Secure Routing Based on Distributed Key Sharing in Large-Scale Sensor Networks

TAEJOON PARK
Samsung Electronics
and
KANG G. SHIN
University of Michigan, Ann Arbor

Sensor networks, usually built with a large number of small, low-cost sensor nodes, are characterized by their large-scale and unattended deployment, necessitating "secure" communications between nearby, as well as remote, sensor nodes for their intended applications and services. Key setup/sharing is crucial to the protection of such applications/services from attacks, but existing (public-key, cluster-based, or pairwise) solutions become too expensive (hence, inefficient) when the underlying applications/services require communications between distant sensor nodes. To remedy this inefficiency, we propose a novel *distributed key-sharing* scheme, in which each participating sensor node shares unique keys with a small number of other sensor nodes—called *distributed key servers* (DKSs)—chosen according to their geographic distance and communication direction. Using DKSs, we develop two secure routing protocols: (1) *secure geographic forwarding* that delivers packets by using a chain of DKS lookups, each secured with its own key and forwarded geographically; and (2) *key establishment* that creates a secure session between two distant sensor nodes based solely on symmetric-ciphers. These protocols enable low-cost, low-power sensors to provide high-level security at a very low cost.

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*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Distributed Networks*

General Terms: Design, Security

Additional Key Words and Phrases: Distributed key sharing and servers, secure geographic forwarding, key establishment, attack tolerance, large-scale sensor networks

The work reported in this paper is supported in part by the ONR and the NRL under Grant No. N00014-04-10726, by the NSF under Grant CCR-0329629, and by Cisco Corporation.
Authors' address: Taejoon Park and Kang G. Shin, Real-Time Computing Laboratory (RTCL), Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan 48109-2122; email: joy.park@samsung.com, kgshin@eecs.umich.edu.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2008 ACM 1539-9087/2008/02-ART20 $5.00 DOI 10.1145/1331331.1331344 http://doi.acm.org/10.1145/1331331.1331344

ACM Reference Format:

Park, T. and Shin, K. G. 2008. Secure routing based on distributed key sharing in large-scale sensor networks. ACM Trans. Embedd. Comput. Syst. 7, 2, Article 20 (February 2008), 28 pages. DOI = 10.1145/1331331.1331344 http://doi.acm.org/10.1145/1331331.1331344

1. INTRODUCTION

An increasing number of safety- and security-critical applications, such as situation monitoring and facility surveillance, rely on a network of small, inexpensive, battery-powered sensor devices that have limited energy supplies, storage, computation, and communication capacities. These sensor networks can be used for various critical applications, such as safeguarding of, and early warning systems for, the physical infrastructure that includes buildings, transportation systems, water supply systems, waste treatment systems, power generation and transmission, and communication systems. To meet these applications' needs, we must resolve several operational issues and challenges, such as energy efficiency (in the sense of maximizing the lifetime of sensor networks), scalability to a large number (thousands to millions) of nodes, and survivability even in a harsh, unattended environment.

With rapid advances in device technology, the processing capability of embedded systems has been improving at an exponential rate. However, this improvement in computing performance accompanies a rapid increase in complexity and power consumption. By contrast, the battery and energy storage technologies have been improving at a much slower pace, failing to meet the increasing energy demands of emerging embedded systems. Energy efficiency is, therefore, critical to all portable, embedded computing devices. Specifically, in sensor networks where it is often very difficult, and sometimes impossible, to change or recharge batteries for devices after their deployment, energy efficiency is one of their most important requirements.

The critical role of sensor networks in their intended applications requires high-level security throughout their lifetime. TinySec [Karlof et al. 2004] realizes a link-layer security mechanism for message encryption and authentication using symmetric-key ciphers. In that scheme, keys can be established and renewed with conventional public-key algorithms [Asokan and Ginzboorg 2000; Carman et al. 2002], such as the well-known Diffie–Hellman (DH) protocol. However, these algorithms are not suitable for sensor networks, as they usually require complicated processing, extensive usage of memory, and large key lengths, causing faster depletion of battery power if they were used in sensor devices. Attempts [Watro et al. 2004; Malan 2004] have been made to realize public-key algorithms on a well-known Mote platform [Crossbow 2003], to show the feasibility of public-key algorithms on sensor devices. Their implementations are, however, still too "heavy" to be employed in sensor devices: each public-key operation consumed 1.19–12.64 [J], allowing just 51,731–4,870 operations even when a sensor's total energy budget of 61,560 [J] was devoted solely to this task. Obviously, this is not acceptable, as sensor nodes must also perform other tasks and are required to be operational for at least several months without replacing/recharging their batteries.

We, therefore, need a *lightweight* security protocol whose primary design objective is energy efficiency. To this end, we have taken an approach to building a secure network layer via "cooperation" among sensor nodes, without relying on a trusted central server. This approach is motivated by the fact that a sensor network inherently relies on collective assurance among multiple low-cost sensor nodes to execute high-precision/assurance missions and, hence, it is important to use mutual cooperation in developing energy-efficient security protocols that achieve high-level security without using resource-demanding public-key ciphers, thus significantly extending the network lifetime.

It is essential to share keys among sensors for them to operate properly in their link-layer security mechanism. Clearly, the degree of key sharing is inversely proportional to the level of security the sensor network can achieve. The highest level of security would be achieved if every pair of sensors share their own key independently of others, in that individual subversions do not compromise the rest of the network. However, this scheme is not realistic due mainly to the large ($\mathcal{O}(N^2)$ per sensor) storage requirement where N is the total number of sensors. Sharing an identical key among all sensors offers the least security because a single compromised sensor completely reveals the secret of the entire network. Localized key sharing—either cluster-based [Carman et al. 2000; Basagni et al. 2001; Park and Shin 2004] or pairwise [Carman et al. 2000; Eschenauer and Gligor 2002; Chan et al. 2003]—schemes may mitigate the risk of sensor subversions, but cannot effectively meet both security and performance requirements, especially in a large-scale network where a sensor node may need to communicate with a distant peer. Hence, the degree (and the way) of key sharing must be chosen (designed) to allow the designer to make a tradeoff between security and performance.

In this paper, we propose a novel *distributed* way of sharing keys that optimizes the tradeoff between (1) *security*, in terms of reducing the effects of a compromised sensor node on the rest of the network; and (2) *performance*, in terms of achieving high energy efficiency and low-degree key sharing. The heart of the proposed key sharing is the concept of *distributed key servers* (DKSs): a sensor node serves as DKSs for a small subset of other sensor nodes (chosen according to their geographic distance and routing direction in the entire network) by sharing unique keys. Using DKSs, we present the following secure routing protocols:

—**secure geographic forwarding** that reinforces the coventional geographic forwarding protocol [Karp and Kung 2000; Jain et al. 2001] with a *secure distributed lookup service* that delivers packets based on limited (and distributed) knowledge of shared keys; and

—**key establishment** between any pair of sensor nodes, in which the two nodes *equally* contribute to the value of the key while achieving *secrecy* to others, without relying on the resource-demanding DH protocol.

This approach (1) systematically builds a secure network-layer; (2) realizes a purely symmetric cipher-based key setup protocol that is flexible enough to trade security for residual energy; and (3) gracefully tolerates device

compromises in that the network security is gracefully degraded as the number of undetected compromised sensors increases. We finally show, via in-depth analysis and simulation, that the proposed protocols are indeed energy efficient, scalable, flexible, and robust to subversion of individual sensors.

The rest of the paper is organized as follows. Section 2 reviews the related work and Section 3 describes the proposed protocols. Section 4 analyzes the security of the proposed protocols while Section 5 evaluates their performance via simulation. Finally, the paper concludes with Section 6.

2. RELATED WORK

We review the architecture and communication models for sensor networks, possible security attacks, and, finally, existing security protocols.

2.1 Sensor Network Architecture

Sensor devices are designed to minimize resource requirements, e.g., Motes [Crossbow 2003] feature an 8-bit CPU running at 4 MHz, 128 KB of program memory, 4 KB of RAM and 512 KB of serial flash memory powered by two AA batteries (2850 mAh each). That is, sensors are usually built with devices that have limited processing, communication, and memory capabilities, in order to prolong their lifetime with the limited energy budget.

Sensor networks are deployed for data acquisition for various applications [Hespanha et al. 1999; Vidal et al. 2002; Duckworth et al. 1996; Mainwaring et al. 2002] ranging from physical infrastructure to habitat monitoring. A sensor network is usually built with a large number (thousands or even millions) of sensor nodes, each capable of, for example, reading temperature or detecting (part of) an object moving nearby. Moreover, the sensor network is usually deployed in a hostile/harsh environment, and removal (because of device failures or depletion of battery energy) and addition of sensor nodes are not uncommon. Sensors collaborate and coordinate with one another to perform a higher-level sensing function, e.g., measuring and reporting, with accuracy, the characteristics (such as speed and direction) of a moving object.

Many sensor network applications require location awareness. To meet this requirement, various localization schemes [He et al. 2003; Hu and Evans 2004; Priyantha et al. 2005] have been proposed that allow sensors to determine, with reasonable accuracy, their relative locations within their coverage area. These localization schemes rely on a small number of location–information-equipped (possibly mobile) anchors that provide reference locations and estimate sensors' locations via triangulation or hop-count-to-distance translation of these reference locations.

