The Pennsylvania State University
The Graduate School

SECURE WIRELESS SENSOR NETWORKS: BUILDING BLOCKS AND APPLICATIONS

A Thesis in
Computer Science and Engineering
by
Hui Song

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 2007

The thesis of Hui Song was reviewed and approved* by the following:

Guohong Cao
Associate Professor of Computer Science and Engineering
Thesis Advisor, Chair of Committee

Thomas F. La Porta
Distinguished Professor of Computer Science and Engineering

Sencun Zhu
Assistant Professor of Computer Science and Engineering
Assistant Professor of Information Sciences and Technology

Carleen Maitland
Assistant Professor of Information Sciences and Technology

Raj Acharya
Professor of Computer Science and Engineering
Department Head of Computer Science and Engineering

*Signatures are on file in the Graduate School.

Abstract

Sensor networks are ideal candidates for a wide range of applications, such as monitoring of critical infrastructures, data acquisition in hazardous environments, and military operations. It is necessary to guarantee the security and resilience of sensor networks (as well as their applications) as they become more and more popular. Despite many security schemes have been proposed to protect building blocks such as routing and key management, some other building blocks (e.g., mobile sink and time synchronization) are largely ignored.

The objective of this thesis is developing security building blocks for sensor networks as well as designing secured sensor network applications. The thesis makes the following three contributions.

The first contribution is the provision of a secured mobile sink building block. In sensor network applications, mobile sinks are often granted with privileges such as accessing and revoking sensors. If they are compromised, the abusing of the privileges may bring down or result in the compromise of the entire sensor network. This thesis describes how to grant mobile sinks with only the minimum required privileges, based on the *principle of least privilege*, and how to quickly revoke their privileges when they are compromised. Simulations and real implementation (using Mica2 motes) have been conducted and shown that the proposed approaches are secure, efficient, and practical.

Secondly, this thesis furnishes a secured time synchronization building block. The existing time synchronization schemes for sensor networks were not designed with security in mind and are vulnerable to many malicious attacks. This thesis is focused on a specific attack using which an attacker can deliberately delay the transmission of time synchronization messages to interfere the time synchronization process between sensors. This attack cannot be addressed by traditional cryptographic techniques. Two approaches have been proposed to detect and accommodate this attack. The first approach uses a statistical method to detect and

remove the outliers (i.e., malicious time synchronization data introduced by the attack), and the second approach uses a time transformation technique to derive the threshold for outlier filtering. Simulations demonstrate that even mild attacks (e.g., introducing only 10 millisecond delay) can be detected effectively (e.g., with 100 percent detection rate and zero percent false positive rate).

The third contribution of this thesis is the design, implementation and evaluation of a **s**ensor-network–based **v**ehicle **a**nti-**t**heft **s**ystem called SVATS. In this system, vehicles are equipped with sensors and sensor networks are automatically formed in parking lots, which actively monitor and identify possible vehicle thefts by detecting unauthorized vehicle movement. When an unauthorized movement is detected, an alert will be reported to a base station in the parking area, which sends warning messages to the security office or car owner. All the messages in the system are secured to mitigate malicious attacks. A prototype based on Mica2 motes is deployed to test the design, which shows that SVATS can detect vehicle theft in four to nine seconds.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I am most grateful and indebted to my thesis advisor, Dr. Guohong Cao, for the large doses of guidance, patience, and encouragement he has shown me during my time here at Penn State. I am also grateful and indebted to Drs. Thomas F. La Porta and Sencun Zhu, for inspiration and enlightening discussions on a wide variety of topics. I am especially indebted for the financial support which they have provided to me over the years. I thank my other committee member, Dr. Carleen Maitland, for her insightful commentary on my work. I also want to thank Yan, my wife, for her support and agape love for taking care of Hannah, our baby girl, during my Ph.D. study. Lastly, not the least, I thank my mother and three sisters for their mental support and encouragement.

Dedication

To my wife and my lovely daughter

Chapter 1

Introduction

Advances in hardware technology and wireless communication have enabled the deployment of large-scale sensor networks [1, 2, 3, 4], where thousands to millions self-powered, small and low-cost sensor nodes are distributed over a vast field to obtain sensing data. These sensor nodes are equipped with sensing, communicating, and data processing units, which allow sensor nodes to collect, exchange, and process information about the environments.

The processing units used in the current generation of sensor nodes, such as Mica/Mica2 motes developed at UC Berkeley [5], are already powerful enough to perform some complicated algorithms to process sensing data, and the units are expected to be more powerful in the future. Due to these attractive characteristics, wireless sensor networks (WSNs) become adopted to many military and civil applications such as battlefield surveillance [6], monitoring critical infrastructure such as the power grid, habitat monitoring [7], environmental control, and security management [4]. This popularity makes it necessary to guarantee the security and resilience of sensor networks as well as their applications.

However, several unique characteristics of sensor networks make it very challenging to build secure and resilient sensor networks. First, sensor nodes shall be produced with low cost, thus typically resource constrained. As a result, it is usually infeasible to use complex, resource-demanded mechanisms such as public key cryptography on such nodes. Second, sensor networks are often deployed in an unattended manner, possibly exposed to physical attacks and be totally compromised. For example, when captured, a sensor node may expose any secret

information (e.g., secret keys) to attackers. Thus, any security mechanism for sensor networks has to be resilient against node compromise attacks.

1.1 The State of the Art in Sensor Network Security

Security in sensor networks has attracted tremendous attention in the past several years. A suite of security schemes have been proposed to address the attacks launched from different layers of the protocol stack. For example, in the application layer, there are security schemes for target tracking [8], software attestation [9, 10, 11], attack-resilient data aggregation [12, 13, 14, 15, 16, 17, 18, 19], secure data dissemination/broadcast authentication [20, 21, 22], secure data collection [23, 24, 25], misbehavior detection [26, 27], and anonymity [28, 29, 30, 31]. In the data link layer, many secure protocols [32, 33] have been proposed to prevent selfish or malicious nodes from abusing the medium access control protocols that are based on the collaboration and trust among neighboring nodes. The severity of channel jamming and defenses has been studied [34]; various attack models for detecting and destroying sensor nodes [35] and the corresponding survival schemes [36] have been proposed as well.

Research on key management has been focused on establishing pairwise keys between two sensor nodes. There are schemes based on the framework of probabilistic key predeployment [37, 38, 39, 40, 41, 42, 43, 44, 45, 46], and recent study on applying public-key cryptography to sensor networks [47, 48, 49]. There are also some work on securing routing protocols in sensor networks [50, 51, 52, 53].

1.2 Problem Statement

Despite that many security schemes have been proposed to protect building blocks such as routing and key management, some other building blocks (e.g., mobile sink and time synchronization) have been mostly overlooked. The objective of this thesis is developing security building blocks and designing new applications, with security in mind, for sensor networks. The following three sections present the

three problems addressed in this thesis, respectively.

1.2.1 Tolerating Mobile Sink Compromises

Mobile sinks are needed in many sensor network applications such as efficient data collection, network maintenance, etc. For such applications, mobile sinks are often granted with privileges such as accessing and revoking sensors. However, if a mobile sink is given too many privileges, it will become very attractive for attack and compromise. Using a compromised mobile sink, an adversary may easily bring down or even take over the sensor network. Thus, security mechanisms that can tolerate mobile sink compromises are essential.

In the literature, the issue of tolerating MS compromise has not been addressed before. Previous research on sensor network security has been focused on broadcast source authentication, key management, and others. These schemes may be employed for securing the communication between regular sensor nodes or between a MS and a regular sensor, but they cannot restrict the privilege of a MS while giving enough privilege for the MS to accomplish its assigned task.

The first idea for tolerating MS compromises is privilege restriction. However, to restrict the privileges of a mobile sink in sensor networks is not easily: the design challenge arises from the fact that sensor nodes are deployed prior to the dispatch of MSs. For some applications, sensor nodes do not know in advance the policy for an individual MS due to the large variety and the on-demand nature of the tasks. One solution could be flooding the entire network to notify all the sensor nodes of the task as well as the privilege of the MS to be dispatched. However, this not only increases the communication overhead, but also requires every sensor node to receive and store information regarding the MS. Moreover, The overhead increases linearly with the number of MSs employed in the system.

Another way to tolerating MS compromises is privilege deprivation. That is, if a mobile sink (MS) is detected compromised when its privileges are still valid, the MS should be revoked as soon as possible. As a simple approach, the base station may send a revocation message to each host node individually. However, this is not feasible since the base station may not know the ids of these nodes. Even the ids are known, the communication overhead increases rapidly as the number

of host nodes increases. Alternatively, the base station may flood the revocation message over the network. This is still not efficient because all nodes are involved in receiving or forwarding the message.

1.2.2 Securing Time Synchronization

Time synchronization is essential for sensor networks and their applications. However, the existing time synchronization schemes in sensor networks were not designed with security in mind, thus leaving them vulnerable to security attacks. For instance, an attacker can replay old synchronization messages, drop, modify, or even forge exchanged timing messages. Since many of these attacks can be addressed by employing appropriate cryptographic techniques, we focus on a specific type of attack called *delay attack* which cannot be addressed by the cryptographic techniques. In the delay attack, a malicious attacker deliberately delays the transmission of time synchronization messages to magnify the offset between the time of a malicious node and the actual time. All network time synchronization methods rely on some sort of message exchanges between nodes and are vulnerable to this attack in one way or another.

1.2.3 Vehicle Theft Detection Using Sensor Networks

Today vehicle theft rate is high, thus tracking/alarming systems are being deployed with an increasingly popularity. While each existing technique has its unique merit, it cannot address the vehicle theft detection problem entirely due to its limitations (such as high initial expense, high false-alarm rate, easily disabled, noisy, etc.).

On the other hand, although sensor networks has been proposed to use in many different kinds of applications, it is still lack of a killer application that can stimulate the large-scale deployment of sensor networks. It is envisioned that, in the near future, sensors will be produced in large quantities at a very low cost. They are also expected to be miniaturized into a cubic millimeter package (e.g., smart dust) to be stealthy in hostile environments. If vehicles are equipped with these cheap, stealthy sensors, we may use the self-formed sensor networks to detect vehicle theft, which could be a complement or even a replacement to the current vehicle theft prevention techniques. However, the success of such a system depends

on whether vehicle thefts can be effectively detected by the sensor network. In addition, this system should be secured and be resilient against attacks (such as destroying the sensors in the vehicle) from thieves.

