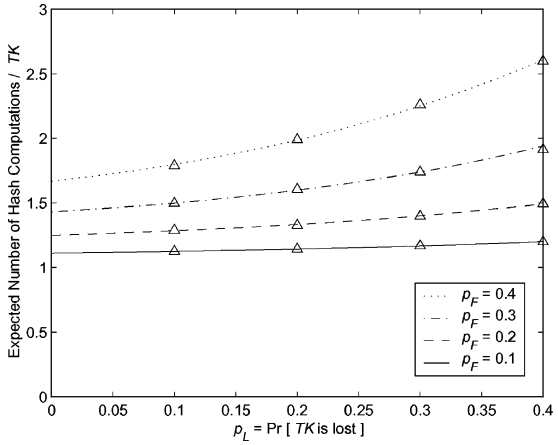

Fig. 10. The expected number of hash computations per TK-disclosure at the client node vs. p_L: $p_F = 0.1 \sim 0.4$, $t = 10$.

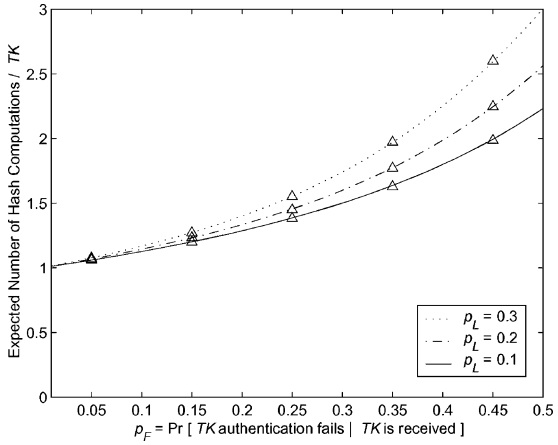

Fig. 11. The expected number of hash computations per TK-disclosure at the client node vs. p_F: $p_L = 0.1 \sim 0.3$, $t = 10$.

where

$$p(0,0) = \frac{1-(p_L+p_F)}{1-(p_L+p_F)^{t+1}}. \tag{2}$$

p_L reflects the channel condition: a higher p_L represents a highly lossy wireless channel. By contrast, p_F is mostly affected by the adversary's attempts to manipulate TKs, leading to a DoS attack. So, $E[N_{\text{client}}]$ must be small even in case of a high p_F.

Figure 10 plots $E[N_{\text{client}}]$ as a function of p_L, while varying p_F from 0.1 to 0.4. Similarly, Figure 11 plots $E[N_{\text{client}}]$ as a function of p_F, while varying p_L from 0.1 to 0.3. The key-buffer length, t, is set to 10. The points marked with '△' in the figures are the simulated numbers of the average hash computations for

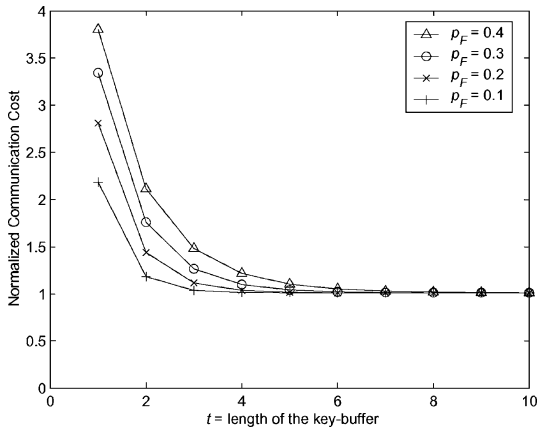

Fig. 12. The normalized communication cost at the client node versus t: $p_F = 0.1 - 0.4$, $\alpha = 10$, $p_L = 0.05$, $n = 1,000$.

100,000 TK-disclosures. The simulation results closely match those obtained from the above equation, verifying the accuracy of the equation.

Figures 10 and 11 show that p_F has greater influence on the expected hash computations than p_L. For instance, if $p_F = 0$, each node incurs one hash computation, while if $p_F = 0.5$, it computes 2.5–3 hash functions, because all incoming TKs must be authenticated via hash computations. Each client computes less than three hash functions per TK-disclosure even in the worst case, that is, when a half of TK broadcasts get corrupted by the attacker ($p_F = 0.5$).

