On Building a Lightweight Security Architecture for Sensor Networks

Taejoon Park[1] and Kang G. Shin[2]

[1] Samsung Advanced Institute of Technology
P.O. Box 111, Suwon, Korea
joy.park@samsung.com
[2] Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, MI 48109-2121, USA
kgshin@eecs.umich.edu

Abstract. Sensor networks are characterized by their large-scale and unattended deployment that invites numerous critical attacks, thereby necessitating high-level security support for their intended applications and services. However, making sensor networks secure is challenging due mainly to the fact that sensors are battery-powered and it is usually very difficult to change or recharge their batteries. In this paper, we give a comprehensive overview of recent research results for securing such sensor networks, and then describe how to build a security framework, called a *Lightweight Security Architecture* (LiSA), for sensor networks, which achieves *energy-aware security* via closely-coupled, mutually-complementary security solutions.

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, storage, computation, and communication capacities. These sensor networks can be used for various 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. The success of these applications hinges on their own *security*; they must protect themselves by preventing and/or tolerating critical attacks from malicious adversaries. However, despite its importance, it is challenging to achieve high-level security throughout the lifetime of sensor networks due mainly to the operational issues and requirements unique to sensor networks, such as energy-efficiency in terms of prolonging the lifetime of sensor devices as much as possible, 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 comes with 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 the most important requirements.

We, therefore, need a security architecture tailored to sensor networks that meets the requirements of both high-level security and energy-efficiency. To this end, we propose how to build an *energy-aware security* framework, called a ***Li**ghtweight **S**ecurity **A**rchitecture* (LiSA), for a network of resource-limited sensors. The two main contributions of this paper are:

- the comprehensive overview of recent research results for securing sensor networks as well as plausible security attacks on such sensor networks, and
- the design and development of LiSA via cooperative interactions among closely-coupled, mutually-complementary security solutions that consist of (1) a soft tamper-proofing technique that verifies integrity of the program residing in each sensor device whenever it joins the network, or is suspected to have been compromised; (2) two key management/sharing schemes, each tailored to local and remote transactions; and (3) an attack-tolerant localization protocol, playing the role of an anomaly-based intrusion detection system tailored to localization.

The remainder of this paper is organized as follows. Section 2 presents an overview of sensor networks. Section 3 summarizes security attacks on sensor networks, while Section 4 describes existing security protocols for sensor networks. Section 5 details the LiSA architecture and its building blocks. Finally, the paper concludes with Section 6.

2 Overview of Sensor Networks

2.1 Device/Network Architecture

For cost and size reasons, sensor devices are designed to minimize resource requirements, e.g., Motes [1] 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 limited processing, communication and memory capabilities in order to prolong their lifetime with the limited energy budget.

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 (due to device failures or depletion of battery energy) and addition of sensor nodes are not uncommon. Sensors collaborate and coordinate with one another to achieve a higher-level sensing task, e.g., measuring and reporting, with accuracy, the characteristics of a moving object, such as the speed and direction of its movement.

2.2 Communication Models

The main challenge associated with a sensor network is the large volume of data to be collected and processed over the entire network. The communication models for sensor networks to address this challenge are *cluster-based* or *peer-to-peer*. The cluster-based model typically appears in a tiered architecture, where multiple clusters are formed statically and/or dynamically, and a cluster-head manages and controls operations inside each cluster, i.e., by aggregating sensed data within their own cluster as well as disseminating the data among themselves.

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 authors of [2] proposed data to be named and communication abstractions to refer to these names rather than sensor IDs. In a data-centric storage [3], 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 and stores data at the sensor geographically closest to the hashed coordinates. This model calls for transactions between remote nodes because data-sinks can be far away from the data-source. The need for long-distance communications will continue to increase as new applications are expected to aggressively exploit the large-scale, distributed nature of sensor networks.