1.3 Our Contributions

For the mobile sink compromise problem, based on the *principle of least privilege*, we have proposed an efficient scheme to restrict the privilege of a mobile sink without impeding its capability of carrying out any authorized operations for an assigned task. To deprive the privilege assigned to a compromise mobile sink, we have designed several efficient message forwarding schemes which can revoke the mobile sink immediately after its compromise has been detected. Extensive simulations and real implementation show that our schemes are secure and efficient, and are highly practical for sensor networks consisting of the current generation sensor Mica2 motes.

For securing time synchronization, we have proposed two approaches to detect and accommodate the delay attack. The first approach uses the generalized extreme studentized deviate (GESD) algorithm to detect multiple outliers introduced by the compromised nodes, and the second approach uses a threshold derived using a time transformation technique to filter out the outliers. Simulation results show that, when the delay attack time is at least 100 milliseconds, the successful detection rate is 100%, with no false positives, and the time synchronization accuracy can be increased by as much as 16 times.

Lastly, we have designed and implemented a secured sensor-network-based vehicle anti-theft system (SVATS) using state-of-the-art sensor MICA2 motes. In this system, the senors in the vehicles that are parked within the same parking area first form a sensor network, then monitor and identify possible vehicle thefts by detecting unauthorized vehicle movement. When an unauthorized movement is detected, an alert will be reported to a base station in the parking area, which sends warning messages to the security office. This thesis focuses on the technical issues specific to the system such as connectivity management, theft detection, intra-vehicle networking, alert reporting and security management. A prototype based on Mica2 motes is deployed to test the design. Experimental results show

that SVATS can detect theft within four to nine seconds.

1.4 Organization

The thesis is organized as follows. Chapter 2 reviews previous work on different aspects of security issues in wireless sensor networks. Chapter 3 presents our strategies on mitigating the effect of mobile sink compromises. Chapter 4 is focused on *delay attack*, an attack against time synchronization schemes that cannot be addressed by cryptographic techniques. In Chapter 5, we present *SVATS*, a **s**ensor-network–based **v**ehicle **a**nti-**t**heft **s**ystem. We conclude the thesis with final comments and discuss future work in Chapter 6.

Chapter 2

Related Work

Security in sensor networks has attracted a lot of attention in the past several years. In this chapter, we review some of the security issues that are related to our research, including key management, broadcast authentication, secure routing, secure location, and denial-of-services attacks in sensor networks.

2.1 Key Management in Sensor Networks

Key Management is one of the most essential services in sensor networks. Establishing pairwise keys between two regular sensor nodes has been extensively studied recently. There are schemes using a trusted third party (base station) [20], schemes exploiting the initial trustworthiness of newly deployed sensors [54], and schemes based on the framework of probabilistic key predeployment [37, 38, 39, 40, 41, 43, 42, 44, 45, 46].

Eschenauer and Gligor [37] propose probabilistic key pre-distribution scheme. In this scheme a key pool is generated off-line and every sensor picks a random subset of keys from this key pool. Any two nodes in the communication range can talk with each other only if they share a common key. Depending on the size of the key pool and the size of the sensor key set, the scheme can achieve different connectivity and resilience. Chan et al. [38] later suggest to use the same idea, but increase the intersection threshold from one to some $q > 1$. In their scheme any two nodes are allowed to establish communication only if they share at least q keys. This modification enhanced the resilience of the scheme when a small number of

nodes are compromised.

The Blundo scheme [55], which is a threshold-based pairwise key establishment scheme, has been recently extended to enable a sensor network to sustain more node compromises under the same memory constraints [39, 43, 56]. Du et al. [39] propose to combine Blom scheme [57] with random pre-distribution scheme. They call the set of keys that each node can generate *a key space* and suggest to use multiple key spaces. In this scheme first w key spaces are generated and each sensor is loaded with τ key spaces, $2 \leq \tau \leq w$. Liu et al. [40] proposed several applications of the Blundo scheme. They combine the Blundo scheme with random pre-distribution scheme: a pool of polynomials is generated off-line and then each node gets polynomial shares of subsets of polynomials. They also propose a grid-based solution, where each sensor is placed on a grid, such that other sensors located at the same row or the same column on the grid share a polynomial. Zhu et al. [42] combine the techniques of probabilistic key sharing and threshold secret sharing to establish pairwise keys between two nodes. Chan and Perrig [44] propose PIKE, a class of key-establishment protocols that involves using one or more sensor nodes as a trusted intermediary to facilitate key establishment. Wacker et al. [45] uses multiple two-hop key paths to enhance resilience of the random pre-distribution scheme [37] further under a slightly weaker attack model.

2.2 Broadcast Authentication in Sensor Networks

Another fundamental security service is broadcast authentication, which verifies the integrity and the source of broadcast messages to multiple receivers. Perrig *et al.* [20] present μTESLA for base station broadcast authentication, based on one-way key chains and delayed key disclosure. Liu and Ning [40] propose several multilevel μTESLA schemes to further extend the scalability of the original μTESLA. Zhu et al. [54] and Deng *et al.* [58] present several local one-hop or multiple-hop broadcast authentication schemes that are also based on one-way key chains. The latter schemes allow a receiver node to verify a broadcast message immediately but with weaker security than the μTESLA-based schemes.

2.3 Secure Routing in Sensor Networks

Various attacks and countermeasures against sensor routing protocols are studied in [50] and many secure routing protocols have been devised [51, 52, 53].

Karlof and Wagner [50] described several security attacks on routing protocols for sensor networks and propose some generic countermeasures. Parno et al. [51] leverage all three approaches (prevention, detection and recovery, and resilience) to design secure routing protocols, which are highly robust to attack. SIGF [52], makes explicit the tradeoff between security provided and state which must be stored and maintained. By avoiding or limiting shared state, the protocols prevent many common attacks against routing (e.g., black hole and Sybil), and contain others to the local neighborhood. Yin and Madria [53] propose to use the symmetric cryptography to secure messages, and uses a small cache in sensor nodes to record the partial routing path (previous and next nodes) to the destination. Their protocol, with small byte overhead, guarantees that the destination will be able to identify and discard the tampered messages and ensure that the messages received are not tampered.

2.4 Secure Localization in Sensor Networks

A number of solutions have been proposed to secure the localization process: some rely on bidirectional communication between the infrastructure and the node [59, 60, 61] and some on unidirectional (broadcast) navigation signals emitted by the infrastructure [62, 63, 64, 65].

Capkun and Hubaux [59] propose SPINE, a range-based positioning system based on verifiable multilateration, which provides secure computation and verification of location. ROPE [60] is a robust positioning system, which allows sensors to determine their location without any centralized computation. In addition, it provides a location verification mechanism that verifies the location claims of the sensors before data collection. Anjum et al. [61] propose a secure location scheme in which beacon nodes transmit nonces at different power levels. The location of sensors can be estimated securely based on a set of received nonces. Lazos and Poovendran [62] propose SeRLoc, in which each sensor achieve its location

based the location information transmitted by beacon nodes, using sectored antennae. An attacker has to impersonate several beacon nodes to compromise the localization process. Liu et al. [63] present a suite of techniques for detecting and revoke malicious beacon nodes that provide incorrect information to regular sensor nodes. Liu et al. [65] design two range-based robust methods (i.e., attack-resilient minimum mean square estimation and voting-based location estimation) to tolerate malicious attacks against beacon-based location discovery in sensor networks. Li et al. [64] present two robust statistical methods for tolerating attacks: one triangulation-based localization and one fingerprinting-based scheme.

2.5 Denial-of-service Attacks in Sensor Networks

A denial-of-service (DoS) attack against a wireless sensor network can take several different forms. It may simply jamming the communication channel, or exhaust resources (such as battery), or misroute data. Note that attacks against the routing protocols are in fact DoS attacks since they may result in the interruption or even disconnection of the services provided by the sensor networks. Wood and Stankovic [34] identify a number of DoS attacks in sensor networks. To address jamming-based DoS attacks, Wood et al. [66] propose to use the technique of service mapping to locate the jammed area and them report the jammed area to the BS for further investigation or actions. Pirretti et al. [8] investigate a battery-draining DoS attack called *sleep deprivation attack* and propose several schemes to mitigate its effect.

Chapter 3

Tolerating Mobile Sink Compromises in Wireless Sensor Networks

Mobile sinks (MSs) are needed in sensor network applications such as efficient data collection, localized sensor reprogramming, network maintenance, etc. However, if a mobile sink is given too many privileges, its compromise may result in the compromise (or collapse) of the whole sensor network. To attack this problem, we have designed several security restriction schemes which only grant the MSs the least privilege required to accomplish their tasks. Our constructions guarantee that (1) an MS cannot lie or modify its assigned task, and (2) without any knowledge about the task, any sensor node can verify if a claimed task is valid. We have also proposed several on-demand schemes for privilege deprivation, important and necessary when an MS has been compromised or the security policy has been changed. We have implemented the system on TinyOS/Mica2 motes.

3.1 Overview

Mobile sinks (or mobile soldiers, mobile sensors, shown in Fig. 3.1) are one of the essential components for the operation of many sensor network applications. One such application is data collection. Wireless sensor networks [4] allow continuous environment monitoring in hazard or remote areas. The sensed data often need to be sent back to the base station for analyzing. However, when the sensing field is too far away from the base station, transmitting the data over long distance to the

station may increase the delay, add more network traffic, and weaken the security strength (e.g., some intermediate may modify the data passing by). To address these problems, researchers [67, 68] have proposed to temporarily store the data in sensor nodes, and let the base station periodically dispatch mobile sinks (MSs) to collect data. In some strategic scenarios such as battlefield, MSs [69, 70] are even allowed to directly query data from sensor nodes at any time. In addition to be useful for data collection, MSs may also be employed for network management. For example, the base station may send MSs to detect nodes being compromised or to repair failed nodes.