5.2 Communication Overhead

We evaluate the overhead of communication between the KS and a client, and show that the key-buffer length determines the communication overhead. The expected communication cost, C_{comm}, normalized by the communication cost of `UpdateKey` transmission is (see the Appendix for its derivation):

$$C_{\text{comm}} = \alpha \left[\frac{1}{n} + (p_L + p_F)^t p(0,0) \right] + \sum_{k=0}^{t-1} (p_L + p_F)^k p(0,0) \tag{3}$$

where α is the ratio of the communication cost of `InitKey` to that of `UpdateKey`. The value of C_{comm} close to 1 means that most TK-disclosures are made through `UpdateKey`, and hence, LiSP is efficient in terms of communication overhead. By contrast, LiSP gets less efficient as C_{comm} approaches α.

Figure 12 plots C_{comm} as a function of t when $\alpha = 10$, $p_L = 0.05$, and $n = 1,000$. This choice of α implies that the cost of `InitKey` transmission is 10 times higher than that of `UpdateKey` broadcasting, due to the larger packet size of `InitKey` and requirements for reliability and authentication services. The results show that, regardless of p_F, a smaller t incurs a higher communication overhead. Thus, it is better for the KS to configure LiSP with a large t, as far as clients can allocate the required key-buffer space. A desirable value of t is 5 or 6, with which the normalized communication cost approaches 1, even under serious attacks.

Table II. Transmission Costs

	No. of Packets	Total Cost in Bytes
LiSP	$0.7N$	$12.6N$
Unicast	$4N$	$58N$

5.3 Efficiency of TK Management

We would like to evaluate the performance of LiSP, and show its effectiveness in reducing energy consumption. Since a significant portion of energy is spent on packet transmission, we measure transmission costs (both the number of control packets and the total cost in number of bytes) per TK-disclosure. We consider a group consisting of a KS and N group members. It is reasonable to assume that the spanning tree for the group has been established, and hence, each node in the group knows its child nodes.

We compare the proposed TK distribution with the scheme based on unicasts plus explicit message authentication. Other link-layer broadcasting schemes [Tourrilhes 1998; Tang and Gerla 2000] are excluded because they do not ensure 100% reliability. In LiSP, the KS broadcasts `UpdateKey` and, upon receiving the packet, intermediate nodes rebroadcast it if they have at least one child node. Therefore, each TK distribution incurs strictly less than N broadcasts. On the other hand, in the unicast scheme, each node in the group must receive a packet from its parent node via handshake (RTS/CTS/Data/ACK), requiring exactly N unicasts. Also, the unicast packet must include a message authentication code in it.

For the purpose of evaluation, we chose parameter values based on settings in Ye et al. [2002], that is, the sizes of the link-layer header, CRC, RTS, CTS, and ACK are 6, 2, 8, 8, and 8 bytes, respectively. Each TK is set to be 10 bytes long and the message authentication code (MD4) is 16 bytes long. Then, the size of broadcast (`UpdateKey`) and unicast packets are 18 and 34 bytes, respectively. To determine the number of broadcast packets generated in LiSP, we conducted simulation, in which locations of N nodes were randomly generated, a spanning tree among them was built, and the total number of broadcast packets were counted. The result is that a group of N nodes (excluding the KS), on average, generated $0.7N$ broadcast packets per TK-disclosure. From these results, we can calculate the transmission costs of LiSP and the unicast scheme, as given in Table II.[12] Clearly, LiSP outperforms the unicast scheme in that the transmission cost of LiSP is only 18% (in number of packets) and 22% (in number of bytes) of the unicast case.

5.4 Security Analyses

We now discuss how LiSP defends against various attacks. As described in Section 3, an adversary either passively eavesdrops ongoing packet transmissions or actively inject packets into the network to disrupt network functions. In particular, the adversary will likely mount active attacks against TK management, that is, modifying/injecting false TKs, jamming the channel to disrupt

[12]In the unicast scheme, we ignored possible collisions on RTS packets.