2.3 Localization Algorithms

There exist many applications that require each sensor node to be location-aware; for example, each sensor must be uniquely identified by its location estimate for geographic routing [4] in which a source or an intermediate sensor forwards a packet to one of its neighbors closest to the packet's destination. As such, *localization* — assigning locations to sensors consistently with measured or estimated distances — is one of the core services of sensor networks. Techniques for estimating the distance between a pair of communicating nodes are typically based on: received signal strength that can be translated into a distance estimate, time of arrival and time difference of arrival that use the signal propagation time, and angle of arrival that estimates the relative angle between nodes.

Besides these ranging techniques, range-free schemes have also been proposed to provide cost-effective, coarse-grained localization. For example, in [5], each sensor forms virtual triangular regions among the anchors of interest, determines in which regions it resides based on its neighbors' measurements, and finally, calculates the overlapping area between these regions. These schemes, however, typically require very high anchor density and long anchors' radio ranges as each sensor has to hear from as many anchors as possible. Hence, it is preferable to provide the localization capability even when there are only a very small number of anchors in the network. In [6], each sensor determines the minimum hop-counts to anchors by running a distance vector algorithm, and computes physical distances by multiplying them to the average per-hop distance. This scheme, unfortunately, yields poor localization accuracy. Approaches like [7] employ mobile anchors to meet the requirements of both low anchor density and high accuracy of distance estimation, but suffers a large latency.

3 Security Attacks

Sensor networks are vulnerable to various security attacks, especially because they are deployed in an unattended, hostile environment. Possible types of adversaries can be classified, in the order of increasing strength, as: (1) passive attackers only eavesdropping conversations; (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, reprogramming and/or cloning sensors; (2) *service disruption attacks* on routing, localization and clock synchronization; (3) *data attacks*, e.g., traffic capture, replaying and spoofing; (4) *resource-consumption and denial-of-service (DoS) attacks* that diminish or exhaust the sensors' capacity/energy to perform its normal function, e.g., by jamming the local area, inducing collisions, and forcing to repeat the same packet transmission; and (5) *sybil attacks* by which a single malicious sensor device claims/presents multiple sensor IDs (locations) to control a substantial fraction of the ID space which, in turn, makes it easier to mount other attacks.

The adversary may also disrupt the integrity/availability of localization service. Possible attacks on the localization service include: (i) sensor displacement or removal; (ii) distance enlargement/reduction via adjustment of transmission power, or placement of obstacles interfering with direct paths; (iii) announcement of false locations, distances or hop-counts; (iv) message modification or replaying; (v) wormhole attacks that create hidden links between remote sensors to be used for replaying messages or altering distance measurements or hop-counts; and (vi) deployment of bogus anchors that propagate false reference location information. These localization-specific attacks try to propagate wrong information about locations of, or distances to, the sensors (or anchors) under the adversary's control in an attempt to disrupt the localization service.

One of the serious attacks to the sensor networks deployed in an unattended environment is physical tampering with sensors. An adversary can easily capture one or more sensors, scrutinize/reverse-engineer/alter the program and/or master-secret in the sensor, and create/deploy (multiple clones of) manipulated sensors. A small number of sensors compromised by physical attacks may serve as malicious slaves for many serious attacks, such as initiating DoS or sybil attacks and sabotaging certain services of the sensor network, which will, in turn, facilitate the subversion of the entire network.

4 Security Protocols

4.1 Cryptography for Sensor Networks

Public-key algorithms have been widely used for the development of various key establishment protocols that derive a common key among nodes. However, they are unsuitable for sensor networks, because of their large energy demands,

let alone the requirement of exchanging public-key certificates. In particular, existing implementations of the Diffie-Hellman (DH) protocol on sensor devices [8] consume 1.19 $\sim$ 12.64 [J] per operation, which is too much to be usable in devices with a limited energy budget, e.g., 61,560 [J] in Motes.