The use of a MS however introduces a new security challenge: The privilege granted to a MS can be abused once it is compromised. This will become a serious problem if a MS is granted many privileges, making it very attractive for attack and compromise. For example, suppose we want to send a MS to collect sensor readings in a specific location during a specific time interval. Without appropriate restrictions, a compromised MS may be able to collect sensor data from the entire network at all time. In another example, suppose we want to dispatch a MS to inspect an abnormal area of a sensor network and the MS is allowed to revoke and isolate a sensor node if the sensor node is identified as compromised. Again, without appropriate restrictions, a compromised MS can easily revoke any nodes of its choice and bring down the entire network by simply sending some revocation messages. The severe consequence of MS compromises can also be foreseen in other applications, which indicates the importance of restricting the privilege of MSs. On the other hand, the convenience of a MS in executing the authorized operations should not be sacrificed by the increased security restriction.

We can limit the privilege of a MS through policy, but the enforcement of the policy cannot rely on the trustworthiness of the MS because the MS may be compromised. Therefore, we have to resort to security mechanisms for policy enforcement. The design challenge arises from the fact that sensor nodes are deployed prior to the dispatch of MSs. For some applications, sensor nodes do not know in advance the policy for an individual MS due to the large variety and the on-demand nature of the tasks. One solution could be flooding the entire network to notify all the sensor nodes of the task as well as the privilege of the MS to be dispatched. However, this not only increases the communication overhead, but

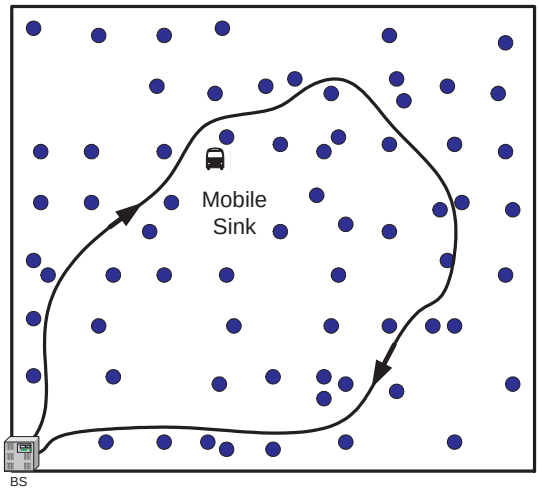


Figure 3.1. An example application where a MS is dispatched to carry out a task along a predetermined trajectory

also requires every sensor node to receive and store information regarding the MS. Moreover, The overhead increases linearly with the number of MSs employed in the system.

To the best of our knowledge, the issue of tolerating MS compromise has not been addressed in the literature. Previous research on sensor network security has been focused on broadcast source authentication [40, 20], key management [38, 39, 37, 43, 54], and others [50, 34, 29]. These schemes may be employed for securing the communication between regular sensor nodes or between a MS and a regular sensor, but they cannot restrict the privilege of a MS while giving enough privilege for the MS to accomplish its assigned task.

Contributions We propose two sets of solutions to address the MS compromise problem. First, based on the *principle of least privilege* [71], we design a security restriction scheme which only grants the MSs the least privilege required to accomplish their tasks. This scheme *allows* and *only allows* a MS to perform the pre-authorized operations. A salient feature of our scheme is that neither do we need to pre-load the sensor nodes with the tasks that might be carried out by MSs, nor do we need to flood the network when dispatching a MS. Indeed, our constructions guarantee that (1) a MS cannot lie or modify its assigned task, and (2) without any knowledge about the task any sensor node can verify if a claimed

task is valid. If the verification fails, a sensor node rejects the request from the MS. We show through detailed analysis and implementation that this scheme is very effective and efficient. In addition, we present an extension to allow conditional trajectory change due to unexpected events.

Second, we propose several on-demand schemes for privilege deprivation: a basic scheme followed by three optimized schemes which can revoke the privilege granted to the MS. Privilege deprivation is important and necessary when a MS has been compromised or the security policy has been changed. Through detailed simulation study, we show that our optimized schemes can greatly reduce the revocation latency as well as the communication overhead compared to the basic scheme.

The remainder of the chapter is organized as follows. We describe our assumptions on node, network, and security as well as our design goal in Section 3.2. Section 3.3 presents our proposed privilege restriction scheme and one extension to it. Section 3.4 presents four privilege revocation schemes. Section 3.5 presents our real implementation of the proposed privilege restriction scheme in Mica2 motes. Finally, Section 3.6 concludes the chapter with a short discussion.

3.2 Network and Security Assumptions

3.2.1 Node and Network Assumptions

We assume regular stationary sensor nodes are constrained in resources. Our proposed schemes target at the current generation of sensor nodes such as Mica/Mica2 motes developed at UC Berkeley [5]. The mote runs the special operating system called TinyOS, which supports the default packet size of 36 bytes out of which 29 bytes are for the actual payload. We assume that every node has space to store several hundred bytes of keying information. A mobile sink (MS) can be as powerful as a laptop-class device or a PDA; however, to make our scheme more general, we also consider resource-constrained mobile devices that have the same amount of resources as the Mica2 Motes, such as CotsBots [72], Micabot [73], Robomote [74], to name a few. A base station (BS) is located in a fixed and secure position.

It is more powerful than MSs and cannot be compromised.

We consider a sensor network divided into grids or cells. Each cell has a unique id and every sensor node knows in which cell it is located [75]. We consider the type of applications in which we dispatch a MS with a known purpose. More specifically, a MS is assumed to perform some tasks along a roughly predetermined physical path (may be an arbitrary curve) in a sensor network. We know the type of a task (e.g., collecting data or network diagnose), the start time and the end time, and the cells involved in the task (the last assumption is relaxed in Section 3.3.4). We assume the clocks of sensor nodes in a network are loosely synchronized. Time synchronization is important for general sensor network applications such as mobile object tracking [76], data aggregation, TDMA radio scheduling, etc. It is also needed in our scheme since a sensor node needs to verify if the task claimed by a MS has the valid time interval.

The required accuracies for both localization and time are application dependent. Currently, using the state of art localization schemes [77, 78, 79], one can achieve meter- or even centimeter-level accuracy; on the other hand, existing time synchronization protocols such as [80, 81, 82] are shown to be able to provide millisecond- or even microsecond-level precision. Generally, the errors introduced by both localization and time synchronization are tolerable in our system.

3.2.2 Security Assumptions

We assume that the base station has a mechanism to authenticate broadcast messages (e.g., based on μTESLA [20]), and every node can verify the broadcast messages. We also assume that when an adversary compromises a node, either a regular node or a MS, it can obtain all the sensitive keying materials possessed by the compromised node. Moreover, the adversary may pool the keying materials from multiple compromised nodes to break the security of the network or to launch advanced attacks. However, we assume that the base station will not be compromised.

Since wireless communication is broadcast-based, we assume that an adversary can eavesdrop on all traffic, inject packets, and replay older packets. We note that an adversary may try to attack the node localization protocol and the time syn-

chronization protocol employed in the sensor network to gain advantages, as a task is location- and time-specific in our proposed schemes. Attacks and countermeasures for node localization schemes and time synchronization have been presented in [83, 62, 65, 84] and in [85, 81, 86, 82], respectively.

The motivation of privilege deprivation (or interchangeably called MS revocation) can be due to the change of security policy, the detection of the MS compromises, or other reasons. In particular, we do not assume that the reason for a MS revocation must be the detection of its misbehavior (e.g, injecting spurious packets). In some applications, MSs are devices carried by other entities (e.g., soldiers, vehicles). Therefore, a MS revocation is often the result of revoking the carrier of the MS. For example, if a mobile soldier is captured by the adversary or is missing in a battlefield, other soldiers can report the event to the network controller, which initiates a MS revocation operation to revoke the MSs carried by this soldier.

3.2.3 Design Goal

Our goal is to design security mechanisms to minimize the potential damages caused by a compromised MS, thus tolerating MS compromises. More specifically, we will propose security mechanisms to meet the following requirements.

- **Least Privilege** A MS should be granted the least privilege required to accomplish its task. Specifically, we should load the MS with the minimal number of keys or security credentials that allow it to communicate with sensor nodes securely.

- **Immediate Privilege Deprivation** Once the compromise of a MS is detected, it should be revoked immediately rather than waiting for the authorized time period to expire.

- **On-demand** We should be able to assign a task on demand as long as the type, the locations and the time interval of a task are valid. The sensor nodes in a sensor network do not need to know a task before deployment or be notified by a network controller on the fly.

- **Efficiency** Since we are targeting at the resource constrained sensor nodes and MSs, the schemes should be efficient in terms of communication, computation and storage.

3.2.4 Notations

The following notations appear in the rest of this discussion.

- u (in lower case) is a regular stationary sensor node.
- MS is a mobile sink.
- TT is the type of a task.
- T_s, T_e are the starting time and the ending time of a task, respectively.
- $MAC(k, s)$ is the message authentication code (MAC) of message s computed with a symmetric key k.

In addition, a sensor node is referred to as a *host node* of a MS if the MS is authorized to talk to it in a task.

3.3 Restricting the Privilege of a Mobile Sink

To restrict the privilege of a MS, we must enable sensor nodes to validate the task claimed by the MS. If the validation fails, sensor nodes will reject any requests from the MS. Thus a MS can only carry out its authorized task. As a MS is attractive for attack and compromise, in addition to adopting a prevention-based privilege restriction approach, we also need a proactive revocation approach to prevent the compromised MS from causing further damages to the host nodes.

We propose two sets of solutions to address the MS compromise problem in the following two sections. This section discusses our proposed schemes for restricting the privilege of a MS. Section 3.4 presents four message forwarding schemes (a basic scheme followed by three optimization schemes) for efficiently and quickly delivering a MS revocation notification to all the host nodes of the compromised MS.

3.3.1 The Strawman Scheme

In the strawman scheme, every node is pre-loaded with an individual key shared with the base station (BS). BS generates a master key K_m, based on which it derives an individual key for every node u as $K_u = G_{K_m}(u)$, where G is a pseudo-random function (PRF) [87]. Now suppose BS knows in advance the ids of all the host nodes for the MS in the task. BS loads the MS with a pairwise key $K_u(MS)$ shared with each host node u.

$$K_u(MS) = H(TT|MS|K_u|T_s|T_e), \tag{3.1}$$

where H is a collision-resistant one-way hash function and $|$ denotes the concatenation of messages.