TK reception, inducing collisions on packets conveying TK, forcing nodes to repeat TK retransmission, and so on. These attacks may, in turn, result in DoS, replay and man-in-the-middle attacks.

LiSP is effective in defeating attacks on TK management as follows. First, any modification to the TK will be rejected by the authentication test at the receiver. Similarly, any dropped TK due to collision will be recovered before its activation. Second, the expected computational overhead is bounded: (i) as shown in Section 5.1, each node incurs, on average, less than three hash computations, even when about 65% of TKs are manipulated/compromised by the attacker; (ii) the KS can easily detect the presence of attacks and disable the associated nodes, if more than a half of TKs fail authentication tests. Therefore, LiSP is robust to DoS attacks (attempting to interfere with TK distribution via high-power jamming and forced collisions or retransmissions) in that each client is expected to compute up to three hash computations per TK-disclosure and lost/corrupted TKs can be recovered without retransmissions. Third, replay attacks will not succeed; since the TK manager expects a unique TK in the given refreshment interval that cannot be inferred from the past TKs, replayed TKs will not pass authentication tests. Fourth, LiSP defeats main-in-the-middle attacks, in which the attacker fools the group as if s/he were the KS, because each client is capable of rejecting a false KS in the mutual authentication stage with the KS. Finally, from the fact that TK is transmitted every T_{refresh}, the adversary can predict when to launch an attack. We can prevent this by introducing randomization: the KS adds Δ, chosen from the interval $[-T_{\text{refresh}}/8,\ T_{\text{refresh}}/8]$, to the scheduled broadcast time. However, this scheme will reduce the timing margin against clock skews to $T_{\text{refresh}}/4$.

The attacker may subvert a node and acquire all key information. In such a case, s/he can eavesdrop communications and immediately inject bogus messages within the group. However, LiSP preserves the security of the whole network as follows. First, this attack will be valid only until an IDS, running on KS, detects/disables the node. Note that it is crucial to have a good IDS, which can uncover any compromise on keys or the node itself with minimal latency. Second, the scope of this attack will be limited to a single group.

Finally, LiSP prevents attacks on data packets. It does not allow the attacker to mount keystream reuse-based attacks by periodically renewing TK, and mixing *nodeID* and per-packet IV in the generation of keystreams, as explained in Section 4.3. Moreover, s/he can neither decipher nor inject/modify data packets, without the knowledge of TK.

6. CONCLUSION

In this paper, we proposed a LiSP that makes security/energy-efficiency tradeoffs via efficient refreshment of keys. In LiSP, a KS independently maintains the security of a group using two main components: intrusion detection and TK management. By employing the cryptographic one-way function and TK double-buffering, the TK management offers (i) efficient TK broadcast without relying on retransmissions/ACKs; (ii) authentication and TK recovery without

incurring additional overhead; (iii) seamless TK rekeying without disrupting ongoing data traffic. Moreover, it tolerates very loose time synchronization and does not require reliability support at the link layer.

We evaluated the performance of LiSP using overhead measurements and security analyses. The measurement results have shown that (1) each node computes, on average, less than three hash functions per TK-disclosure, even in the presence of severe attacks on TKs, and (2) with the storage of $\geq$8 TKs ($t \geq 6$), LiSP distributes most TKs via broadcasting, and hence, it is very efficient compared to other reliability mechanisms. Our security analyses have demonstrated LiSP's effectiveness in defeating various security attacks. LiSP's strength lies in meeting conflicting goals of providing high-level security and maximizing energy efficiency.

APPENDIX

We derive, through the Markov chain analysis, the computation overhead of client nodes and the communication overhead between the KS and the client. We assume that each occurrence of TK loss or failure is random and mutually independent. We also assume that if the key-buffer of a node becomes empty, it finishes the operation, the request/reception of a current TK (via `RequestKey`), within T_{refresh}. These are reasonable assumptions in that T_{refresh} is typically large enough to uncorrelate them. In this Appendix, we first derive a closed-form expression for steady-state distributions for key-buffer states, and then derive computation and communication overheads.