2.2 Communication Models

The communication models for sensor networks include *cluster-based* and *peer-to-peer* models. The cluster-based model is typically used for a tiered architecture [Cerpa et al. 2001; Wang et al. 2003], where multiple clusters are formed statically and/or dynamically, and a cluster head—that is more capable than the usual member devices—manages and controls operations inside each

cluster. Cluster heads aggregate sensed data within their own cluster (intracluster communication) as well as disseminating/relaying aggregated data among themselves (intercluster communication).

Many emerging applications and services rely more on the peer-to-peer model: each sensor communicates directly with any of the other sensors without relying on dedicated devices. The main challenge associated with a sensor network is the large volume of data to be collected and processed over the entire network. To address this challenge, researchers have proposed efficient ways of storing and extracting relevant data from the network based on the peer-to-peer model. Heidemann, Silva, Intanagonwiwat, Govindan, Estrin, and Ganesan [2001] proposed data to be named and communication abstractions to refer to these names rather than sensor IDs. In a data-centric storage [Ratnasamy et al. 2003], the sensor network stores and looks up relevant data by name, i.e., it hashes the data into geographic coordinates (name) using a geographic hash table (GHT) and stores data at the sensor geographically closest to the hashed coordinates. The two-tier data dissemination [Ye et al. 2002] provides, based on a grid structure, a data-delivery mechanism to mobile data sinks. The locations of mobile data sinks can be looked up through the application of location-management schemes [Li et al. 2000; Xue et al. 2001; Park and Shin 2005a], in which each mobile node chooses a small subset of sensor nodes and periodically updates them with its location information, thus allowing data sources to query these nodes for the data sinks' locations.

Both communication models call for transactions between remote nodes because data sinks can be far away from the data source and, in such a case, the data delivery should be done via intercluster communications or data storage/lookup/dissemination services. The need for long-distance communications will continue to increase as new applications/services are expected to aggressively exploit the large-scale, distributed nature of sensor networks; otherwise, the deployment of, and internet working among, a large number of sensors would not be necessary.

2.3 Security Attacks

Sensor networks are vulnerable to various security attacks, especially because they are deployed in an unattended, hostile environment. For instance, an adversary with a compatible radio receiver/transmitter can easily eavesdrop ongoing communication sessions, inject or modify packets, jam the surrounding area, and even locate specific sensors or hot spots. Possible types of adversaries can be classified, in the order of increasing strength, as: (1) passive attackers, only eavesdropping conversations in the network; (2) active attackers, possessing no cryptographic keys but capable of injecting packets into the network; and (3) active attackers, having all keys of multiple compromised sensors. The last type of attacks is considered as *insider* attacks, while the first two as *outsider* attacks.

Attacks on the sensor network can be classified as: (1) physical attacks on sensor devices, e.g., destroying, analyzing, and/or reprogramming sensors; (2) service disruption attacks on routing, localization, and time synchronization;

(3) data attacks, e.g., traffic capture, replaying, and spoofing; (4) resource-consumption and denial-of-service (DoS) attacks [Wood and Stankovic 2002] that diminishes or eliminates the network's capacity of performing its normal function; and (5) sybil attacks [Douceur 2002] by which a single compromised sensor device claims/presents multiple sensor IDs to control a substantial fraction of the ID space which, in turn, invites other attacks, such as disruption of routing services [Karlof and Wagner 2003]. The adversary may first capture, reverse-engineer, modify, and abuse sensor devices. The compromised sensors can then be redeployed to mount many serious attacks, such as disrupting network services, initiating DoS attacks, and so on. Tamper-proofing techniques, such as the one by [Park and Shin 2005b], can be used as a countermeasure to these physical attacks. The adversary may also disrupt the integrity/availability of localization service by using bogus anchors, announcing false locations or hop-count information, replaying messages (wormhole attacks), and so on [Hu and Evans 2004].

2.4 Security Protocols

Various key establishment protocols [Asokan and Ginzboorg 2000; Carman et al. 2002] have been developed to derive a common key among nodes using public-key algorithms like the DH protocol. However, they are unsuitable for sensor networks, because of their excessive energy demands, let alone the requirement of exchanging public-key certificates. In particular, existing implementations of the DH protocol on sensor nodes [Watro et al. 2004; Malan 2004] consume 1.19–12.64 [J] per operation, which is too much to be usable on devices with a limited energy budget, e.g., 61,560 [J] in Mote powered by two AA batteries. By contrast, symmetric-key ciphers and cryptographic hash functions use significantly less energy, e.g., 0.115 [mJ] in TinySec. It is, therefore, desirable to set up keys based solely on symmetric-key ciphers.

The cluster-based key management [Carman et al. 2000; Basagni et al. 2001; Park and Shin 2004] is concerned with (periodic) distribution and refreshment of a shared cluster key by the cluster head acting as a key server within the cluster. Although this scheme performs well for local transactions, it still has problems, e.g., each cluster head (even though better-equipped and better-protected than member sensor nodes) is a single point of failure, implying that it may break the network's security if compromised. Moreover, an efficient mechanism for securing intercluster communications must be provided to deal with transactions between remote nodes. Note that using a globally shared key for all clusters (rather than inter-cluster key management) makes the entire network vulnerable to a single sensor compromise.

Key predeployment schemes [Carman et al. 2000; Eschenauer and Gligor 2002; Chan et al. 2003; Liu and Ning 2003a; Liu and Ning 2003b; Liu et al. 2005] statically set up pairwise shared keys based on keys loaded into sensor devices prior to their deployment. In the probabilistic key sharing [Eschenauer and Gligor 2002], each sensor is preloaded with multiple (a couple of hundreds) keys randomly chosen from a large pool of keys and, hence, pairwise key sharing is possible between two sensor nodes only if both happen to have

a common key. If this scheme is used to set up a key between a pair of distant sensor nodes, it performs poorly for large-hop path communications, as they require transcoding (decryption followed by reencryption) for each and every hop, thereby significantly risking the security as any malicious sensor node on the path may take control of the communication, as well as increasing sensors' workloads (as routers) and the packet-delivery latency. Hence, it is preferable to minimize the number of transcodings per communication for both security and performance reasons. In fact, this need motivates the proposed DKS approach that intelligently preconfigures pairwise keys with as small a number of remote sensors as possible.

Attempts have also been made to combine several key-sharing schemes. For example, in [Zhu et al. 2003], each node simultaneously maintains an individual, pairwise, and cluster keys to support in-network processing. However, it still lacks support for long-distance communications.

3. THE PROPOSED SECURE ROUTING

We first give an overview of the proposed approach and then describe details of its components.

3.1 Overview

We propose lightweight, secure routing protocols for a network of resource-constrained sensor devices. The proposed protocols:

—are tailored to secure communications between *distant* sensor nodes;

—are *flexible* enough to make a tradeoff between security and energy consumption;

—augment the existing localized key sharing with a *global* distributed key sharing infrastructure;

—preserve *compatibility* with existing link-layer security mechanisms; and

—support

1. *confidentiality* that protects data from unauthorized disclosures,
2. *data integrity* that does not allow unauthorized creation or modification of data,
3. *authenticity* that correctly associates the sensor IDs with data/services/keys, and
4. *resilience* against service-disruption (e.g., DoS) attacks.

These salient features will enable the proposed protocols to play a crucial role in securing many critical applications/services, such as data storage based on GHT, data dissemination, location management, and intercluster communications for cluster-based networks.

We use the following widely accepted assumptions.

A1. Each sensor is uniquely identified by its location estimate obtained from the localization service executed during bootstrapping (Section 3.2.2).

A2. Used as an underlying routing protocol is a well-known geographic forwarding protocol (GFP) [Karp and Kung 2000; Jain et al. 2001], in which a

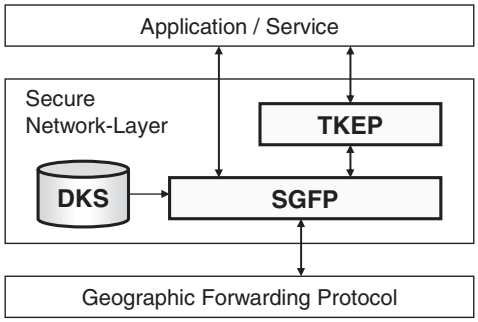

Fig. 1. The proposed secure network layer.

source or an intermediate sensor sends each packet to one of its neighbors geographically closest to the packet's destination.

A3. For the proper operation of GFP, each sensor keeps a list of its neighbors' locations based on BEACON packets exchanged.

For compatibility, each sensor maintains its own key, as well as the cluster (shared within its cluster) and/or pairwise keys (shared with its neighbors). The cluster and pairwise keys are created during the initial bootstrapping of the network, via the cluster-based key management and key predeployment schemes, respectively. The challenge in this environment is that each sensor does not have keying relationships with most of other sensors located outside the local cluster and/or its neighborhood. To address this challenge, we present a distributed key-sharing scheme in which a chosen sensor elects, from the entire network, a small number of other sensors to serve as distributed key servers (DKSs) by creating/sharing unique keys. The proposed scheme builds an efficient, global key-sharing framework that covers the entire network throughout its lifetime.

Built on top of this framework, we propose two protocols for secure routing: a *secure geographic forwarding protocol* (SGFP) and a *temporal-key establishment protocol* (TKEP). SGFP provides a secure distributed lookup service that executes recursive DKS queries, each secured with a shared key, until a neighbor of the destination node is found. This can be viewed as a secure extension of the distributed hash table (DHT) routing [Li et al. 2000; Stoica et al. 2001; Ratnasamy et al. 2001; Rowstron and Druschel 2001] for peer-to-peer and ad-hoc networks. TKEP then relies on SGFP to realize the purely symmetric cipher-based key setup. As shown in Figure 1, these security building blocks interact with each other to form a secure network layer between applications/services and GFP. Accordingly, applications/services invoke either SGFP (for per-packet protection) or TKEP (for securing session establishment), which will then ask GFP to deliver the security-added packets. Note that we use GFP simply as a mechanism to forward the packets created/processed securely by SGFP and TKEP, and that security depends on how each packet's payload is handled, but not on how the packet gets to its destination.