To establish a pairwise key with node u, the MS needs to send node u its id MS, TT, T_s, and T_e, based on which node u can compute $K_u(MS)$ in the same way. Next node u and MS authenticate to each other by, for example, exchanging the following authentication messages

$$MS \rightarrow u : MS, seq, MAC(K_u(MS), MS|seq) \tag{3.2}$$

$$u \rightarrow MS : u, seq + 1, MAC(K_u(MS), u|seq + 1). \tag{3.3}$$

If node u can successfully verify the message from MS, it will trust the MS and hence assist the MS in the task of type TT during the time interval $[T_s, T_e]$; otherwise, it knows the claimed task is not authorized. In addition, node u saves MS (the id of mobile sink) and seq if the verification for message (3.2) succeeds. Note that due to the one-way property of function H, a compromised MS cannot forge any of the values in (TT, MS, T_s, T_e).

The MS maintains the seq, which is increased by one each time when the MS is communicating with a new node. Here seq is used for preventing replay attacks. Specifically, by saving MS and seq at node u, future authentication messages like (3.2) claimed from the same MS will be dropped automatically. Similarly, replayed message (3.3) will be dropped by the MS automatically too. On the other hand, because of saving MS and seq information at node u, the storage overhead for u could be large if there are a huge number of legitimate MSs in the system. In this

case, a window-based strategy will help make a tradeoff.

The strawman scheme meets our design goal and should work well if a MS only communicates with a small number of host nodes. The scheme however is not scalable in terms of storage if the MS is expected to talk to a large number of host nodes, because the MS needs to store one pairwise key shared with each host node. A more concerned limitation is the requirement of global knowledge on the network topology. Although the BS approximately knows the locations that the MS should visit and it can design an appropriate trajectory for the MS, it might not know the ids of the host nodes (i.e., nodes located on the trajectory). Thus, the BS cannot pre-load the MS with the pairwise keys shared with the en-route host nodes.

Next, we present our proposed scheme, which is built on the strawman scheme and provides scalability and security while restricting the privilege of the MS.

3.3.2 The Proposed Scheme

The limitations of the strawman scheme are due to the lack of scalability of the PRF-based pairwise key pre-deployment scheme. In this subsection, we first present a Blundo scheme-based construction to achieve scalability. Then we use techniques such as Merkle hash tree to address the security concerns remaining in the Blundo scheme-based construction.

3.3.2.1 A Blundo Scheme-based Construction

A scalable scheme for establishing pairwise keys should have the following properties. First, the number of preloaded keys should not increase with the number of host nodes. Second, the scheme should not rely on the pre-knowledge of the ids of the host nodes. Thus we need an on-demand scheme which allows a MS to establish a pairwise key with any sensor node on the fly.

Recently many pairwise key establishing schemes [38, 39, 37, 43, 54] have been proposed. All these schemes enable two nodes to establish a pairwise key on the fly once the two nodes know each other's id, although they provide different security guarantees and incur different performance overheads. In this work, we choose the Blundo scheme [55] to construct our protocols, although several other schemes

[39, 43] might as well be adapted for our purpose. As we shall see shortly that the Blundo scheme provides clear security guarantee. Therefore, the use of the Blundo scheme greatly eases the presentation of our work and enables us to provide a clearer security analysis.

The Blundo scheme, when applied to ad hoc or sensor networks, usually involves the following steps.

- The BS (or a key server) chooses a random symmetric bivariate polynomial $f(x, y)$ of degree t with coefficients over a finite field $GF(q)$, where q is a prime number large enough to accommodate a symmetric key.

$$f(x, y) = \sum_{0 \leq i,j \leq t} a_{ij} x^i y^j, \tag{3.4}$$

  where $a_{ij} = a_{ji}$.

- The BS loads every node n with $f(n, y)$, which is a polynomial obtained by evaluating $f(x, y)$ at $x = n$.

- If two nodes u and v want to set up a pairwise key, each of them evaluates the other's id in its own polynomial. The result $f(u, v) = f(v, u)$ serves as their pairwise key.

Suppose every node in the network has been loaded with its polynomial share before its deployment. Every node can establish a pairwise key with every neighbor node or any other node in the network if needed. After the BS gives a share of the polynomial $f(x, y)$ to the MS (i.e., $f(MS, y)$), the MS can establish pairwise keys with any node in the network on the fly without knowing the id of that node in advance, thus addressing the limitation in the strawman scheme.

The security of the Blundo scheme is determined by t. The scheme provides unconditional secrecy if no more than t nodes collude. If more than t nodes collude by pooling their shares, they can recover the polynomial $f(x, y)$ and hence break the system. Therefore, t should be large enough for the application under consideration. On the other hand, a node stores a bivariate polynomial share represented by $t + 1$ coefficients. The size of a coefficient is the same as that of a symmetric key. For the current generation of sensor nodes with $4K$ RAM, t could be up to several hundreds under the memory constraint [40, 39].

This scheme if applied directly has one drawback. It does not limit the privilege of the MS node. The MS can establish a pairwise key with any node in the network, thus its compromise will lead to a global disaster. To prevent MS from establishing pairwise keys with arbitrarily selected nodes in the network, we propose to embed some information about the host nodes into the id of the MS so that a sensor node can verify if it is a host node for the MS. An example construction is as follows. Suppose node u is a host node for MS. The BS constructs the id of the MS as

$$MS(u) = H(TT|T_s|T_e|u). \tag{3.5}$$

The BS then pre-loads MS with a polynomial share $f(MS(u), y)$. To establish a pairwise key with node u, the MS sends (TT, Ts, Te) to u. Node u can then derive $MS(u)$ in the same way. Next both MS and node u compute their pairwise key $f(MS(u), u)$ $(=f(u, MS(u)))$. Finally, they can authenticate each other by exchanging authentication messages as shown in the strawman scheme.

The scheme however still has several limitations. First, storage is still a concern. If the MS is going to communicate with m nodes, it needs to store $m(t + 1)$ coefficients. Here t could be large because of the security consideration in the Blundo scheme. For example, if $t = 50$, $m = 100$, and the size of a coefficient is 8 bytes, the MS needs to store about 40 KB keying material. This precludes the use of low-end mobile sensors as MSs. Second, more importantly, loading MS with multiple polynomials greatly endangers the polynomial $f(x, y)$ because an attacker can get multiple shares of the polynomial once it compromises the MS. As an extreme example, if m is larger than t, an attacker will be able to reconstruct $f(x, y)$ by solely compromising the MS.

3.3.2.2 Reducing The Number of Polynomial Shares To One

To address the remaining issues of the Blundo-based construction, we reduce the number of polynomial shares possessed by a MS—ideally a MS only possesses *one* polynomial share. In this section we present a construction that achieves this goal. There are two techniques. First, we use the locations of host nodes, instead of their ids, to reduce the information of the host nodes that has to be stored by a MS. Second, we use a Merkle hash tree [88] to construct the id for a MS, so that

only one polynomial share has to be assigned to the MS. Below we describe these two techniques in more detail.

Cell Merging A network field is divided into cells and a cell is referenced by an index (i, j). BS is located at cell $(0, 0)$, as shown in Fig. 3.3. If the MS is scheduled to cross cell (i, j), BS can generate a specific id for MS, $MS(i, j)$.

$$MS(i, j) = H(TT|T_s|T_e|i|j) \tag{3.6}$$

Now the MS can establish a pairwise key $f(MS(i, j), u)$ with any node u in the cell (i, j) using the Blundo scheme-based technique. Due to the one-way property of function H, MS cannot forge any one of the values in (TT, T_s, T_e, i, j).

Using cells instead of host node ids can reduce the number of polynomial shares possessed by a MS if every cell on average includes multiple sensor nodes. The number, however, might still be large because the size of a cell cannot be too large. In an extreme case, if there is only one cell for the entire network, the privilege of the MS will not be restricted because it can establish a pairwise key with any node in the network.

To further reduce the number of polynomial shares for a MS, we propose an encoding algorithm to merge contiguous cells into blocks. We denote the id of each block as a four-variable tuple (i, j, d, s), where (i, j) is the index of the left-bottom cell (called base cell) in a block, d is the locations of other cells relative to the base cell, and s is the number of other cells in the direction d. d has only two values, '0' denoting top and '1' denoting right; therefore, it can be represented by 1 bit. $s = 0$ means that a block only has one cell, the base cell. In Fig. 3.3, the MS traverses 139 cells. Using the block representation, these 139 cells can be merged into 57 blocks. For example, the first block is $(0, 0, 0, 5)$ and the second one along the trajectory is $(1, 5, 1, 3)$.

The algorithm runs in multiple iterations. Starting from the base station cell (0,0), we process one row and one column in each iteration. Specifically, the i^{th} row and the i^{th} column is processed in the i^{th} iteration. We find the first horizontal block in the i^{th} row and the first vertical block in the i^{th} column. If the vertical block is larger than the horizontal one, the column is processed first; otherwise, the

Notations

- n,(i, j): the network field is divided into $n \times n$ cells, each is represented as a 2-tuple (i, j). In this algorithm, we only consider the cells which the MS can talk to.
- $\mathcal{B}$: the set of blocks generated by this algorithm.
- $b(i, j, d, l)$: the largest vertical (if $d = 0$) or horizontal (if $d = 1$) block in which cell (i, j) is the left-bottom cell and the block contains l ungrouped cells, where a cell is *ungroup* if it is not in any block in $\mathcal{B}$.