A sensor device, therefore, cannot use public-key algorithms due mainly to its severe resource constraints. The symmetric-key ciphers and cryptographic hash functions, which are orders-of-magnitude cheaper and faster (e.g., 0.115 [mJ] in TinySec), 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, the authors of [9] and [10] realize the stream cipher by running the RC5 block cipher in the counter mode and in the output feedback mode, respectively.

4.2 Key Management and Sharing

The cluster-based key management [11, 12] is concerned with 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; for example, each cluster-head (even though better-equipped and better-protected than normal sensor nodes) is a single point of failure, implying that if compromised, it may break the cluster's security. Moreover, an efficient mechanism for securing inter-cluster communications must be provided to deal with transactions between remote nodes residing in different clusters.

Key pre-deployment schemes [13, 14] statically set up pairwise shared keys based on keys loaded into sensor devices prior to their deployment. That is, each sensor is preloaded with multiple (a couple of hundreds) keys randomly chosen from a large pool of keys, and hence, a pairwise key is established between a pair of neighboring sensors if a key happens to be common to both sensors. However, the pairwise keying performs poorly for communications over multiple-hop paths, since it requires transcoding (decryption followed by re-encryption) for each and every hop, thereby risking the security as any malicious sensor node on the path may take control of the communication as well as increasing sensors' workloads and the packet-delivery latency.

4.3 Countermeasures against Attacks on Localization

Determining sensors' locations in an untrusted environment is a challenging, but important, problem that has not yet been fully studied. Like other security applications, one may want to authenticate all the messages to protect the network against attacks. However, it suffers high computational overhead and/or a large authentication latency, and, more importantly, many of the localization-targeted attacks are non-cryptographic in nature, making these authentication-based solutions highly unlikely to succeed. In [15], each sensor hears directly from multiple anchors, identifies a region it resides in, and determines its location as the

center of the region. However, its main drawback is the requirement of a large number of specialized anchors equipped with directional/sectored antennae and capable of high power transmission.

Recently, statistical approaches have been proposed [16, 17]. In [16], the authors presented an attack-tolerance mechanism for triangulation-based localization, in which each sensor applies the least median squares algorithm on the distance estimates to anchors in order to mitigate the effect of attacks. The authors of [17] also use a collection of anchors' reference locations associated with estimated distances, and apply the mean square error criterion to identify and discard malicious location references. Unfortunately, these methods invite attacks from relaying sensors and require a significant amount of redundant location/distance information from anchors, incurring a high network overhead to achieve a reasonable degree of robustness against attacks. These drawbacks mainly come from the fact that they do not fully extract/utilize the available information and ignore the relationship/correlation among sensors' locations.

4.4 Tamper-Proofing Techniques

Code obfuscation [18] converts the executable code into an unintelligible form that makes analysis/modification difficult. However, the level of difficulty to tamper with gets substantially lower as the program becomes smaller, and hence, it cannot protect against determined attackers. Furthermore, as shown in [19], obfuscating programs while preserving its functionality is theoretically impossible. Result checking [20] examines the validity of intermediate results produced by the program, but it is inappropriate for use in battery-powered devices because it continuously incurs verification overhead. Aucsmith [21] proposed to store the encrypted executable and decrypt it before execution. However, this scheme suffers from a very high decryption/re-encryption overhead, and the security of self-decrypting programs can be easily broken unless the decryption routines are protected from reverse-engineering. Self-checking techniques [22, 23] use embedded codes to compute a hash value on the program and compare it with the correct value (also embedded in the program). However, similarly to the self-decryption techniques, they become defenseless once the hash computation code and/or the hash value has been identified/analyzed.

Besides protection of software itself, researchers studied techniques based on external servers to examine the program code. In [24], a server sends the checksum code to the remote system, computes a hash, and uses timing to determine the system's genuinity. Its key idea is the randomized memory access that triggers more page faults and cache misses on a virtual memory system, leading to a severe slowdown in hash computation. However, this scheme is not suitable for sensor devices that do not have virtual memory support. The authors of [25] proposed a technique that also uses randomized memory traversal to force an attacker to check if the current memory access is made to a modified location, causing a detectable increase in the hash calculation time. However, this scheme is inefficient as it incurs more memory accesses than sequential scanning of the program, without guaranteeing 100 % detection of memory modifications.