We define and use three types of unique symmetric keys: (1) a *sensor key* (SK) individually generated by each sensor, (2) a *mission key* (MK) shared by a sensor and each of its DKSs (or each of its direct neighbors), and (3) a *temporal key* (TK) for encrypting/authenticating a data traffic session. We use notation, SK_s, $MK_{i,j}$ and $TK_{i,j}$, to refer to the SK of sensor s, the MK and TK shared between sensors i and j, respectively. During their initialization, sensors i and j agree on a unique $MK_{i,j}$. $TK_{i,j}$ will be established, whenever needed, between i and j using TKEP.

3.2 Distributed Key Sharing

The distributed key sharing is proposed to address the challenges in dealing with a large number of battery-powered sensors. It differs from the dedicated key-server solution in that (chosen) sensors act both as *key servers* (i.e., DKSs) storing a small number of MKs shared with other sensors chosen in the network coverage area, and as *key clients* querying DKSs for secure routing. It is essentially a distributed database that is cooperatively maintained and accessed.

3.2.1 *DKS Architecture.* To control the overhead of initially setting up DKSs, we enforce that a sensor, called a *DKS sensor*, builds the DKS map only if it has no DKS sensor within its neighborhood (just like Bluetooth's piconet). Otherwise, a sensor establishes a pairwise key shared with one of its neighbors that acts as a DKS sensor, thereby relying on that sensor for secure routing. That is, if a sensor has not heard from the DKS sensor (e.g., via a BEACON packet), it declares itself as a DKS sensor by broadcasting this decision to all its neighbors. This way, only a small fraction ($\leq$ 10%) of sensors will execute the DKS setup process.

Figure 2 shows how each DKS sensor constructs its *own* map of DKSs associated with its geographic location. It partitions the network into squares of various levels and elects DKS(s) in each square. A sensor elects its DKSs based on the facts that (1) distribution of its DKSs should be denser in its proximity, but sparser farther away from it, and (2) DKSs are direction-aware, i.e., one DKS for each of eight directions at the same level.[1] Here we allow up to level-K squares and DKSs.

The DKS map of a DKS sensor s is built as follows. First, a level-0 square, $L_0(s)$, with a predefined area λ^2, is formed around s, which then establishes pairwise keys shared with each of the sensors in $L_0(s)$, as well as the shared cluster key. Second, eight level-1 squares, $L_{1,m}(s)$, $m = 1, \ldots, 8$, each of the same size as $L_0(s)$, are located around $L_0(s)$. The DKS sensor closest to the center of each $L_{1,m}(s)$ is then selected as a level-1 DKS, denoted as $\mathcal{DKS}_{1,m}(s)$. Third, eight level-2 squares, $L_{2,m}(s)$'s, are formed to surround the cumulative area of level-0 and all level-1 squares, each with area $9\lambda^2$, i.e., nine times the area of $L_0(s)$. Again, the DKS sensors closest to the center of each level-2 square are elected as level-2 DKSs, i.e., $\mathcal{DKS}_{2,m}(s), m = 1, \cdots, 8$. Likewise, higher-level squares (up to level-K) are constructed and DKSs elected. All DKS sensors in the network construct their own map of DKSs by using the above procedure.

[1]The eight directions—NW, N, NE, E, SE, S, SW and W—are assign numbers $1, \ldots, 8$, respectively.

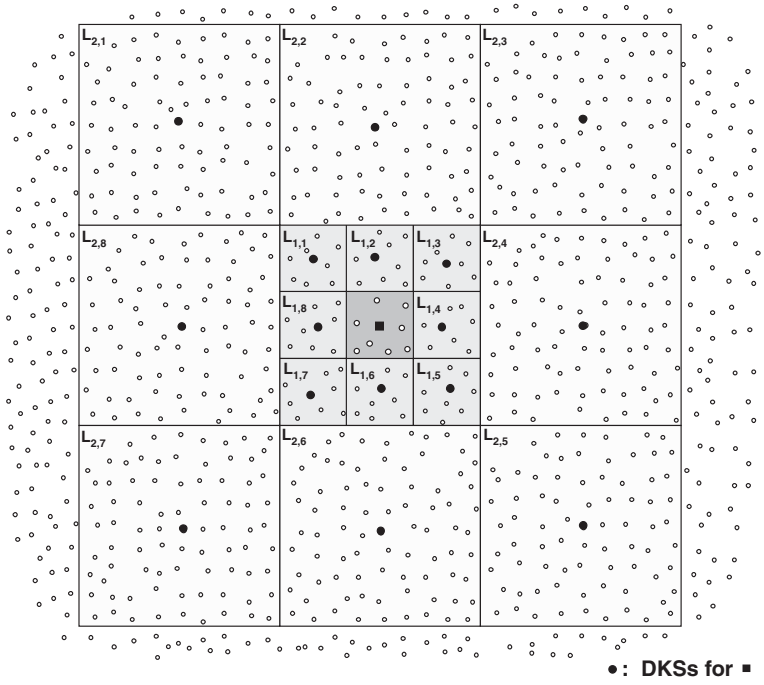

Fig. 2. The map of DKSs for a DKS sensor (located at the center) when $K = 2$. It partitions the network into squares of levels 1 & 2 and elects DKS for each square.

Each DKS sensor elects no more than $8K$ DKSs when the network has been configured to have up to level-K squares. For instance, when $K = 2$, it elects up to 16 DKSs regardless of the total number of sensors in the network. The average value (over the entire network) is slightly less than $8K$, since sensors near the border of the network area would elect less than 8 DKSs for outer levels.

3.2.2 *Initial Bootstrapping.* Each and every sensor, right after its deployment, executes the conventional bootstrapping process that consists of the following sequential steps:

B1. determine its location estimate by running the (attack-tolerant) localization algorithm with other sensors;

B2. generate its own SK;

B3. set up pairwise keys (and a cluster key) with its neighbors (and a cluster) according to the existing key predeployment schemes such as [Eschenauer and Gligor 2002]; and

B4. elect a DKS sensor (either itself or a neighbor) with all its neighbors.

In step B1, we may apply the existing localization protocols that can defeat and/or resist localization-targeted attacks. One of such protocols is VeIL [Park], under which sensors cooperatively safeguard the localization services by exploiting the high spatiotemporal correlation between adjacent nodes, hence, requiring no cryptographic bindings among sensors at this stage. For details of VeIL, see [Park].

After step B3, any sensor s establishes cryptographic bindings between the pairwise keys and the neighbors' IDs leading to the construction of secure links with each of its neighbors, e.g., it agrees on a unique $MK_{s,g}$ with its neighbor g. The key predeployment method, albeit communication intensive, has been widely used in, and is known to be a feasible solution for, resource-constrained sensor devices. The communication overhead would not be an issue as long as the bootstrapping is executed only once per sensor.

3.2.3 *DKS Setup.* After bootstrapping, a sensor s, if elected as a DKS sensor, executes the DKS setup procedure to build key sharing relationships with other DKS sensors. This is to construct, for each of the DKS sensors, a cryptographic binding between a MK and the IDs of the two remote DKS sensors based on already-established pairwise keys.

During the DKS setup, s sequentially contacts each of its candidate DKSs and establishes a shared MK. That is, **for** $1 \leq k \leq K$ and $1 \leq m \leq 8$, sensor s:

1. identifies the location of the center of $L_{k,m}(s)$, based on its own location and DKS map;
2. discovers $f_{k,m} = \mathcal{DKS}_{k,m}(s)$ that is closest to this desired location; and
3. sets up a shared $MK_{s,f_{k,m}}$ with $f_{k,m}$.

Here, $f_{k,m}$ can be found easily as follows. First, an $f_{k,m}$-discovery packet is relayed using GFP until it arrives at the first sensor that has the center of $L_{k,m}(s)$ within its transmission range. Note that GFP [Karp and Kung 2000] is guaranteed to find such a sensor, if it exists, even in the presence of hole(s) along the path. If the sensor has a neighbor closer to the center of $L_{k,m}(s)$, it forwards the packet to that neighbor; otherwise, the sensor determines itself or its DKS sensor as $f_{k,m}$ of s. If the sensor fails to find $f_{k,m}$ (possibly because of a hole near the desired location), it may flood the received packet within its proximity to find a DKS sensor eligible to be $f_{k,m}$. Also, note that the use of insecure GFP in the $f_{k,m}$ discovery does not cause any security vulnerability, because GFP is just an underlying routing protocol used to deliver the security-added packets that will be processed by external mechanisms like the DH protocol, as described next.

We present three different ways of setting up $MK_{s,f_{k,m}}$. First, the DH protocol can be applied to establish a unique MK between s and $f_{k,m}$. The use of DH protocol at this stage is acceptable, because it is executed only during the DKS setup while future transactions will be secured by our proposed protocols. Second, s and $f_{k,m}$ can use the key predeployment scheme (over a multihop path) to find if they have a common preloaded key. If so, they can come up with their own $MK_{s,f_{k,m}}$ using this common key. Third, we may simplify the MK establishment under the assumption that each sensor is safe against physical attacks for a certain period of time after its initial deployment, during which it can complete the DKS setup.[2] Then, relaying sensors would not harm the DKS setup process and, hence, s and $f_{k,m}$ exchange their SKs and random numbers, n_1 and n_2, and then compute $MK_{s,f_{k,m}}$ using these values. The

[2]The rationale behind this assumption is that it would take time for an adversary to locate/capture the victims.