The Algorithm

$i = 0$; $j = 0$; $B = \emptyset$
while ($i < n$ and $j < n$)
    find $b(i, j, 0, l_0)$ and $b(i, j, 1, l_1)$
    if ($l_0 \leq l_1$)
        process_row(i,j); process_column(i,j)
    else
        process_column(i,j); process_row(i,j)

process_row(i,j)
    add $b(i, j, 1, l_1)$ to $\mathcal{B}$
    while (existing ungrouped cells on this row)
        find the next horizontal block $b(k, j, 1, l)$
        if ($l > 1$) add $b(k, j, 1, l)$ to $\mathcal{B}$
        else add $b(k, j, 0, l')$ to $\mathcal{B}$
    $i = i + 1$

process_column(i,j)
    add $b(i, j, 0, l_0)$ to $\mathcal{B}$
    while (existing ungrouped cells on this column)
        find the next vertical block $b(i, k, 0, l)$
        if ($l > 1$) add $b(i, k, 0, l)$ to $\mathcal{B}$
        else add $b(i, k, 1, l')$ to $\mathcal{B}$
    $j = j + 1$

Figure 3.2. The basic algorithm for finding blocks

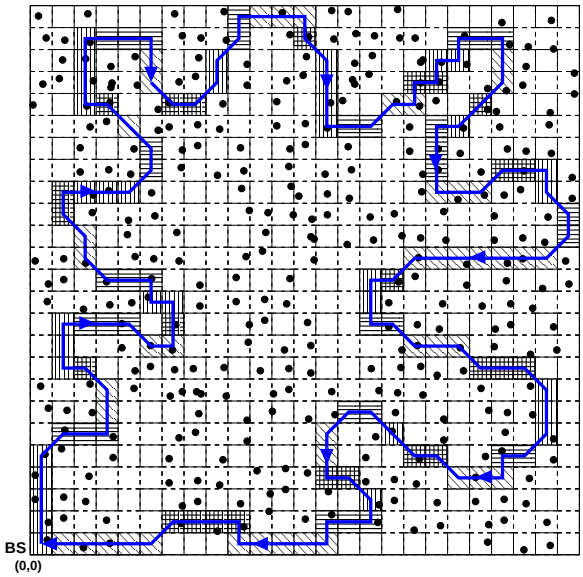


Figure 3.3. A sensor network field is divided into cells and a mobile sink traverses the field along a predetermined trajectory

order is reversed. When processing a column, all the vertical blocks in the column are identified. If an identified vertical block has only one cell, we will replace it with a horizontal block starting from this cell. A row is processed in the similar way. The process continues until all blocks have been identified. The cell merging algorithm is formally presented in Fig. 3.2. Fig. 3.3 shows the blocks identified using this algorithm.

Block Compression Our second technique generates a single id for the MS based on a Merkle hash tree [88]. Suppose we have obtained m blocks after running the above cell-merging algorithm. Let the id of block i be B_i. A Merkle hash tree is constructed in a bottom-up fashion using block ids as the leaf nodes. A non-leaf node in the tree is a hash of its two child nodes, recursively until the root node X_{1m} is generated. Fig. 3.4 depicts an example where $m = 8$. Here $X_{18} = H(X_{14}|X_{58})$, $X_{14} = H(X_{12}|X_{34})$, $X_{34} = H(B_3|B_4)$, and H is a collision-resistant hash function. we can derive the id of the MS node as

$$MS = H(TT|Ts|Te|X_{1m}) \tag{3.7}$$

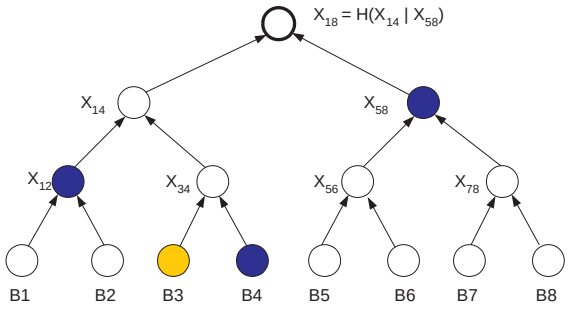


Figure 3.4. A Merkle-hash tree constructed from the ids of the blocks to be traversed by a mobile sink

To establish a pairwise key with a node u in block B_i, the MS provides MS, TT, Ts, Te, X_{1m} as well as several auxiliary values in the Merkle hash tree allowing node u to verify X_{1m} efficiently. The auxiliary values are the nodes sibling to the nodes on the path from B_i to the root X_{im}. Suppose node u is in block B_3 in Fig. 3.4. MS provides node u with B_3, B_4, X_{12}, and X_{58}. Node u first checks if its own cell is in block B_3. If so, it derives $X_{18} = H(H(X_{12}|H(B_3|B_4))|X_{58})$; otherwise, it terminates the verification process. Next node u derives the id MS based on Eqn. (3.7) and further computes its pairwise key $f(u, MS)$ shared with the MS. The MS also computes $f(MS, u)$. Finally, as in the strawman scheme, node u and MS provide mutual authentication using their pairwise key as the MAC key.

To establish a pairwise key with another node v in Block B_6, the MS also provides with MS, Ts, Te, X_{1m}, but the auxiliary values it presents are B_5, B_6, X_{78}, X_{14}. Thus node u derives $X_{18} = H(H(X_{14}|(H(B_5|B_6)|X_{78}))$. The remaining steps are the same as in pairwise key establishment with node u. More generally, with the same node id, the MS can establish a pairwise key with any node in these predetermined blocks within a specific time interval. Note that only one share is assigned to the MS and no knowledge about the network topology is required in advance.

3.3.2.3 Defending Against Denial-of-Service Attacks

Finally, we address a specific denial-of-service (DoS) attack against our scheme. This attack is possible because of the small packet size (29-byte payload in TinyOS [5]) used in sensor networks and mutual authentication being the last step. Recall that the first message sent from a MS to a host node includes the id MS and $log(m)$ auxiliary values besides the others. To be computationally secure, the size of the id MS or an auxiliary value should be at least the same as that of a symmetric key, which is normally 8 bytes for sensor networks. This limits the number of auxiliary values carried in a packet to at most three. As such, the MS has to send the message via multiple packets. A host node has to receive all the packets to reconstruct the message, computes MS, and finally proceeds to the mutual authentication process. Only after the mutual authentication has been done can a host node know if the MS is authenticated or not. That is, a host node cannot verify a received packet immediately. An attacker may exploit this security vulnerability to launch DoS attack against a host node. The attacker can send a large number of false packets to a host node to overrun its buffer and to entangle it in processing false packets and sending authentication messages.

Next we propose two enhancements to address the above security weaknesses. First, a host node u and a MS execute the mutual authentication process before the MS provides the parameters regarding the task.

$$MS \rightarrow u : MS, seq, MAC(f(MS, u), MS|seq) \tag{3.8}$$

$$u \rightarrow MS : u, seq + 1, MAC(f(u, MS), u|seq + 1). \tag{3.9}$$

When node u receives the message from the MS, it computes its pairwise key shared with MS, then verifies if the message is authenticated. This authentication-first strategy prevents DoS attacks launched by an outsider attacker which does not have a valid polynomial share. However, this scheme still cannot prevent insider attacks because node u cannot tell if the MS is a mobile sink or a compromised regular sensor node in the network.

Our idea to address this problem is to construct the ids for regular sensor nodes and the ids for MS nodes differently so that a host node can tell immediately if the

one it is talking to is a MS or a regular sensor node. Recall that the id for a MS is a pseudo-random number output from a hash function and the size of the id is, for example, 8 bytes. Differently, we can choose the ids for regular sensor nodes with certain patterns. A simple pattern is that the ids are integers between 1 and N, where N is the number of sensor nodes in the network. Consider a network size of $N = 65,536$, where a node id can be represented by 2 bytes. Normally the id of a MS will not fall into the interval $[1, 65536]$ because it is unlikely that all the other 6 bytes of a MS id output from a hash function are all zeros. Nevertheless, in case that the rare situation happens, we can change either the value T_s or T_e slightly to derive a MS id falling outside of the interval $[1, N]$ while without sacrificing much security. Thus, this approach can prevent a compromised sensor node from impersonating a MS.

Once a host node is certain that the MS id is valid via the authentication process, it proceeds to verify if it is a host node for the MS and if the MS is authorized for the task to be carried out. A MS is required to provide TT, T_s, T_e, X_{1m} and the auxiliary values, which usually have to be sent in multiple packets. To enable a host node to verify every received packet immediately, for every packet to be sent, the MS provides a MAC using its pairwise key shared with the host node as the MAC key. This prevents any other nodes from injecting packets while impersonating a MS. After obtaining all the packets and deriving X_{1m}, a host node verifies if all the parameters (e.g., time and location) regarding the claimed task are authenticated. If the verification succeeds, it assists the MS for the task; otherwise, it knows the MS is compromised and drops future packets from the MS.

3.3.3 Security and Performance Analysis

Next we analyze the security and the performance of our proposed scheme.

Security Analysis The security of our scheme is based on the assumption that Merkle hash tree, hash function, and the MAC algorithm are secure. In practice, as long as we pick a proper hash function and a MAC algorithm (e.g., using RC5 [89]) and the size of the hash/MAC output is large enough, the scheme guarantees that a compromised MS cannot lie about the type of task, the locations

and the time interval in which it is authorized to carry out the task. Since a MS only possesses one polynomial share, an attacker will not gain more advantages from compromising a MS instead of a regular sensor node if its goal is to recover $f(x, y)$. Moreover, our scheme with the DoS-resistance enhancement enables a regular sensor node to verify every received packet from a MS immediately, thus protecting its packet buffer space from being overflowed by false packets injected by other nodes.

Performance Analysis

- *Computational Cost* Given the auxiliary values in a Merkle hash tree, a host node computes $log(m)$ hashes to get the root value, where m is the number of blocks; it performs one hash computation to derive the id of the MS and two MAC computations (one for generating a MAC and the other for verifying a MAC) during mutual authentication. In practice, if RC5 [89] is used for providing all these security primitives, the total number of RC5 computations is about $log(m) + 3$.

  Now let us consider the computational cost for computing a pairwise key based on the Blundo scheme. Let the degree of a polynomial share be t, the size of a coefficient in a polynomial be 64 bits, and the sizes of a regular node id and a MS node id be 16 bits and 64 bits, respectively. For a MS node, it evaluates the id of a regular node at its polynomial share. The number of modular multiplications is $t+1$ and every modular multiplication is between a 64-bit number and a 16-bit number. For a host node, it evaluates the id of a MS at its polynomial share. The number of modular multiplications is also $t + 1$, but every modular multiplication is between two 64-bit numbers. Hence, the computational overhead for a host node in computing a pairwise key is 4 times as large as that for a MS node. It has been shown [43] that when $t = 50$ and an id is 16 bits, the number of CPU cycles for computing a pairwise key based on the Blundo scheme is equivalent to that for computing a RC5 MAC; therefore, when $t = 50$, the computational cost on a host node side is equivalent to computing four RC5 MACs.

  Combining the computational costs in all the above processes. When $m = 64$

and $t = 50$, the overall computational cost taken by a host node is equivalent to $log(m) + 3 + 4 = 13$ RC5 MACs. It has been shown that the energy a sensor node spends on computing one MAC is about the same as that used for transmitting 1 byte [24]. Thus, from energy point of view, the energy used by a host node to establish a pairwise key with a MS node is about the same as that used in transmitting 13 bytes. The computation cost of a MS is smaller because it does not need to compute hashes. In practice, we believe this is an affordable overhead for the current generation of sensor nodes.