5 Lightweight Security Architecture

In what follows, we describe how to address the challenges of both security and energy-efficiency in the design of LiSA.

5.1 Classification of Attacks

The threats in Section 3 is rehashed into attacks on the program code of sensors (Class-1) vs. attacks on the data traffic (Class-2). The former relates to the adversary's attempt to physically compromising sensor devices, while the latter ranges from passive eavesdropping to traffic replaying/modification/injection leading to, e.g., service disruption or DoS attacks.

5.2 The Proposed Approach

We propose an approach to building LiSA as a set of closely-coupled security protocols tailored to each type of attacks to meet the following design objectives: the protocols must be

- **lightweight** so as to prolong the network lifetime significantly, which requires the use of computationally-efficient ciphers such as symmetric-key algorithms and cryptographic hash functions;
- **cooperative** in the sense of achieving high-level security via mutual collaboration/cooperation among sensor nodes as well as with other protocols;
- **attack-tolerant** to enable the network to gracefully tolerate attacks and device compromises as well as heal itself by detecting, identifying, and removing the sources of attacks;
- **flexible** enough to trade security for energy consumption;
- **compatible** with existing security mechanisms and services; and
- **scalable** to the rapidly growing network size.

5.3 The Proposed Architecture

Fig. 1 shows a complete LiSA framework consisting of the following components. First, *program-integrity verification* serves as a strong access control mechanism against compromised sensors, under which a sensor joining the network or suspected of having been compromised must register itself to the server after verification of its program. Second, *key management/sharing* deals with efficient distribution, sharing and renewal of keys, and consists of two mutually complementary schemes: the cluster-based scheme is tailored to localized, cluster-based communications, while the distributed key sharing achieves extremely lightweight protection for communications between distant sensor nodes. Note the former defeats attacks of Class-1, while the latter targets the Class-2 attacks. Third, *attack-tolerant services* make it possible for the sensors' core services, such as routing, localization, and clock synchronization, to gracefully tolerate attacks from compromised/malicious nodes. Finally, *intrusion detection*, e.g., running on each cluster-head, monitors or probes network activities to uncover misbehaving sensors followed by the activation of either of the first two services.

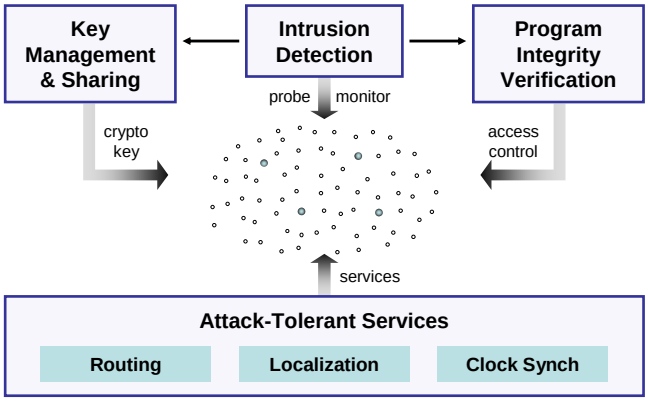

Fig. 1. The Lightweight Security Architecture

5.4 Secure Network Layer

As shown in Fig. 2, the building blocks of LiSA cooperate with each other to form a secure network layer that provides security services to the application layer. Below we detail the roles of these building blocks as well as their interactions.

Program-integrity verification: achieves purely software-based prevention of sensors from physical attacks using, for example, a randomized hash function and mobile agent technology presented in [26]. This technique examines integrity of the program residing in each sensor device whenever it joins the network or is found (by the intrusion detector) to have been compromised. Based on the cluster-based model, it uses two types of dedicated devices: the *cluster-head* executing the verification protocol on a sensor, and the *authentication-server* acting as a trusted third party for the sensor in testing the cluster-head.