Level	Location	Key
1	$x_{f_{1,1}}$, $y_{f_{1,1}}$	$MK_{s,f_{1,1}}$
	…	…
	$x_{f_{1,8}}$, $y_{f_{1,8}}$	$MK_{s,f_{1,8}}$
…	…	…
K	$x_{f_{K,1}}$, $y_{f_{K,1}}$	$MK_{s,f_{K,1}}$
	…	…
	$x_{f_{K,8}}$, $y_{f_{K,8}}$	$MK_{s,f_{K,8}}$

Fig. 3. The routing table of s, having $8K$ DKS entries.

delivery of $\{SK_s, n_1\}$ from s to $f_{k,m}$ is protected by pairwise keys of intermediate sensors: s uses its pairwise key to get to one of its neighbors, which will then forward it to the next sensor closer to $f_{k,m}$ after reencrypting it with its own pairwise key; this process will be repeated until $\{SK_s, n_1\}$ arrives at $f_{k,m}$. Likewise, $\{SK_{f_{k,m}}, n_2\}$ will be delivered from $f_{k,m}$ to s via hop-by-hop transcoding. Finally, s and $f_{k,m}$ computes $MK_{s,f_{k,m}} = F(SK_s, SK_{f_{k,m}}, n_1, n_2)$, where F is a fixed hash function. Again, the hop-by-hop transcoding is used only once during this MK setup while SGFP will be applied to protect future communications from attacks and compromised sensors.

A complete DKS setup protocol is summarized as follows. For each $k \leq K$ and $m \leq 8$,

D1. s generates a packet containing the location of $L_{k,m}(s)$'s center and the security context (necessary to set up MK), and geographically forwards the packet toward the center of $L_{k,m}(s)$;

D2. a sensor receiving the packet:

D2.1. **if** it (or its neighboring DKS sensor) is closest to the location marked in the packet, declares itself (or its neighbor) as $f_{k,m}$, who will then reply back to s with its own location and security context;

D2.2. **else**, relays the received packet to the next hop toward the center of $L_{k,m}(s)$;

D3. both s and $f_{k,m}$ compute (or agree on) a unique $MK_{s,f_{k,m}}$ and store it in their routing table.

Figure 3 shows the structure of the routing table of DKS sensor s built according to the above procedure. It consists of three fields—the DKS level, the location, and a shared MK—for all $8K$ DKS sensors chosen during the DKS setup. Hence, each DKS sensor stores, at most, $8K$ additional MKs (e.g., 16 MKs when $K = 2$), which is significantly fewer than the requirement of N^2 MKs (where N is the total number of sensors in the network) needed for maximum security. This indicates that our distributed key sharing is very efficient in terms of the key storage requirement, incurs a very low degree of key sharing, and scales well with the network size.

When $K = 2$, the DKS setup incurs eight medium-distance (level-1) and eight long-distance (level-2) handshaking processes to less than 10% of sensors in

the network. This overhead of initially setting up DKSs should not be an issue, as it takes place only *once* in the beginning (and incremental reconfiguration of DKSs thereafter as described in Section 3.5) and only those chosen sensors participate in computing/sharing MKs. Moreover, this overhead is insignificant, compared to the overhead of localization (that must be executed for other applications/services). Both SGFP and TKEP, even with this overhead, will extend the lifetime of the network significantly by consuming much less energy than existing schemes, especially for securely transporting long-distance traffic.

3.3 Secure Geographic Forwarding

SGFP is a multihop routing protocol that establishes a secure, *unidirectional* path between two arbitrary sensors based on the limited and distributed knowledge of DKS sensors' locations/MKs. SGFP achieves a high level of tolerance/robustness to sensor compromises by minimizing the number of transcodings for each route discovery.[3]

We use the term "link" to refer to GFP between two sensors in the DKS relationship. Each $f_i \rightarrow f_j$ is said to be a level-k link if $f_j = \mathcal{DKS}_{k,m}(f_i)$. The security is preserved over each link by using the shared MK, e.g., MK_{f_i,f_j} for the $f_i \rightarrow f_j$ link. Since each link uses a unique MK, the edge sensor relaying the packet from one link to another should transcode the packet, e.g., from MK_{s,f_i} to MK_{f_i,f_j}.

The heart of SGFP is the DKS selection rule that determines a DKS to establish a link to: if a node is not a DKS sensor, it chooses a neighbor DKS sensor; else, it chooses one of its DKSs that significantly reduces the distance to the destination. Using this simple DKS selection rule, SGFP constructs the path by concatenating multiple GFP links, each of which is secured by a distinct MK. Hence, SGFP successfully sets up a path to d upon successful completion of all of its underlying GFP operations. Described below is SGFP:

S1. **if** a sensor s is a DKS sensor,
- *S1.1.* it forwards the packet to an intermediate sensor $f_{k,m}$ ($k \leq K$)—instead of the destination d—that is closest to d among DKSs listed in its routing table;
- *S1.2.* $f_{k,m}$, upon receiving the packet, forwards the packet to one of its own DKSs closer to d;
- *S1.3.* the subsequent forwarding is handled in the same way until the packet reaches $f_{1,m'}$ for which $d \in L_0(f_{1,m'})$; and, finally,
- *S1.4.* $f_{1,m'}$ uses a pairwise/cluster key to deliver the packet to d;

S2. **else**, it asks its nearby DKS sensor to deliver the packet to d (through steps S1.1–S1.4).

SGFP takes a "divide-and-conquer" approach: when d belongs to a level-k square of s where $1 \leq k \leq K$, SGFP builds at most k links, $s \rightarrow f_{k,*} \rightarrow \cdots \rightarrow f_{1,*}$, until a DKS sensor belonging to $L_0(d)$ is found. Figure 4 illustrates how

[3]The transcoding of a packet consists of (1) decrypting and verifying the authenticity of the packet using the current MK, then (2) reencrypting and recomputing the message authentication code with the next MK.

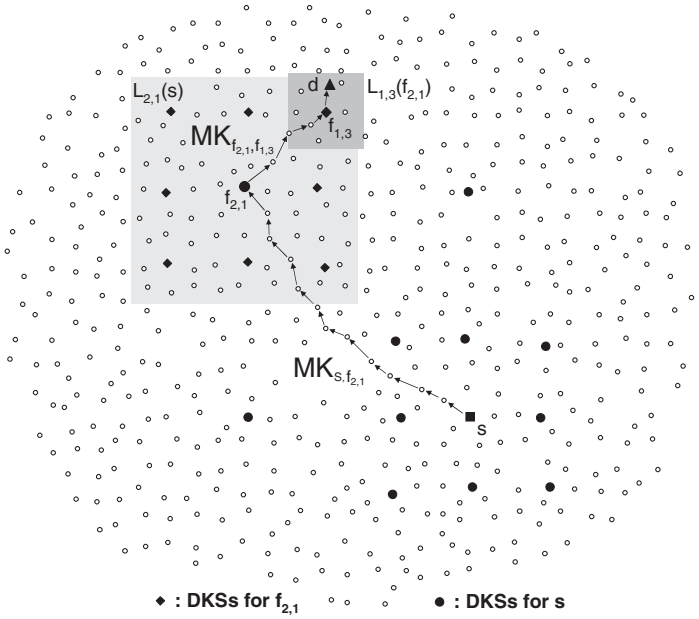

Fig. 4. SGFP from s to d.

SGFP forwards the packet from s to d when d resides in $L_{2,1}(s)$. s (being a DKS sensor) selects $f_{2,1} = \mathcal{DKS}_{2,1}(s)$ according to the DKS selection rule, encodes the packet with $MK_{s,f_{2,1}}$, then geographically forwards it to $f_{2,1}$. Since $f_{2,1} \neq d$, $f_{2,1}$ repeats the same procedure. d now belongs to $L_{1,3}(f_{2,1})$ and, hence, $f_{2,1}$ gets $f_{1,3} = \mathcal{DKS}_{1,3}(f_{2,1})$, transcodes the packet with $MK_{f_{2,1},f_{1,3}}$, and geographically forwards it to $f_{1,3}$. Note that $f_{2,1}$ suffices to search up to level-1 DKSs. Finally, $f_{1,3}$ finds that d is its neighbor, thus forwarding the received packet to d using the pairwise or cluster key.

When d is outside all level-K squares, s chooses and forwards the packet to its level-K DKS, $f'_{K,*}$ closest to d. If d belongs to one of $f'_{K,*}$'s squares, the packet is routed via $f'_{K,*} \rightarrow f_{K,*} \rightarrow \cdots \rightarrow f_{1,*}$; otherwise, $f'_{K,*}$ again forwards the packet to one of its level-K DKS. Therefore, it incurs one or more level-K links in the beginning.

It is possible that d is not directly reachable from $f_{1,*}$ due mainly to the errors in the location estimates, irregular deployment of sensors, absence of the shared key, etc. In such a case, $f_{1,*}$ selects, and forwards the packet to, its neighbor g which is closer to d. Then, g will likely have a pairwise key, $MK_{g,d}$, shared with d, and successfully deliver the packet. Otherwise, g will repeat the same procedure. Thus, SGFP incurs additional (hop-by-hop) transcoding to reach d.