A.1 Steady-State Distributions

We model the state of each node with a 2-dimensional Markov chain, as shown in Figure 13. Each state (i, j) represents that there were i TK losses and j TK failures, and hence, there are $(i + j)$ empty slots in the key-buffer. The state transition is triggered by three events: a TK loss, a TK authentication failure, and a successful TK reception. As the node misses a TK, the state will be transitioned horizontally, whereas the authentication failure of the received TK will cause a vertical state transition. Let $p_L = \Pr\{\text{TK is lost}\}$, $p_F = \Pr\{\text{TK authentication fails} \mid \text{TK is received}\}$ and $p_S = 1 - p_L - p_F$. Also, let $p(i, j)$ denote the steady-state probability of state (i, j), and $p_e(k)$ the probability that there were exactly k empty slots. Then, we have

$$p_e(k) = \sum_{i+j=k} p(i, j), \quad k = 0, \ldots, t \tag{A.1}$$

and

$$\sum_{k=0}^{t} p_e(k) = 1. \tag{A.2}$$

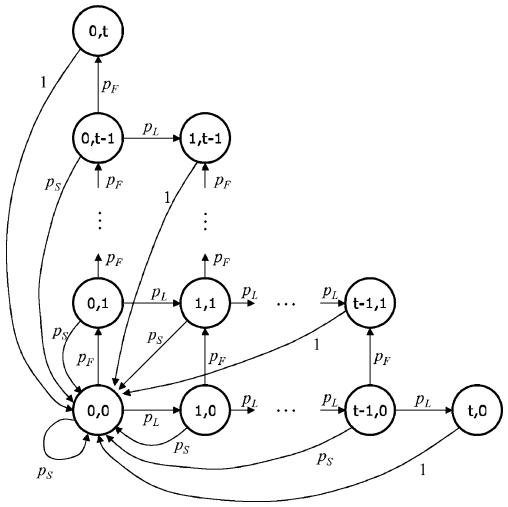

Fig. 13. The state transition diagram for LiSP.

We derive steady-state distributions, $p(i, j)$ and $p_e(k)$ $\forall i, j, k$, as follows. From the global balance equations, we get

$$p(i,j) = \begin{cases} p_F \cdot p(i, j-1), & i = 0, j > 0 \\ p_L \cdot p(i-1, j), & i > 0, j = 0 \\ p_F \cdot p(i, j-1) + p_L \cdot p(i-1, j), & i > 0, j > 0 \\ p^*/(p_F + p_L), & i = j = 0 \end{cases} \tag{A.3}$$

where

$$p^* = \sum_{i'+j'<t,(i',j')\neq(0,0)} p_S \cdot p(i', j') + \sum_{i'+j'=t} p(i', j'). \tag{A.4}$$

Representing $p(i, j)$ with respect to $p(0, 0)$ yields

$$p(i,j) = \binom{i+j}{i} \cdot p_L^i p_F^j \cdot p(0,0), \quad (i,j) \neq (0,0). \tag{A.5}$$

From Equations (A.1) and (A.5), we have

$$p_e(k) = (p_L + p_F)^k \cdot p(0,0), \quad k = 0, \ldots, t. \tag{A.6}$$

Then, from Equations (A.2) and (A.6), $p(0, 0)$ is given by

$$p(0,0) = \frac{1 - (p_L + p_F)}{1 - (p_L + p_F)^{t+1}}. \tag{A.7}$$

A.2 Computational Overhead

We derive the expected number of hash computations per TK-disclosure. Let N_{client} denote the number of hash computations per TK-disclosure. If there is

exactly k ($< t$) empty slots, N_{client} is given by

$$N_{\text{client}} = \begin{cases} 0, & \text{TK is lost} \\ k+1, & \text{TK authentication fails} \\ k+1, & \text{TK authentication succeeds.} \end{cases} \tag{A.8}$$

When all t slots are empty, if the node encounters a TK loss or failure, it must explicitly request the next TK via `RequestKey`, and then do $(t+1)$ additional hash computations using the received TK. Therefore,

$$N_{\text{client}} = \begin{cases} t+1, & \text{TK is lost} \\ 2t+2, & \text{TK authentication fails} \\ t+1, & \text{TK authentication succeeds.} \end{cases} \tag{A.9}$$