Cluster-based key management: is designed specifically for the cluster-based model that rely heavily on local transactions inside the cluster, and makes a tradeoff between security and resource consumption via highly efficient re-keying based on the cryptographic one-way function and double-buffering of keys, as detailed in [27]. Applications like data aggregation/dissemination can use this scheme to secure the intra- and inter-cluster communications.

Distributed key sharing: is tailored to securing communications between remote sensors. Based on the peer-to-peer model, it enables each sensor to share unique pairwise-keys with a small number of geographically-chosen sensors, which leads to the development of two attack-tolerant routing protocols: *secure geographic forwarding* that delivers packets by concatenating the pairwise-key paths, each secured with its own key and forwarded geographically; and *session-key setup* that creates a secure session between two sensors by applying the secure geographic forwarding twice. While the former is invoked for per-packet protection, the latter is activated to establish

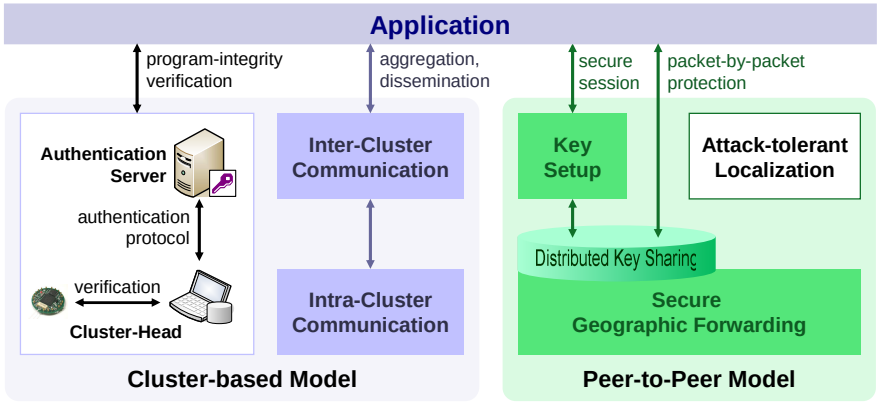

Fig. 2. Secure Network Layer of LiSA

the secure session. These protocols gracefully resist device compromises as well as replace public-key-cipher-based protocols with a purely symmetric-cipher-based alternative.

Attack-tolerant localization: autonomously determines sensors' relative locations by using mutual collaboration among sensors to achieve high-level attack-tolerance in terms of detecting/identifying/rejecting sources of attacks, if present. By exploiting the high spatio-temporal correlation existing among adjacent nodes, it realizes adaptive management of a profile for normal localization behavior and distributed detection of false locations advertised by attackers, and hence, plays the role of an anomaly-based intrusion detection system tailored to localization that safeguards the network from localization-targeted attacks. Note that the same methodology can be applied to the development of attack-tolerant clock synchronization protocol.

6 Conclusion and Future Work

In this paper, we addressed the problem of energy-efficient security technology for resource-limited embedded sensor devices, and developed LiSA that enables low-power sensors to provide high-level security at a very low cost. We avoided using the traditional cryptography-based approaches intended for environments equipped with sufficient resources, and instead, focused on building security protocols via collaboration/cooperation among sensor nodes as well as among the protocols themselves. To broaden the applicability of LiSA, it is required to solve open research problems associated with the attack-tolerant protocol design and the generalization of tamper-proofing technology.