3.4 Temporal-Key Establishment

When two sensors need to maintain a persistent session for a certain period of time, it is preferable to establish a shared TK for that session if they do not yet have a shared key. To meet this need, we present TKEP that enables any two sensors to agree on a common TK, meeting the following two requirements:

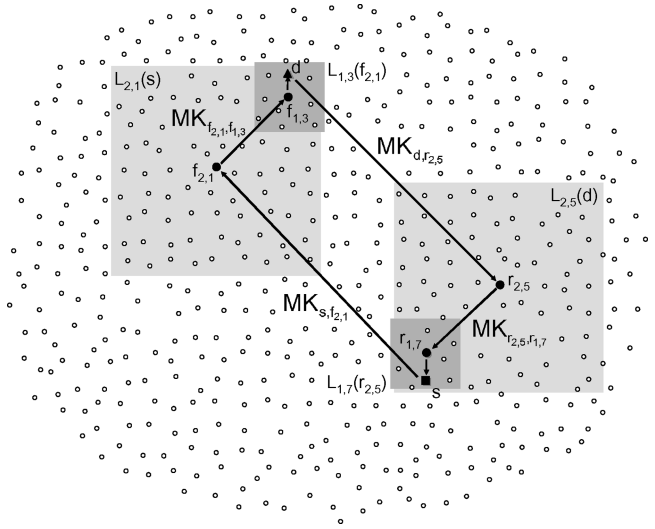

Fig. 5. TKEP between s and d.

R1. none of the two sensors can dictate the value of TK, and

R2. the rest of the network should not be able to duplicate the TK established between the two.

TKEP is a purely symmetric cipher-based key setup protocol and, hence, serves as a lightweight alternative to the resource-demanding DH protocol in a large-scale network of sensors.

3.4.1 *Basic TKEP.* Built on top of SGFP, TKEP realizes the concept of spatial diversity by exploiting the novel DKS infrastructure. Suppose s initiates TKEP between itself and d where both s and d are DKS sensors (for ease of description). If forward and backward SGFP paths (s-to-d and d-to-s, respectively) were run via different sets of DKSs, both sensors could contribute to TK via each of the two paths, while no other sensors could duplicate the complete TK. In practice, they exchange random seeds, R_s and R_d (generated by s and d, respectively) using SGFP and, then, individually compute $TK_{s,d} = F(R_s, R_d)$, where F is a fixed hash function. TKEP consists of the following steps.

T1. In the forward path, s:
  T1.1. randomly generates R_s;
  T1.2. transmits R_s to d using SGFP.

T2. In the backward path, d:
  T2.1. randomly generates R_d;
  T2.2. transmits R_d to s using SGFP.

T3. s and d individually compute $TK_{s,d} = F(R_s, R_d)$.

Figure 5 illustrates how TKEP works between s and d. Let $f_{k,*}$'s and $r_{k,*}$'s refer to DKSs on the forward and backward paths, respectively. Then, the forward and backward SGFP paths are routed via $\{f_{2,1}, f_{1,3}\}$ and $\{r_{2,5}, r_{1,7}\}$, respectively.

As shown in this example, TKEP satisfies the requirement R1 that both s and d must equally contribute to the value of TK. The requirement R2 (keeping the rest of the network from duplicating the TK) would be met if no sensors other than s and d can get the plaintexts of both R_s and R_d. This condition is automatically met by using SGFP, as explained next.

Thanks to the way DKSs are constructed, all $f_{k,*}$'s and $r_{k,*}$'s must be distinct sensors. For example, in Figure 5, $f_{2,1}$ and $f_{1,3}$ reside in $L_{1,7}(d)$ (one of d's level-1 squares) and $L_0(d)$ (d's level-0 square), respectively, while $r_{2,5}$ and $r_{1,7}$ must belong to the d's level-2 square ($= L_{2,5}(d)$) and, trivially, DKSs in each of the two paths must be distinct. Therefore, no sensor can serve as DKS for both forward and backward paths. Moreover, all the MKs involved in TKEP are unique. Each link is thus secured using MK known only to the end sensors, implying that no intermediate sensors on that link can decipher it. As a result, sensors other than s and d may decrypt at most one of R_s and R_d, but cannot reproduce both of them. For example, even if the link $d \rightarrow r_{2,5}$ uses $f_{1,3}$ as a relaying node (in Figure 5), $f_{1,3}$ does not know $MK_{d,r_{2,5}}$ and, hence, it can neither decrypt R_d nor construct $TK_{s,d}$.

3.4.2 *TKEP with Randomization.* To withstand attacks from compromised DKSs,[4] s (and d) "randomly" selects the next DKS $f_{k,*}$ (and $r_{k,*}$) to form an SGFP path to d (and s). Note that only s and d randomly select the next DKS while intermediate DKSs follow the basic TKEP, thus guaranteeing delivery of packets to their destination. That is, if the basic TKEP converges, so does this scheme.

Because the initial DKSs are randomly picked, the odds that two compromised nodes (sharing common information) are on each of the two paths are very small. Thus, it is very difficult for the adversary to create a *general* policy to choose victims to compromise. For example, he has to compromise 14 sensors to figure out TK of a single (s,d) pair. However, this would be too much of an effort for the attacker to make because there are no hot spots or dedicated devices (like cluster heads) in our distributed environment and, hence, he has no other way but to compromise as many sensors as possible to take control of the entire network.

3.4.3 *μ-Split TKEP.* We may achieve the highest-level protection of TK for a session between a pair of distant nodes s and d in the presence of compromised nodes by splitting each random seed into μ pieces and then forwarding each of them over a randomly chosen SGFP path. Note that there may be as many as 8^K distinct routes because it may choose one of 8 DKSs for each level. As a result, both s and d collect all the pieces and compute $TK_{s,d} = F(R_{s,1}, \ldots, R_{s,\mu}, R_{d,1}, \ldots, R_{d,\mu})$. This scheme is capable of trading security for computation (and energy consumption) via the choice of μ.

3.5 Steady-State Operations

We now describe the DKS reconfiguration process that adds/removes DKS sensors to/from the DKS infrastructure and renews shared MKs.

[4]See Section 4.3 for details of this attack scenario.

3.5.1 *DKS Reconfiguration.* The DKS architecture is reconfigured *periodically*, setting up a new DKS sensor s to replace the old one c within its proximity. It is compatible with the power-saving mode operation [Chen et al. 2001; Miller and Vaidya 2005], allowing sensors to sleep to conserve energy as well as balance their energy consumption. Using this DKS reconfiguration, nearby sensors periodically rotate (in an *a priori* agreed-on order) the role of a DKS sensor among themselves while keeping the rest in a low-power state if the power-saving mode was activated. This enables our proposed protocols to resist attacks on DKS sensors because a malicious device cannot dictate the role of a DKS sensor, thus significantly lowering the risk of network-wide service disruption and stressing the adversary to compromise many sensors. The frequency of reelecting DKS sensors is a network-wide design parameter that makes a tradeoff between security and energy consumption. It can be preconfigured at the time of deployment.

The DKS reconfiguration that replaces c with s (a neighbor of c) consists of the following operations: s discovers the $8K$ sensors—that elected c as their DKS—this operation can be done efficiently via the following steps. For each $k \leq K$ and $m \leq 8$:

1. s contacts $g_{k,m}$ that is closest to the center of $L_{k,m}(s)$ by executing the discovery protocol of DKS setup;
2. $g_{k,m}$ broadcasts locally (within at most a few hops) to find $f_{k,m}$ that has a shared MK with c; then
3. $f_{k,m}$ deletes (or deactivates) the entry corresponding to c from its routing table and elects s as a new DKS by establishing a shared $MK_{f_{k,m},s}$ with s.

Upon successful completion of the above operations, s serves as DKSs for the $8K$ sensors that have been relying on c. Note that $f_{k,m}$ expects to be contacted periodically by no more than one sensor (near the current DKS) and, hence, any activity deviating from this normal behavior will be regarded as an attack. Note, also, that this scheme achieves fair energy consumption among sensors regardless of the addition of DKS sensors.

3.5.2 *Renewal of MKs.* It is required to renew keys in stream ciphers such as that of TinySec resulting from the limitation in the maximum number of packets that can be transmitted using the same key. In our proposed key sharing, two DKS sensors can set up a new MK from two random seeds, each (1) generated independently and individually, (2) encrypted with the current shared MK, and then (3) exchanged via GFP links. The MK can be renewed periodically to simplify the implementation on sensor devices.

4. SECURITY ANALYSIS

This section discusses how our proposed protocols defend against possible attacks.

4.1 Prevention of Sybil Attacks

It is possible that a compromised sensor joins the network and creates/uses many different IDs/locations to mount a sybil attack. However, we detect/

prevent sybil attacks by providing countermeasures in all our protocols as described below.

First, the DKS setup and reconfiguration mechanisms can defeat sybil attacks as follows. It is impossible for a DKS sensor to claim multiple locations because the eligibility for DKS, as well as the underlying packet delivery, depend on locations. That is, a malicious DKS sensor cannot make its (multiple) fake locations inserted into the others' routing tables during the DKS setup. Moreover, the sybil attack is not an issue for non-DKS sensors, because they should ask, for secure routing, their own DKS sensors, each of which may act as a central entity for checking 1-to-1 correspondence between IDs and locations of sensors in its neighborhood.