The expected N_{client} when there are k empty slots, $E[N_{\text{client}} \mid k\text{empty}]$, is thus derived as

$$E[N_{\text{client}} \mid k \text{ empty}] = \begin{cases} (k+1)\cdot(1-p_L), & k < t \\ (t+1)\cdot(1-p_L) + (t+1)\cdot(p_L+p_F), & k = t \end{cases} \tag{A.10}$$

Finally, $E[N_{\text{client}}]$ is given by

$$E[N_{\text{client}}] = \sum_{k=0}^{t} E[N_{\text{client}} \mid k \text{ empty}] \cdot p_e(k) \tag{A.11}$$

$$= \sum_{k=0}^{t} (k+1)(1-p_L)(p_L+p_F)^k p(0,0) + (t+1)(p_L+p_F)^{t+1} p(0,0). \tag{A.12}$$

A.3 Communication Overhead

We finally derive the communication cost of a client per TK-disclosure. Let C_{init} and C_{update} denote communication costs for transmitting `InitKey` and `UpdateKey` packets, respectively. Also, let $\alpha = C_{\text{init}}/C_{\text{update}}$. Then, $\alpha > 1$, because the `InitKey` packet consumes more bandwidth/resources. The KS will transmit the `InitKey` packet (1) once every n TK-disclosures; and (2) if all t slots in the key-buffer become empty, else the `UpdateKey` packet is broadcast. Therefore, the expected communication cost of a client is

$$C_{\text{init}} \cdot \left[\frac{1}{n} + p_e(t)\right] + C_{\text{update}} \cdot \sum_{k=0}^{t-1} p_e(k).$$

The communication cost normalized by C_{update} is given by

$$C_{\text{comm}} = \alpha \cdot \left[\frac{1}{n} + p_e(t)\right] + \sum_{k=0}^{t-1} p_e(k) \tag{A.13}$$

$$= \alpha \cdot \left[\frac{1}{n} + (p_L+p_F)^t p(0,0)\right] + \sum_{k=0}^{t-1} (p_L+p_F)^k p(0,0). \tag{A.14}$$

REFERENCES

ANDERSON, R. AND KUHN, M. 1996. Tamper resistance—a cautionary note. In *Proceedings of 2nd Usenix Workshop Electronic Commerce*. USENIX, Oakland, CA, 1–11.

BASAGNI, S., HERRIN, K., BRUSCHI, D., AND ROSTI, E. 2001. Secure pebblenets. In *Proceedings of ACM MobiHoc'01*. ACM, Long Beach, CA.

BASS, T. 2000. Intrusion detection systems and multisensor data fusion. *Commun. ACM*.

BONNET, P., GEHRKE, J., AND SESHADRI, P. 2001. Towards sensor database systems. In *Mobile Data Management (MDM'01)*, Hong Kong, China.

BORISOV, N., GOLDBERG, I., AND WAGNER, D. 2001. Intercepting mobile communications: the insecurity of 802.11. In *Proceedings of IEEE/ACM MobiCom'01*. ACM, Rome, Italy, 180–189.

BURNSIDE, M., CLARKE, D., MILLS, T., DEVADAS, S., AND RIVEST, R. 2002. Proxy-based security protocols in networked mobile devices. In *Proceedings of SAC'02*.

CARMAN, D. W., KRUUS, P. S., AND MATT, B. J. 2000. Constraints and Approaches for Distributed Sensor Network Security. NAI Labs Technical Report #00-010, NAI Labs.

CHANG, I., ENGEL, R., KANDLUR, D., PENDARAKIS, D., AND SAHA, D. 1999. Key management for secure internet multicast using Boolean function minimization techniques. In *Proceedings of IEEE INFOCOM'99*. IEEE.

CROSSBOW. 2003. *MICA, MICA2 Motes & Sensors*. Available at http://www.xbow.com/.