References

1. Crossbow. MICA, MICA2 Motes & Sensors. Available: http://www.xbow.com/.

2. J. Heidemann et al. Building Efficient Wireless Sensor Networks with Low-Level Naming. In *Proceedings of SOSP '01*, October 2001.
3. S. Ratnasamy et al. Data-Centric Storage in Sensornets with GHT. *MONET: Algorithmic Solutions for Wireless, Mobile, Ad Hoc and Sensor Networks*, 2003.
4. B. Karp and H. T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *Proceedings of MobiCom '00*, August 2000.
5. T. He et al. Range-Free Localization Schemes for Large Scale Sensor Networks. In *Proceedings of MobiCom '03*, September 2003.
6. D. Nicolescu and B. Nath. Ad-Hoc Positioning Systems (APS). In *Proceedings of IEEE GLOBECOM '01*, November 2001.
7. L. Hu and D. Evans. Localization for Mobile Sensor Networks. In *Proceedings of MobiCom '04*, October 2004.
8. D. Malan. Crypto for Tiny Objects, 2004. Harvard Univ. Tech. Rep. TR-04-04.
9. A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security Protocol for Sensor Networks. In *Proceedings of MobiCom '01*, July 2001.
10. M. Burnside, D. Clarke, T. Mills, S. Devadas, and R. Rivest. Proxy-based Security Protocols in Networked Mobile Devices. In *Proceedings of SAC '02*, March 2002.
11. S. Basagni, K. Herrin, D. Bruschi, and E. Rosti. Secure Pebblenets. In *Proceedings of MobiHoc '01*, October 2001.
12. D. W. Carman, P. S. Kruus, and B. J. Matt. Constraints and Approaches for Distributed Sensor Network Security, September 2000. NAI Tech. Rep. #00-010.
13. L. Eschenauer and V. D. Gligor. A Key-Management Scheme for Distributed Sensor Networks. In *Proceedings of ACM CCS '02*, November 2002.
14. H. Chan, A. Perrig, and D. Song. Random Key Predistribution Schemes for Sensor Networks. In *Proceedings of IEEE Security and Privacy '03*, May 2003.
15. L. Lazos and R. Poovendran. SeRLoc: Secure Range-Independent Localization for Wireless Sensor Networks. In *Proceedings of ACM WiSe '04*, October 2004.
16. Z. Li, W. Trappe, Y. Zhang, and B. Nath. Robust Statistical Methods for Securing Wireless Localization in Sensor Networks. In *Proceedings of IPSN '05*, April 2005.
17. D. Liu, P. Ning, and W. Du. Attack-Resistant Location Estimation in Sensor Networks. In *Proceedings of IPSN '05*, April 2005.
18. G. Wroblewski. General Method of Program Code Obfuscation. In *Proceedings of SERP '02*, June 2002.
19. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (Im)possibility of Obfuscating Programs. *CRYPTO'01*, 2001.
20. F. Ergun, S. Kannan, S. R. Kumar, R. Rubinfeld, and M. Vishwanathan. Spot-Checkers. In *Proceedings of ACM Symp. Theory of Computing*, May 1998.
21. D. Aucsmith. Tamper Resistant Software: An Implementation. *Information Hiding, LNCS 1174*, pages 317–333, 1996.
22. B. Horne, L. Matheson, C. Sheehan, and R. E. Tarjan. Dynamic Self-Checking Techniques for Improved Tamper Resistance. *DRM'01*, pages 141–159, 2002.
23. H. Chang and M. J. Atallah. Protecting Software Code by Guards. *DRM'01*, pages 160–175, 2002.
24. R. Kennell and L. H. Jamieson. Establishing the Genuinity of Remote Computer Systems. In *Proceedings of USENIX Security Symposium*, August 2003.
25. A. Seshadri, A. Perrig, L. Doorn, and P. Khosla. SWATT: SoftWare-based AT-Testation for Embedded Devices. In *Proceedings of IEEE S&P '04*, May 2004.
26. T. Park and K. G. Shin. Soft Tamper-Proofing via Program Integrity Verification in Wireless Sensor Networks. *IEEE Trans. on Mobile Computing*, May/June 2005.
27. T. Park and K. G. Shin. LiSP: A Lightweight Security Protocol for Wireless Sensor Networks. *ACM Trans. on Embedded Computing Systems*, August 2004.