Second, a malicious sensor node cannot claim arbitrary locations due mainly to the correlation among locations of neighboring sensors. Let's assume a (simple or two thirds) majority of neighbors of a sensor within the region of interest are well behaving. Then, if a malicious device announces a new location without changing the ID, its neighbors would easily detect this discrepancy via cooperative location validation among themselves to blacklist/block the misbehaving sensor from the network. Thus, the malicious node must risk being detected if the false location deviates too far away from its true location because its unusual distances to its neighbors make it conspicuous in their neighbors' routing tables. Otherwise, the bogus locations would not impose more threat than a compromised node, which does not lie about its location.

Third, the malicious node may claim to be a new sensor node by creating the binding of ID and falsified location. In this case, however, it must go through the bootstrapping and DKS setup phases to be qualified as a legitimate sensor as well as establishing shared keys. If addition of new sensors is disallowed after network-wide bootstrapping, the neighbors can simply ignore the new ID from the network; otherwise, we may easily capture the misbehaving device by applying a strong access control mechanism, such as the program integrity verification (PIV) in [Park and Shin 2005b], during the initial setup.

In summary, the sybil attack takes place in a decentralized virtual network, the ID space of which is *completely* decoupled from physical network connectivity [Douceur 2002]. However, this is clearly not the case in our proposed protocols and environments thanks to their reliance on spatial correlation. A systematic, distributed way of detecting invalid locations (i.e., far from the majority of others' locations in the neighborhood) has been proposed in [Park] for the development of attack-tolerant localization service.

4.2 Attacks on DKS Setup/Reconfiguration

The key management protocols would usually become susceptible to masquerading and man-in-the-middle attacks without proper key authentication that cryptographically binds the key and the communicants' IDs. Fortunately, this security risk does not exist in our distributed scheme, because it first establishes the "local" cryptographic bindings using a well-known method and then builds the proposed "remote" bindings using the thus-established local bindings. Therefore, if the former resists the above-mentioned attacks, so does our

distributed scheme. Moreover, the DKS setup/reconfiguration is safe against man-in-the-middle attacks if the DH protocol (one of three options as described earlier) is used to establish MKs.

A malicious device can appoint itself as a DKS sensor to intercept messages to be transcoded by itself. However, our proposed protocols successfully tolerate this threat via DKS reconfiguration (Section 3.5.1), under which all sensors in the neighborhood periodically rotate the role of a DKS sensor. Thus, the malicious device cannot always eavesdrop messages, because it can neither dictate the role of DKS sensor nor freely initiate a DKS reconfiguration, which, in turn, significantly lowers the risk of abusing the DKS reconfiguration process. Moreover, as described in Section 3.4.2, the attacker that owns multiple compromised slaves can only decipher a small fraction of network traffic, thanks to our distributed environment that uses neither hot spots nor dedicated devices. To further thwart attacks from compromised nodes disguised as DKS sensors, we may apply a soft tamper-proofing protocol [Park and Shin 2005b] that does a deep inspection of the program code of the node chosen as a DKS sensor to make sure it is genuine.

Finally, one may argue a malicious device can establish MKs with a large number of other DKSs by replying, to the packet destined for some remote location, as though it were the correct destination. However, this belongs to the category of sybil attacks and, hence, a cooperative defense mechanism like the one in [Park] can be applied to defeat this attack.

4.3 Attacks on TKEP/SGFP

To establish a TK between s and d, s sends R_s through the forward SGFP path, and d replies with R_d through the reverse SGFP path. With a collusion attack, if the two malicious nodes, m_1 and m_2, are on each of the two paths, they may share information to reconstruct the TK. However, this collusion attack is quite opportunistic in that the secrecy of TK is broken only if the attacker happens to own DKSs on both paths. As analyzed in the Appendix, this probability is very small unless he compromises a large number of sensors in the entire network. Moreover, the application of randomization and μ-split schemes in selecting DKSs, as well as DKS reconfiguration, make TK eavesdropping very unlikely. Therefore, TKEP "tolerates" collusion attacks by degrading its security gracefully as the number of undetected compromised sensors increases.

We do not consider DoS attacks on SGFP (precisely speaking, on GFP) assuming the existence of an external countermeasure to the DoS attacks.

4.4 Tolerance to Physical Attacks

Both SGFP and TKEP tolerate physical attacks very well since they are robust to compromises of individual sensors, thanks to the distributed key sharing that allows each DKS sensor to share only a small number of MKs. By compromising/owning a sensor c, an attacker can only take over the data traffic passing through and transcoded by c. Therefore, the only way to take control of a significant portion of network traffic is to capture/compromise as many sensors as possible. Moreover, there is no difference between random and "planned"

Table I. P_{SGFP} and P_{TKEP_μ} versus p_c

p_c	P_{SGFP}	P_{TKEP_1}	P_{TKEP_3}
0.001	0.0041	0.000017	1.95×10^{-14}
0.005	0.0202	0.000409	2.90×10^{-10}
0.010	0.0401	0.0016	1.75×10^{-8}
0.020	0.0791	0.0062	0.99×10^{-6}
0.050	0.1887	0.0356	1.66×10^{-4}

selection of victims. For instance, the adversary can capture a "cut" through the network to monitor all traffic over the cut. However, he can decode only a small portion of the traffic that happens to have been encoded with the key he knows of. Consequently, the security will be degraded gracefully as the number of undetected compromised sensors increases. This is important as it stresses the adversary's attempts to subvert the entire network. Note that the adversary can still eavesdrop communications of specific sensors for a certain period of time by compromising those sensors serving as DKSs for them.

To quantify the tolerance/robustness of SGFP and TKEP to sensor compromises, we derive, in the Appendix, the following probabilities: (1) P_{SGFP} of eavesdropping SGFP packets and (2) P_{TKEP_μ} of breaking μ-split TKEP. Both probabilities provide a useful basis for evaluating robustness to sensor compromises in very large-scale sensor networks. Table I shows the numbers obtained from Eqs. (10–12) (derived in the Appendix) while varying the ratio of the number of compromised sensors to the total number of sensors (fully captured in p_c) when $K = 3$ and $\alpha = 1.2$.[5] P_{SGFP} is shown to be proportional to p_c, i.e., approximately four times p_c. This means that the required number of sensors to be compromised will be very large, demonstrating the robustness of SGFP. P_{TKEP_μ} is shown to be almost negligible, when there are a small number of compromised sensors, and to increase gradually with p_c. Moreover, when $\mu = 3$, it is very unlikely for an adversary to be able to eavesdrop even after compromising 5% (e.g., 500 out of 10,000) of sensors, indicating TKEP's robustness to sensor compromises.

5. PERFORMANCE EVALUATION

We use simulation to evaluate the performance of proposed protocols in terms of the overhead and energy consumption. We first quantify the initial DKS setup overhead, then compare the energy consumption of TKEP and the DH key setup protocol, and finally evaluate the security/energy tradeoffs.

5.1 Simulation Environment

Although *ns-2* is widely used to simulate network protocols, it cannot be used to evaluate our proposed protocols for the following reasons. First, the ns-2 simulation is limited to network sizes in the order of a couple of hundreds of sensors [Ratnasamy et al. 2003] and, hence, it is very difficult, albeit not impossible, to simulate very large-scale networks. By contrast, SGFP and TKEP

[5]See the Appendix for the definition of p_c, K and α.

are tailored to very large-scale networks of thousands to millions of sensors, thanks to their capability to work at arbitrary K values, meaning that *ns-2* is unsuitable for the evaluation of these protocols. Second, we do not need a detailed simulation of link-layer behavior, packet losses, sensor dynamics, and the effects of energy depletion, because we are only interested in network-layer behaviors. We, therefore, developed a customized simulator with a simple radio transmission model: at any time, each sensor can directly communicate with all the sensors within its transmission range, and the packet delivery to neighbors is instantaneous and error-free. It is reasonable to use this simplified model as sensors are stable and stationary, and hence, the neighbors of each sensor do not vary with time [Ratnasamy et al. 2003].

Our simulation environment is based on a network of 10,000 sensors, placed in a square area of 200×200 [m^2] and electing up to level-3 DKSs ($K = 3$). Each sensor has a radio transmission range of radius 5–6 [m].[6] The location (estimate) of each sensor is generated randomly within the network coverage area. That is, we do not simulate the localization service because it is not our intended contribution in this paper. The GFP is implemented/simulated as follows: either the source or the relaying sensor determines its next-hop sensor as the one closest in the direction to the destination. The distance to a neighbor is not considered, as the selection of next-hop based on specific distance-based policies (e.g., either minimum- or maximum-distance policies) has its own merits and demerits. We, therefore, only use the direction-based policy, assuming that each sensor can adjust the transmission power according to the distance to the next-hop sensor so as to minimize interferences.

5.2 Overhead of DKS Setup

We measured the total number of packets generated and sent/relayed during the DKS setup (counting each hop as a distinct packet) while varying the size of level-0 square and the transmission range. We also counted the number of DKS sensors chosen during this setup; it elected 6.8–9.7% of sensors as DKS sensors, depending on the transmission range (i.e., the larger the transmission range, the smaller the number of DKS sensors). We then divided these values by the total number of sensors to compute the average number of packets relayed per sensor. This average behavior is important, because all sensors take turns to serve as DKS sensors throughout their lifetime (Section 3.5). Figure 6 plots the results: each sensor received/relayed less than 20 packets during the DKS setup. Moreover, when K is set to 2, this overhead gets even smaller. This initial DKS setup overhead is reasonable (and low) considering the localization overhead that takes place prior to the DKS setup.