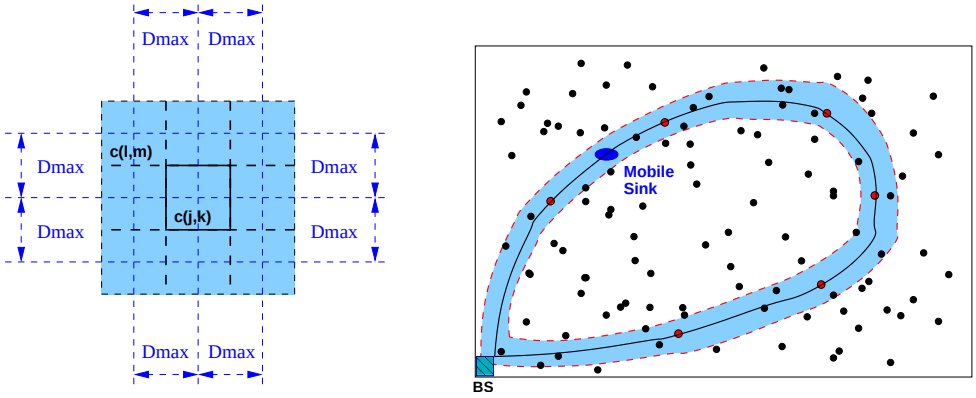
- *Communication Cost* The authentication between a MS and a host node involves totally three messages. The first message sent from the MS to a host node includes TT, T_e, T_s, MS and $log(m)$ auxiliary values in the tree, where m is the number of blocks to be traversed by the MS. The two authentication messages are very small because they only include a nonce and a MAC.

- *Storage Overhead* A regular sensor node only needs to store its polynomial share, i.e., $(t + 1)$ coefficients. In addition to storing a polynomial share, a MS stores all the block ids. As an example, suppose a sensor field is divided into 64 by 64 cells and the maximum size of a block is 8 cells, then we can use 2 bytes to represent a block id. Hence, even if a MS is to traverse hundreds of blocks and its resources are as scarce as those of a Mica2-based mobile sensor [72], the storage overhead is still not a concern.

We will show some detailed results through our implementation in Section 3.5.

3.3.4 An Extension: Enabling Trajectory Changing

The trajectory discussed previously is fixed. However, to accomplish the mission, the MS may need to change its trajectory in certain conditions, such as meeting large obstacles (e.g., enormous rocks) or coverage holes (e.g., a big pool) along the original trajectory. In a battlefield scenario, such a condition could be the discovery of enemies' appearance in certain area of the trajectory. As another example, suppose the sensor network is deployed to monitor the temperature in an area and the MS is sent to collect the sensed data. When the MS collects an



(a) Determine whether a cell is conditionally accessible or not using system parameter D_{max}.

(b) Example of a strip of cells around the trajectory which is accessible to the MS when certain conditions are met.

Figure 3.5. Illustration of conditional trajectory change.

abnormal temperature datum, it needs to search around and finds out how big the abnormal area is. Such investigation requires the MS to change the trajectory too.

To enable the MS to change the trajectory, we extend our scheme and propose a *conditional* privilege granting scheme. The basic idea is that, when certain conditions are met, the MS is allowed to expand its trajectory and access certain cells that are not on the original trajectory.

To achieve this goal, we integrate condition(s) into the ticket that is granted to the MS and we called this ticket a *conditional ticket*. Using the conditional ticket, the MS can access a strip of cells around the trajectory when certain conditions are satisfied. The strip of accessible cells can be determined by their distance to the trajectory. That is, if the distance of a cell from the trajectory is less than a system parameter D_{max}, this cell will be accessible to the MS. More formally,

Definition 1. *Suppose node u is in cell(l, m). Given D_{max}, condition C, and the last accessed cell cell(j, k), if $|l - j| \leq D_{max}$ and $|m - k| \leq D_{max}$, we say that cell(l, m) is* conditionally accessible *to the MS given condition C.*

Fig. 3.5(a) shows the conditional accessible cells around a cell on the trajectory. Fig. 3.5(b) shows the strip of cells around the trajectory which are accessible to the MS using the definition above.

Next, we describe the conditional privilege granting scheme in detail. In the

original scheme, the MS only possesses one polynomial share and one ticket (i.e., the id of the MS constructed using Eqn. 3.7). Similarly, we construct the conditional ticket as follows

$$MS = H(TT|Ts|Te|X_{1m}|C_i), \tag{3.10}$$

where C_i is the condition(s) under which the MS is allowed to change the trajectory. C_i can be Arabic numbers such as $1, 2, 3, ..., n$, denoting various predefined conditions. Alternatively, it can be more descriptive, e.g., $C_i = \{$the temperature is higher than 120F$^\circ\}$ or $C_i = \{$there is no sensors in cell(10,10)$\}$, and so on. Depending on the application, we can either preload D_{max} into each sensor (if it is fixed) or add it into the conditional ticket (if it is variable).

To establish a pairwise key with a node u in block B_i, the MS provides MS, TT, Ts, Te, X_{1m}, together with several auxiliary values in the Merkle hash tree, which is similar to the original scheme. In addition, the MS also needs to provide C_i and $cell(j, k)$ to node u, where $cell(j, k)$ is the last successfully accessed cell on the trajectory.

When receiving these values from the MS, node u first checks whether C_i is in the ticket. If there is no C_i in the ticket, no trajectory change is needed and node u and the MS can mutually authenticate each other as in the original scheme. Otherwise, if C_i is in the ticket and node u's cell is in the strip around the trajectory (based on D_{max}), node u will need to verify whether the condition (C_i) holds before granting MS's request.

Next, we use a simple example to explain how to verify the condition C_i at a node u. Suppose the condition C_i equals to one, denoting trajectory changing is allowed when there is a coverage hole. If node u is close to the coverage hole, then, it can verify the existence of the hole in several ways. First, it may have observed the hole by itself, if it has never received any messages from the cells in the hole area for a long time. If this is the case, node u can confirm the condition at once and grant the request from the MS, if it can mutually authenticate with the MS based the conditional ticket. Alternatively, when queried by the MS, node u can send a query message (which contains C_i) to its neighbors to check whether they have observed the same condition C_i as well. Note that this message should be

authenticated by the pairwise key shared between node u and each of its neighbors, respectively. (Node u may also simply broadcast the message encrypted using the one-hop key (or cluster key) shared with its neighbors if LEAP protocol [54] is used.) Each neighbor of u should generate a true/false report on condition C_i, attached with a keyed MAC and send it to node u. After collecting all the reports, if node u concludes that the condition holds, it proceeds to verify MS's conditional ticket and authorize MS's request at last.

In the spirit of the above scheme, we can construct more complex conditional tickets or assign multiple conditional tickets to a MS. The security and performance overhead of regular sensors and mobile sensors are similar to that of the original scheme.

3.4 Revoking a Mobile Sink On-Demand

If a mobile sink (MS) is detected compromised when its granted privilege is still valid, the MS should be revoked as soon as possible. To do this, the base station may send a revocation message to all the host nodes of the MS. As a simple approach, the base station may send a revocation message to each host node individually. However, this is not feasible since the base station may not know the ids of these nodes. Even the ids are known, the communication overhead increases rapidly as the number of host nodes increases. Alternatively, the base station may flood the revocation message over the network. This is still not efficient because all nodes are involved in receiving or forwarding the message. For efficiency, a revocation message should be multicast only within the smallest area (called *revocation area*) that covers the host nodes of the MS.

3.4.1 The Basic Scheme

To multicast a revocation message within the revocation area, the basic idea of location-based multicasting [90, 91] can be applied. The base station first generates a message which indicates the id of the MS to be revoked and the scope of revocation area. Then, the base station broadcasts the message to its neighbors. On receiving the message, each node acts as follows:

- If it has received the message before or is outside of the revocation area, the message is dropped.
- If it is within the revocation area indicated by the message, it records the id of the revoked MS, and rebroadcasts the message to its neighbors.

The basic scheme is efficient when the revocation area is regular; for example, it is a rectangle or a circle. In many scenarios, however, the revocation area could be in any arbitrary shape. Although an irregular area can be divided into and represented as a set of smaller regular (e.g., rectangular) subareas (as described in Section 3.3.2.2), this may need large space in a revocation message. For example, if a revocation area is represented as 100 rectangles, and each rectangle needs 4 bytes, 400 bytes are demanded to represent the area. This is not trivial for the current sensor network in which a packet contains only a few tens of bytes. To revoke the MS, $\lceil \frac{400}{29} \rceil = 14$ revocation messages must be multicast to the host nodes. In order to broadcast all these messages, each host node needs to record the forwarding path when receiving the first message for this revocation event. Based on this information, it can forward the following revocation messages received later. During this course, each host node has to receive and/or forward 14 packets.

3.4.2 Enhanced Schemes

To address the problems of the basic scheme, we use two techniques:

(1) The revocation area is divided into multiple subareas, and multiple revocation messages are sent to and multicast within these subareas simultaneously. Using this technique, the revocation delay can be reduced.

(2) The basic blocks forming the revocation area are further combined into a smaller number of blocks (called *expanded blocks*). Using this technique, the number of revocation messages can be reduced.

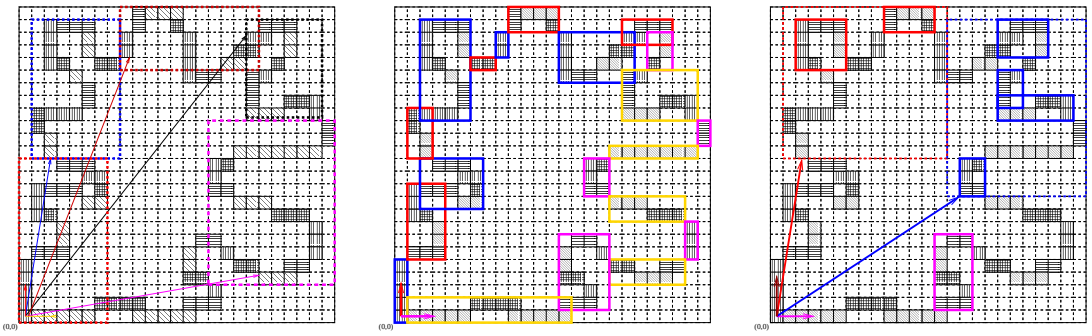
In the following, we first present two enhanced schemes, each is based on one of the above techniques. Then, we use both techniques to design another enhanced scheme.