DUCKWORTH, G. L., GILBERT, D. C., AND BARGER, J. E. 1996. Acoustic counter-sniper system. In *International Symposium on Enabling Technologies for Law Enforcement and Security*. SPIE, Boston, MA.

ESCHENAUER, L. AND GILGOR, V. D. 2002. A key-management scheme for distributed sensor networks. In *Proceedings of ACM CCS'02*. ACM, Washington, DC. 41–47.

ESTRIN, D., GOVINDAN, R., HEIDEMANN, J., AND KUMAR, S. 1999. Next century challenges: scalable coordination in sensor networks. In *Proceedings of IEEE/ACM MobiCom'99*. ACM.

HALLER, N. 1995. The s/key one-time password system. Request for Comments (Informational) 1760. IETF.

HARNEY, H. AND MUCHENHIRN, C. 1997. Group key management protocol (GKMP) architecture. RFC 2094. IETF.

HESPANHA, J. P., KIM, H. J., AND SASTRY, S. 1999. Multiple-agent probabilistic pursuit-evasion games. In *Proceedings of the 38th Conference on Decision and Control*. IEEE, Phoenix, AZ. 2432–2437.

HILL, J., SZEWCZYK, R., WOO, A., HOLLAR, S., CULLER, D., AND PISTER, K. 2000. System architecture directions for networked sensors. In *Proceedings of ASPLOS*.

IEEE. 1997. Part 11: wireless LAN medium access control (MAC) and physical layer (PHY) specifications. In *IEEE Std 802.11-1997*. IEEE.

ILGUN, K., KEMMERER, R. A., AND PORRAS, P. A. 1995. State transition analysis: a rule-based intrusion detection approach. *IEEE Trans. Softw. Engng. 21*, 3, 181–199.

KONG, J., ZERFOS, P., LUO, H., LU, S., AND ZHANG, L. 2001. Providing robust and ubiquitous security support for mobile ad-hoc networks. In *Proceedings of ICNP'01*. IEEE, Riverside, CA.

KUMAR, S. AND SPAFFORD, E. H. 1995. A software architecture to support misuse intrusion detection. In *Proceedings of the 18th National Information Security Conference*. 194–204.

LI, J., JANNOTTI, J., COUTO, D. S. J. D., KARGER, D. R., AND MORRIS, R. 2000. A scalable location service for geographic ad hoc routing. In *Proceedings of IEEE/ACM MobiCom'00*. ACM.

LI, X. S., YANG, Y. R., GOUDA, M. G., AND LAM, S. S. 2001. Batch rekeying for secure group communications. In *Proceedings of 10th International World Wide Web Conference*.

MADDEN, S., SZEWCZYK, R., FRANKLIN, M. J., AND CULLER, D. 2002. Supporting aggregate queries over ad-hoc wireless sensor networks. In *Proceedings of IEEE WMCSA'02*. IEEE, New York.

MAINWARING, A., POLASTRE, J., SZEWCZYK, R., CULLER, D., AND ANDERSON, J. 2002. Wireless sensor networks for habitat monitoring. In *Proceedings of ACM WSNA'02*. ACM.

MCGREW, D. A. AND FLUHRER, S. R. 2000. The stream cipher encapsulating security payload. In *draft-mcgrew-ipsec-scesp-01.txt*. IETF.

MITTRA, S. 1997. Iolus: a framework for scalable secure multicasting. In *Proceedings of ACM SIGCOMM'97*. ACM.

PAGANI, E. AND ROSSI, G. P. 1997. Reliable broadcast in mobile multihop packet networks. In *Proceedings of IEEE/ACM MobiCom'97*. ACM.

PERRIG, A., CANETTI, R., SONG, D., AND TYGAR, J. D. 2001. Efficient and secure source authentication for multicast. In *Proceedings of NDSS'01*. ISOC, San Diego, CA.

PERRIG, A., SZEWCZYK, R., WEN, V., CULLER, D., AND TYGAR, J. D. 2001. SPINS: security protocol for sensor networks. In *Proceedings of IEEE/ACM MobiCom'01*. ACM, Rome, Italy, 189–199.