5.3 Energy Consumption

Experimental results [Malan 2004; Carman et al. 2000] have shown public-key algorithms to consume a significant amount of energy. To compare the energy

[6]Note that this choice is just for the purpose of simulation. The realistic values for transmission range and network coverage area would depend on other factors, such as the accuracy of localization service.

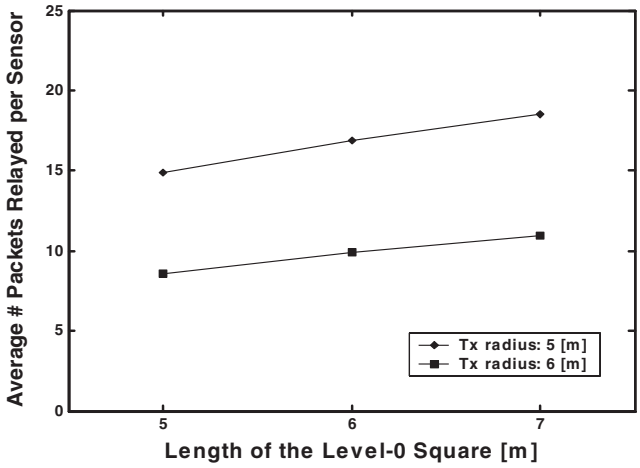

Fig. 6. The average number of packets relayed by each sensor.

Table II. Energy Costs of TinySec and DH Protocols

	TinySec		DH
	Encryption	MAC	Protocol
Energy [mJ]	0.04796	0.06677	1185

Table III. Comparison of Energy Costs for TKEP and DH

	Energy [mJ]		TKEP/DH
μ	TKEP	DH	[%]
1	2.2603	1185	0.19
2	5.4382		0.46
3	8.1573		0.69

consumption of TKEP and DH protocols, we use the measurement results of [Malan 2004] for the energy costs of relevant ciphers (summarized in Table II).

As derived in the Appendix, the average number of transcoding attempts per TK setup for both basic and μ-split TKEP are $2(K+\alpha+p_n-\frac{1}{8})$ and $2\mu(K+\alpha+p_n+\frac{7}{8})$, respectively. Using the fact that each transcoding requires two TinySec encryptions and MAC computations, we can compute the energy consumption of TKEP given the protocol parameters such as μ, K, α, and p_n. When $K=3$, $\alpha=1.25$, and $p_n=0.9$, Table III presents, as a function of μ, the energy costs for TKEP and the DH protocol, and the amount of energy savings by TKEP. The result shows that TKEP consumes energy far less than 1% of the DH protocol, confirming its high energy efficiency.

5.4 Security-Energy Tradeoffs

Tables I and III demonstrate how TKEP can make a tradeoff between security and energy consumption of cryptographic operations. Figure 7 plots the probability, P_{TKEP_μ}, of eavesdropping TKEP as a function of energy consumption in [mJ], while varying the percentage of compromised sensors. The result confirms P_{TKEP_μ} to be inversely proportional to the energy consumption. Thus,

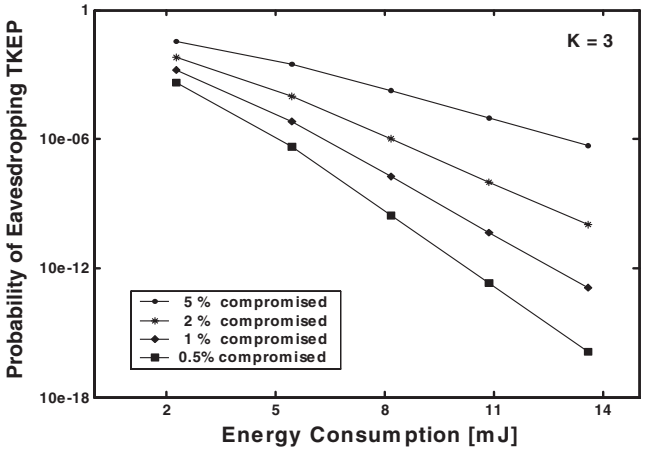

Fig. 7. P_{TKEP_μ} vs. energy consumption.

TKEP is very "flexible" in that any sensor, either as source or destination, can reconfigure TKEP according to its residual energy.

We also measured the number of transcodings for both SGFP and GFP to evaluate SGFP's capability to withstand compromised sensors. GFP incurred 24.7–27.8 transcodings (hops) per path, while SGFP required about 5.5 transcodings per path. That is, the risk of compromised sensors to SGFP is just about one fifth of the hop-by-hop transcoding scheme.

In terms of the energy consumption in communicating packets, SGFP generated 20–30% more packets than GFP. This increase was caused by making the insecure GFP secure. Without SGFP, one would have to rely on the hop-by-hop transcoding (via the key predeployment schemes) or the on-demand key establishment (via the public-key algorithms). However, the former suffers higher risks to physical attacks as well as larger processing and key storage overheads, while the latter incurs extremely large processing overheads. This shows SGFP successfully achieved the two conflicting goals of both high-level security and low-energy consumption, by effectively trading away the communication overhead for processing and the key storage overhead.

6. CONCLUSION

In this paper, we proposed two protocols for secure routing—a secure geographic forwarding protocol (SGFP) and a temporal-key establishment protocol (TKEP)—as cost-effective security solutions for large-scale sensor networks. The distributed key sharing played a crucial role in our proposed protocols: by having a sensor share keys only with a small number of other sensors chosen based on their geographic location and communication direction, we successfully (1) constructed a light-weight, secure network layer and (2) replaced the resource-expensive DH key-setup protocol with a purely symmetric cipher-based (hence, energy efficient) alternative.

Our security analysis and performance evaluation have shown that the distributed key sharing is practically useful and effective in defeating and/or

tolerating many critical attacks, such as sybil, physical, man-in-the-middle, and collusion attacks, while incurring (consuming) only a small amount of overhead (energy) in the packet forwarding and key setup. This, in turn, enabled the realization of light-weight, secure routing protocols at the expense of initial setup overhead in constructing DKS relationships.

APPENDIX

We derive the probability that the adversary eavesdrops SGFP and TKEP when a portion of the network had been compromised, and the expected number of transcodings for both SGFP and TKEP.

A.1 Preliminaries

We make the following assumptions: (1) the adversary eavesdrops communications for a sufficiently large period of time; (2) sensors are uniformly distributed in the entire network; (3) each sensor has up to level-K DKSs; (4) a level-0 square contains, on average, N_0 sensors in a $\lambda \times \lambda$ square area; and (5) the network covers a square area of $(3^K\lambda) \times (3^K\lambda)$. Then, the expected total number of sensors in the network, N_{net}, is $9^K N_0$. The adversary randomly selects and manipulates sensors to acquire all MKs stored in the compromised sensors. Note that compromising the currently active DKSs does not increase the eavesdropping probabilities as compared to the random selection, because a compromised sensor can serve as the DKS only for a very small fraction of the observation time interval.

Let s and d denote source and destination sensors establishing an SGFP path between them. The number of compromised sensors is denoted by N_c. We define a set of all compromised sensors as $\mathcal{C} = \{c_i, i = 1, \ldots, N_c\}$. The set of all uncompromised sensors is then $\mathcal{U} = \mathcal{C}^c$. The probability that a randomly selected sensor f has already been compromised is defined as

$$p_c = Pr\{f \in \mathcal{C}\} = \frac{N_c}{9^K N_0} \tag{1}$$

We define the level-k cumulative area of s as:

$$A_k(s) = \begin{cases} L_0(s) & k = 0 \\ \sum_{m=1}^{8} L_{k,m}(s) & 1 \le k \le K \end{cases} \tag{2}$$

Then, the probability that d lies within $A_k(s)$ is given by:

$$p_{A_k} = Pr\{d \in A_k(s)\} = \begin{cases} \frac{1}{9^K} & k = 0 \\ \frac{8 \cdot 9^{k-1}}{9^K} & 1 \le k \le K \end{cases} \tag{3}$$

We also define the conditional probability associated with $A_k(s)$ as:

$$p_{L_{k,m}|A_k} = Pr\{\, d \in L_{k,m}(s) \,|\, d \in A_k(s) \,\} = \frac{1}{8} \tag{4}$$

where $1 \le k \le K$ and $1 \le m \le 8$.

A.2 Eavesdropping Probabilities

The sample space is a set $\Omega = \{(s, d),\ s \neq d,\ d \in \sum_{k=0}^{K} A_k(s)\}$, and the event space $\mathcal{E}$ consists of (s, d) pairs for which the adversary successfully decrypts an SGFP packet. Let P_{SGFP} $(= Pr\{\mathcal{E}\})$ and P_{TKEP_μ} denote the probabilities of eavesdropping SGFP and μ-split TKEP, respectively. We then define the following conditional probabilities associated with $\mathcal{E}$:

$$P_k = Pr\{\mathcal{E} \mid s \in \mathcal{U},\ d \in A_k(s)\},\ 0 \leq k \leq K \tag{5}$$

and

$$Q_{k,m} = Pr\{\mathcal{E} \mid s \in \mathcal{U},\ d \in L_{k,m}(s)\} \tag{6}$$

where $1 \leq k \leq K$, $1 \leq m \leq 8$.