OPT-I: GPSR-based Scheme When the revocation area is large or complicated, and it must be represented using multiple revocation messages, we may not solely rely on host nodes to forward these message. Instead, we can use the GPSR protocol [92, 93, 94] to send each revocation message to a certain node within the subarea indicated by the message, and then multicast the message within that subarea.

Fig. 3.6(a) illustrates how to multicast revocation messages using the GPSR-based scheme. Unlike in the basic scheme all the revocation packets follow the trajectory, in the GPSR-based scheme each packet is forwarded towards the first cell of its destination area, based on a path dynamically determined by the GPSR routing protocol. Once a revocation packet arrives at its destination area, it is multicast and forwarded based on the block ids contained in the packet. For illustration purpose, in the figure we plot the GPSR paths as straight lines, although in reality they could be more complex subject to factors such as node distribution and network topology.

An important advantage of the GPSR-based scheme over the basic scheme is that it can significantly reduce the revocation latency. This is because the base station can send multiple packets almost simultaneously along different paths and every path is the shortest path between the base station and its destination revocation area. In the basic scheme, multiple packets follow the direction of the trajectory, thus the sensor nodes within the cells (blocks) that are in the arriving direction to the base station in the trajectory always receive the revocation notification at the latest. In practice, we should have the high priority to notify these sensor nodes because it is likely a compromised MS has not contacted with them yet. Thus using the GPSR-based scheme allows us to notify each subarea of host nodes as early as possible.

On the other hand, the GRSR-based scheme introduces some additional overhead. When a revocation message is sent along a path towards a revocation area, all the nodes in the cells cross the path are involved in receiving and/or forwarding the message. We call these cells *redundant cells*. The cost of sending a revocation message can be measured by the number of redundant cells cross the path along which the messages is transmitted.



(a) OPT-I: GPSR-based scheme (b) OPT-II: Minimum message scheme (c) OPT-III: Optimized multi-message scheme

Figure 3.6. Illustration of the three enhanced privilege revocation schemes

OPT-II: Minimum Message Scheme Suppose the original revocation area is represented by b rectangular blocks (called *basic blocks*) using the algorithm presented in Section 3.3.2.2, and one packet can contain the representation of at most k ($k < b$) blocks. To reduce the number of revocation messages, we can further combine the basic blocks into a limited number (say, k) of larger rectangular blocks (called *expanded blocks*), such that only a minimal number of revocation messages are demanded. Note that, an expanded block can also be represented as a four-variable tuple (i, j, l_x, l_y), where (i, j) is the index of the starting cell, and l_x (l_y) is the length (in the unit of cell) of the block in the X (Y) dimension. So the representation of an expanded block has the same size as that of a basic block. During this expanding course, some cells that are not included in any basic block may be included in some expanded block. We also call these cells *redundant cells*. Multicasting a revocation message to the redundant cells is not harmful, but it increases communication overhead. Therefore, the number of redundant cells should be minimized. In the following, we present a dynamic programming-based approach to minimize the number.

- According to the visiting order, all the basic blocks are sorted into a sequence denoted as follows:

  $$B_1, B_2, \cdots, B_b.$$

  Here, b is the total number of basic blocks, and block B_i is visited earlier than any block B_j ($j > i$).

- Let $R_m(i,l)$ be the minimum number of redundant cells introduced when basic blocks $B_i, B_{i+1}, \cdots, B_b$ are combined into at most l expanded blocks, and $R(i,j)$ be the minimum number of redundant cells introduced when basic blocks B_i, $\cdots$, B_j are combined into a single expanded block. Then, the minimum number of redundant cells introduced when all the basic blocks into at most k expanded blocks, denoted as $R_m(1,k)$, can be computed as follows:

$$R_m(i,l) = \begin{cases} 0 & l = b, \\ R(i,b) & l = 1, \\ min\{R(i,j) + R_m(j+1, l-1) & \\ \mid i \le j \le b-l+1\} & \text{others.} \end{cases} \tag{3.11}$$

  Here, $1 \le i < b$ and $2 \le l \le k$.

A more detailed description of the algorithm is shown in Fig. 3.7. Fig. 3.6(b) shows an example, in which the revocation area is represented by 20 expanded blocks and two messages are used to revoke all the host nodes.

OPT-III: Optimized Multi-message Scheme The above enhanced schemes still have some limitations, especially when the revocation area is composed of a large number of basic blocks. In the GPSR-based scheme, we may need many revocation messages to represent the whole revocation area; i.e., the base station may have to send many revocation messages. The minimum message scheme can minimize the number of revocation messages, but lots of redundant cells may be introduced. Since the overall communication cost increases as the number of redundant cells or revocation messages increases, it is important to carefully design the representation scheme to minimize the overall cost.

In the following, we extend the algorithm described in Fig. 3.7 to design a new algorithm that can minimize the overall communication cost. We consider the overall communication cost including three parts:

- the necessary cost of multicasting a revocation message within the original revocation area;

Notations

- B_i, $R_m(i,l)$, $R(i,j)$: defined before.
- $next(i,l)$: when basic blocks $B_i, \cdots, B_b$ are combined into at most l expanded blocks, the first expanded block is composed of $B_i, \cdots, B_{next(i,l)-1}$; i.e., $next(i,l)$ is the starting cell of the next expanded block.

The Algorithm

```
for l = 1 to k
      R_m(b+1, l) = 0
for i = 1 to b
      R_m(i, 1) = R(i, b) /*initialization*/

for i = b to 1
      for l = 2 to k
            R_m(i, l) = +∞
            for j = i to b − l + 1
                  if R(i, j) + R_m(j + 1, l − 1) < R_m(i, l) then
                        R_m(i, l) = R(i, j) + R_m(j + 1, l − 1);
                        next(i, l) = j + 1

/*Following:
      output the starting cell of each expanded block*/
i = 1; print i
for l = k to 1
      if next(i, l) > b then end
      else print next(i, l)
      i = next(i, l)
```

Figure 3.7. The algorithm for the minimum message scheme

- the additional cost of multicasting the message within the redundant cells; and
- the additional cost of sending revocation messages from the base station to all the sub-scopes.

In the algorithm, we use the following notations:

- k,b: as in the previous algorithm, k is the maximum number of expanded

blocks that can be represented in a single packet, and b is the maximum number of basic blocks in the whole revocation area.

- $R_m(i, l, e)$: this is extended from the notation $R_m(i, l)$ defined in the previous algorithm. It represents the minimum number of redundant cells introduced when basic blocks B_i, B_{i+1}, $\cdots$, B_e are combined into at most l expanded blocks.

- $C(i)$: the number of cells involved in sending a revocation message from the base station to block B_i.

- $C_m(i)$: the minimum number of redundant cells introduced when sending revocation messages to host nodes in blocks $B_i, \cdots, B_b$.

Using the dynamic programming technique, $C_m(i)$ can be calculated as follows:

$$C_m(i) = \begin{cases} C(i) & i = b, \\ min\{R_m(i,k,j) + C(i) + C_m(j) & \\ \quad | \; i \le j \le b - l + 1\} & \text{others.} \end{cases} \tag{3.12}$$

Fig. 3.6(c) shows the result of applying the optimized multiple message scheme on the previous example. After using this scheme, the 57 basic blocks are combined into 40 expanded blocks, and 4 revocation messages are used to revoke the MS.

3.4.3 Discussions

During the description of all the four revocation schemes, we have implicitly assumed that a revocation packet can arrive at the start forwarding cell reliably and correctly. If the adversary has compromised some stationary sensor nodes, these nodes may attack the revocation schemes by dropping the revocation messages that they should forward, modifying passing revocation messages, or inserting false revocation messages.

To prevent compromised nodes from modifying revocation messages or inserting false revocation messages, the messages should be authenticated. For example, we may use the μTESLA scheme [20] for this purpose. Packet dropping cannot be

prevented, but may be detected. The *watchdog* mechanism [26] and the reputation-based scheme [27] may be employed to thwart a compromised node from dropping revocation messages.

3.4.4 Performance Evaluations

We evaluate the performance of the revocation algorithms by simulations. We assume that a revocation packet can only store ten blocks. As a result, multiple revocation packets are needed when the number of blocks are larger than ten. Four revocation algorithms will be evaluated: the basic algorithm, the GPSR-based scheme (OPT-I), the minimum message scheme (OPT-II), and the optimized multi-message scheme (OPT-III). All these four schemes have been discussed in the last section.

Three metrics are used to evaluate the performance of the proposed schemes: the *average revocation delay*, the *maximum revocation delay*, and the *message overhead*. The revocation delay for a cell is defined as the number of hops that the revocation message transmits before it reaches the cell. The average revocation delay is the average of all revocation delays for all cells on the trajectory. Similarly, the maximum revocation delay is the maximum among all the delays. The message overhead is defined as the total number of transmitted hops of all the revocation messages. In OPT-II and OPT-III, the message overhead also includes the messages due to using redundant cells.

These four algorithms are evaluated using four typical trajectories: triangle, polygon, ellipse and irregular shape, and the results are illustrated in figures 3.8, 3.9, 3.10 and 3.11, respectively.

From the figures, we can make the following observations. First, all three optimization schemes outperform the basic scheme. In the basic scheme, all the revocation messages are sent in one direction along the trajectory, resulting in a large revocation delay and high message overhead.

Second, in terms of revocation delay, the OPT-I scheme performs the best. In the OPT-I scheme, the revocation messages are sent in both directions simultaneously. Instead of following the trajectory, each message is forwarded to the first block in the message using the shortest path, thus reducing the revocation delay.