SETIA, S., KOUSSIH, S., JAJODIA, S., AND HARDER, E. 2000. Kronos: a scalable group re-keying approach for secure multicast. In *Proceedings of IEEE Symposium on Security and Privacy'00*. IEEE.

SETIA, S., ZHU, S., AND JAJODIA, S. 2002. A comparative performance analysis of reliable group rekey transport protocols for secure multicast. In *Proceedings of Performance'02*.

SHAMIR, A. 1979. How to share a secret. *Commun. ACM 22*, 11.

SINGH, S. AND RAGHAVENDRA, C. S. 1998. Pamas: power aware multi-access protocol with signalling for ad hoc networks. *ACM Comput. Commun. Rev. 28*, 3 (July), 5–26.

STEINER, M., TSUDIK, G., AND WAIDNER, M. 1998. Cliques: a new approach to group key agreement. In *Proceedings of ICDCS'98*.

SUN, M., HUANG, L., ARORA, A., AND LAI, T. H. 2002. Reliable mac layer multicast in IEEE 802.11 wireless networks. In *Proceedings of IEEE ICPP'02*. IEEE.

TANG, K. AND GERLA, M. 2000. Mac layer broadcast support in 802.11 wireless networks. In *Proceedings of MILCOM'00*, Los Angeles, CA.

TOURRILHES, J. 1998. Robust broadcast: improving the reliability of broadcast transmissions on CSMA/CA. In *Personal, Indoor and Mobile Radio Communications*. IEEE, Boston, MA.

VIDAL, R., SHAKERNIA, O., KIM, H. J., SHIM, H., AND SASTRY, S. 2002. Probabilistic pursuit-evasion games: theory, implementation and experimental evaluation. *IEEE Trans. Robotics Automat. 18*, 5 (Oct.), 662–669.

WALKER, J. R. 2000. Unsafe at any key size; an analysis of the wep encapsulation. IETF.

WALLNER, D. M., HARDER, E. G., AND AGEE, R. C. 1999. Key management for multicast: issues and architecture. RFC 2627. IETF.

WONG, C. K., GOUDA, M. G., AND LAM, S. S. 1998. Secure group communications using key graphs. In *Proceedings of ACM SIGCOMM'98*. ACM.

WOO, A. AND CULLER, D. 2001. A transmission control scheme for media access in sensor networks. In *Proceedings of IEEE/ACM MobiCom'01*. ACM, Rome, Italy, 221–235.

WOOD, A. D. AND STANKOVIC, J. A. 2002. Denial of service in sensor networks. *IEEE Comput. 35*, 10 (Oct.).

YANG, Y. R., LI, X. S., ZHANG, X. B., AND LAM, S. S. 2001. Reliable group rekeying: design and performance analysis. In *Proceedings of ACM SIGCOMM'01*. ACM.

YE, F., LUO, H., CHENG, J., LU, S., AND ZHANG, L. 2002. A two-tier data dissemination model for large-scale wireless sensor networks. In *Proceedings of IEEE/ACM MobiCom'02*. ACM, Atlanta, GA.

YE, W., HEIDEMANN, J., AND ESTRIN, D. 2002. An energy-efficient mac protocol for wireless sensor networks. In *Proceedings of IEEE INFOCOM'02*. IEEE.

ZHANG, R., QIAN, D., BA, C., WU, W., AND GUO, X. 2001. Multi-agent based intrusion detection architecture. In *Proceedings of International Conference on Computer Networks and Mobile Computing*. IEEE, Beijing, China, 494–504.

ZHANG, Y. AND LEE, W. 2000. Intrusion detection in wireless ad hoc networks. In *Proceedings of IEEE/ACM MobiCom'00*. ACM, Boston, MA, 275–283.

ZHOU, L. AND HAAS, Z. J. 1998. Securing ad hoc networks. *IEEE Netw. Mag. 13*, 6 (Nov.).

ZHOU, L., SCHNEIDER, F. B., AND VAN RENESSE, R. 2002. Coca: a secure distributed on-line certification authority. *ACM Trans. Comput. Syst. 20*, 4 (Nov.).

Received February 2003; accepted July 2003