We derive P_{SGFP} and P_{TKEP_μ} by considering the following two cases. First, when $s \in \mathcal{U}$ and d is located inside $A_0(s)$, the adversary can decrypt the packet if d and/or intermediate sensors have been compromised. Hence, $P_0 = \alpha \cdot p_c$ where α is the average number of hops taken inside the level-0 square. Second, when $s \in \mathcal{U}$ and d lies within $A_k(s)$, $k \geq 1$, P_k is derived by considering $d \in L_{k,m}(s)$, $m = 1, \ldots, 8$, for each of which the adversary decrypts the SGFP packet with the success probability $Q_{k,m}$. Therefore, the following relationship holds:

$$P_k = \sum_{m=1}^{8} p_{L_{k,m}|A_k} \cdot Q_{k,m} \tag{7}$$

If $d \in L_{k,m}(s)$, s asks $f_{k,m} = \mathcal{DKS}_{k,m}(s)$ to search, on behalf of itself, a reduced area $A_{k-1}(f_{k,m})$ for d. In this case, the adversary succeeds in eavesdropping if $f_{k,m} \in \mathcal{C}$ or $f_{k,m} \in \mathcal{U}$, but the subsequent forwarding inside $A_{k-1}(f_k)$ is routed via compromised DKSs. Therefore,

$$Q_{k,m} = p_c \cdot 1 + (1 - p_c) \cdot P_{k-1} \tag{8}$$

Consequently,

$$P_k = \begin{cases} \alpha p_c & k = 0 \\ p_c + (1 - p_c) P_{k-1} & 1 \leq k \leq K \end{cases} \tag{9}$$

From Eqs. (3) and (9), P_{SGFP} is derived as:

$$P_{SGFP} = \sum_{k=0}^{K} p_{A_k} P_k \tag{10}$$

P_{TKEP_1} of the basic TKEP (with no randomization) is then derived from the event that the attacker eavesdrops both R_s and R_d as:

$$P_{TKEP_1} = P_{SGFP}^2 \tag{11}$$

Finally, each SGFP path of the μ-split TKEP is bounded by $[\, p_c + (1 - p_c) P_K \,]^\mu$ and, hence,

$$P_{TKEP_\mu} \leq [\, p_c + (1 - p_c) P_K \,]^{2\mu} \tag{12}$$

A.3 Expected Number of Transcodings

Let T_{SGFP} and T_{TKEP_μ} denote random variables that count the transcodings by SGFP and TKEP, respectively, and p_n the ratio of the number of non-DKS sensors to the total number of sensors. We first derive the conditional expectation, $E_{A_k}[T_{SGFP}] = E[T_{SGFP}|d \in A_k(s)]$, as:

$$E_{A_k}[T_{SGFP}] = k + \alpha + p_n, \quad 0 \le k \le K \tag{13}$$

$E[T_{SGFP}]$ is then derived as:

$$E[T_{SGFP}] = \sum_{k=0}^{K} p_{A_k} E_{A_k}[T_{SGFP}] \tag{14}$$

$$= K + \alpha + p_n - \frac{1}{8}\left(1 - \frac{1}{9^K}\right) \tag{15}$$

Finally, $E[T_{TKEP_\mu}]$ is approximated as:

$$E[T_{TKEP_\mu}] \simeq \begin{cases} 2\left(K + \alpha + p_n - \frac{1}{8}\right) & \mu = 1 \\ 2\mu\left(K + \alpha + p_n + \frac{7}{8}\right) & \mu \ge 1 \end{cases} \tag{16}$$

REFERENCES

ASOKAN, N. AND GINZBOORG, P. 2000. Key agreement in ad hoc networks. *Computer Communications*, 1627–1637.

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

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.

CARMAN, D. W., MATT, B. J., AND CIRINCIONE, G. H. 2002. Energy-efficient and low-latency key management for sensor networks. In *Proceedings of 23rd Army Science Conference*.

CERPA, A., ELSON, J., ESTRIN, D., GIROD, L., HAMILTON, M., AND ZHAO, J. 2001. Habitat monitoring: application driver for wireless communications technology. In *Proceedings of ACM Workshop on Data Communications in Latin America and Caribbean*.

CHAN, H., PERRIG, A., AND SONG, D. 2003. Random key predistribution schemes for sensor networks. In *Proceedings of IEEE Symposium on Security and Privacy '03*.

CHEN, B., JAMIESON, K., BALAKRISHNAN, H., AND MORRIS, R. 2001. SPAN: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. In *Proceedings of ACM/IEEE MobiCom '01*. Rome, Italy.

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

DOUCEUR, J. 2002. The Sybil attack. In *Proceedings of 1st International Workshop on Peer-to-Peer Systems*.

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 GLIGOR, V. D. 2002. A key-management scheme for distributed sensor networks. In *Proceedings of ACM CCS '02*. Washington, DC.

HE, T., HUANG, C., BLUM, B. M., STANKOVIC, J. A., AND ABDELZAHER, T. 2003. Range-free localization schemes for large scale sensor networks. In *Proceedings of ACM/IEEE MobiCom '03*.

HEIDEMANN, J., SILVA, F., INTANAGONWIWAT, C., GOVINDAN, R., ESTRIN, D., AND GANESAN, D. 2001. Building efficient wireless sensor networks with low-level naming. In *Proceedings of ACM SOSP '01*.

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

HU, L. AND EVANS, D. 2004. Localization for mobile sensor networks. In *Proceedings of ACM/IEEE MobiCom '04*.

JAIN, R., PURI, A., AND SENGUPTA, R. 2001. Geographical routing using partial information for wireless ad hoc networks. *IEEE Personal Communications*.

KARLOF, C. AND WAGNER, D. 2003. Secure routing in wireless sensor networks: attacks and countermeasures. *Ad Hoc Networks*.

KARLOF, C., SASTRY, N., AND WAGNER, D. 2004. TinySec: A link layer security architecture for wireless sensor networks. In *Proceedings of ACM SenSys '04*.

KARP, B. AND KUNG, H. T. 2000. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proceedings of ACM/IEEE MobiCom '00*. Boston, MA.

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 ACM/IEEE MobiCom '00*. Boston, MA.

LIU, D. AND NING, P. 2003a. Establishing pairwise keys in distributed sensor networks. In *Proceedings of ACM CCS '03*.

LIU, D. AND NING, P. 2003b. Location-based pairwise key establishment for static sensor networks. In *Proceedings of ACM SASN '03*.

LIU, D., NING, P., AND DU, W. 2005. Group-based key pre-distribution in wireless sensor networks. In *Proceedings of ACM WiSe '05*.

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

MALAN, D. 2004. Crypto for tiny objects. Technical Reprot TR-04-04, Harvard University.

MILLER, M. J. AND VAIDYA, N. H. 2005. A MAC protocol to reduce sensor network energy consumption using a wakeup radio. *IEEE Transactions on Mobile Computing 4,* 3 (May/June).

PARK, T. LiSP: Lightweight security protocols for wireless sensor networks. Ph.D. thesis, EECS Department, The University of Michigan, Ann Arbor, MI.

PARK, T. AND SHIN, K. G. 2004. LiSP: A lightweight security protocol for wireless sensor networks. *ACM Transactions on Embedded Computing Systems 3,* 3 (Aug.).

PARK, T. AND SHIN, K. G. 2005a. Optimal tradeoffs for location-based routing in large-scale ad hoc networks. *IEEE/ACM Transactions on Networking 13,* 2 (Apr.).

PARK, T. AND SHIN, K. G. 2005b. Soft tamper-proofing via program integrity verification in wireless sensor networks. *IEEE Transactions on Mobile Computing 4,* 3 (May/June).

PRIYANTHA, N. B., BALAKRISHNAN, H., DEMAINE, E. D., AND TELLER, S. 2005. Mobile-assisted localization in wireless sensor networks. In *Proceedings of IEEE INFOCOM '05*. Miami, FL.

RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. 2001. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM '01*. San Diego, CA.

RATNASAMY, S., KARP, B., SHENKER, S., ESTRIN, D., GOVINDAN, R., YIN, L., AND YU, F. 2003. Data-centric storage in sensornets with ght, a geographic hash table. *Mobile Networks and Applications (MONET) Special Issue on Algorithmic Solutions for Wireless, Mobile, Ad Hoc and Sensor Networks*.

ROWSTRON, A. AND DRUSCHEL, P. 2001. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms*.

STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, F., AND BALAKRISHNAN, H. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM '01*. San Diego, CA.

VIDAL, R., SHAKERNIA, O., KIM, H. J., SHIM, H., AND SASTRY, S. 2002. Probabilistic pursuit-evasion games: Theory, implementation and experimental evaluation. *IEEE Transactions on Robotics and Automation 18,* 5 (Oct.).

WANG, H., ESTRIN, D., AND GIROD, L. 2003. Preprocessing in a tiered sensor network for habitat monitoring. *EURASIP JASP Special Issue of Sensor Networks*.

WATRO, R., KONG, D., CUTI, S., GARDINER, C., LYNN, C., AND KRUUS, P. 2004. TinyPK: Securing sensor networks with public key technology. In *Proceedings of ACM SASN '04*. 59–64.

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

XUE, Y., LI, B., AND NAHRSTEDT, K. 2001. A scalable location management scheme in mobile ad hoc networks. In *Proceedings of IEEE Conf. on Local Computer Networks (LCN)*.

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 ACM/IEEE MobiCom '02*. Atlanta, GA.
ZHU, S., SETIA, S., AND JAJODIA, S. 2003. LEAP: Efficient security mechanisms for large-scale distributed sensor networks. In *Proceedings of ACM CCS '03*.

Received July 2005; revised February 2006; accepted November 2006