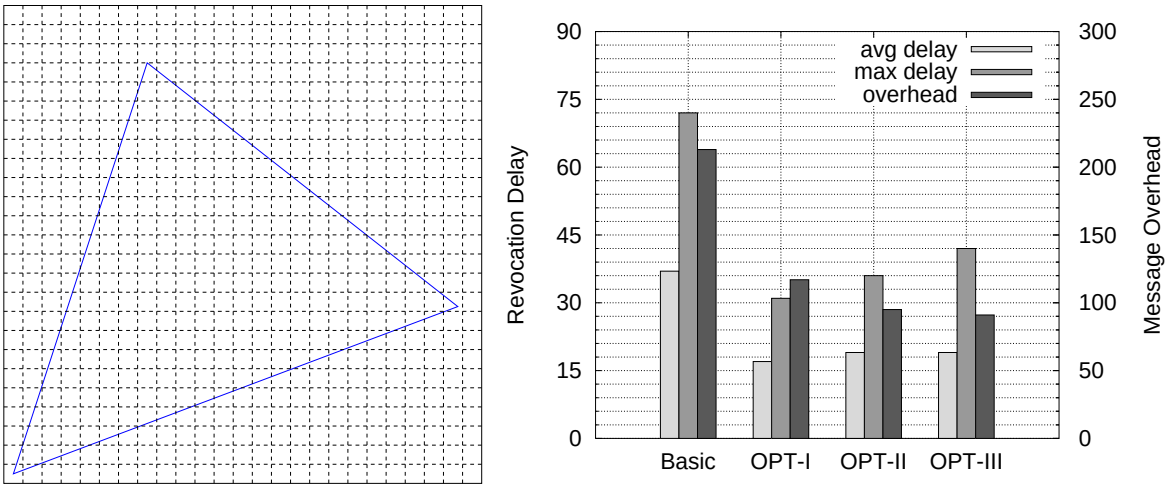


Figure 3.8. Triangle trajectory

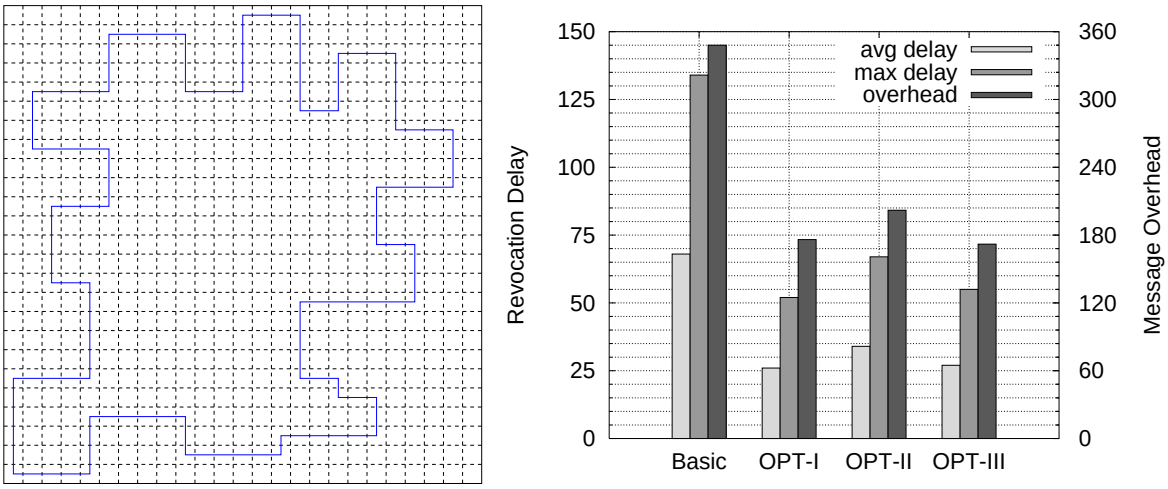


Figure 3.9. Polygon trajectory

In addition, the number of messages are larger, or at least equal to, the other two optimized schemes, which results in smaller number of blocks in each message. The smaller the number of blocks in a message, the faster the message can be forwarded to them. All these explain why OPT-I has the lowest revocation delay among all the schemes.

Third, in terms of message overhead, the OPT-III scheme performs the best. Both OPT-II and OPT-III are designed to minimize the message overhead. As a result, both outperforms OPT-I in terms of message overhead. Furthermore, the OPT-III scheme is designed to minimize the message overhead using multiple messages. Therefore, OPT-III further reduces the message overhead compared to

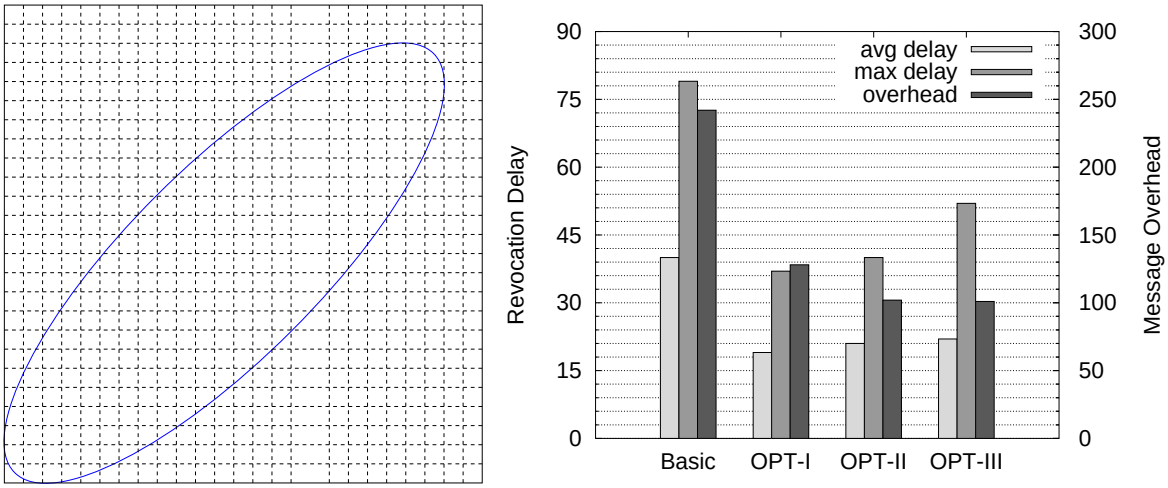


Figure 3.10. Ellipse trajectory

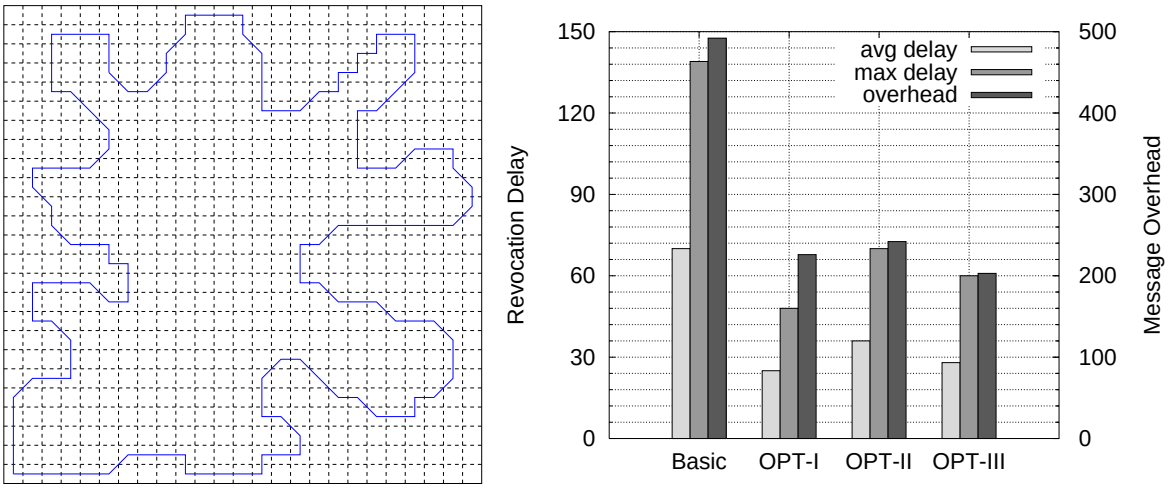


Figure 3.11. Irregular shape trajectory

OPT-II.

Fourth, from figures 3.9 and 3.11, we can see that OPT-II does not perform very well compared to OPT-I and OPT-III when the trajectory is complex. This can be explained as follows. In the OPT-II scheme, no matter how many blocks exist, the scheme always ends up with two revocation messages. Hence, the revocation delay is high when the trajectory is complex and the number of blocks is high. The message overhead is also increased due to the use of redundant cells.

In summary, there is a tradeoff between revocation delay and message overhead. In practice, we can choose different revocation schemes based on the design goals and the requirements of the applications. For example, if the design goal is to

revoke the MS faster, we can use the OPT-I scheme. On the other hand, if the design goal is to revoke the MS with minimum message overhead, we can choose the OPT-III scheme.

3.5 Implementation

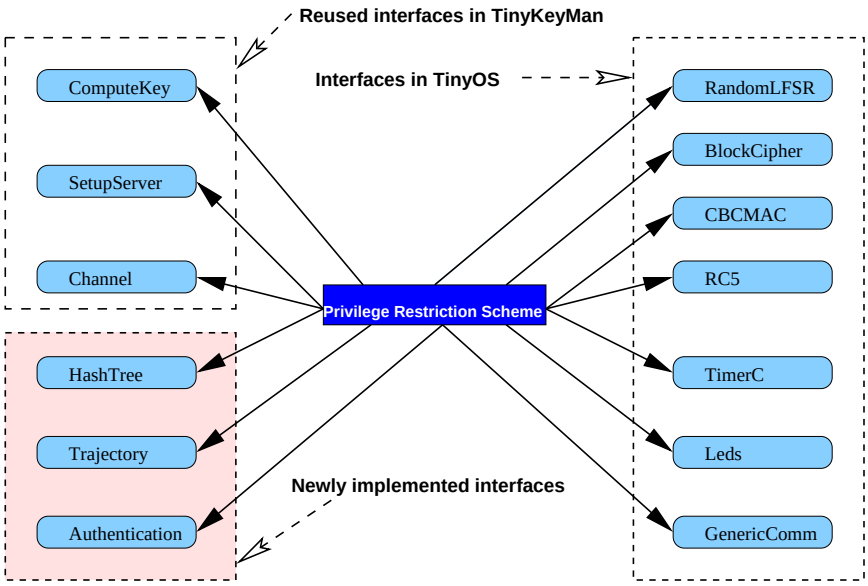


Figure 3.12. The components and the interfaces used in implementing the proposed privilege restriction scheme. The seven interfaces on the right are directly adopted from TinyOS. On the left, the top four interfaces are provided in the TinyKeyMan package and the bottom three are new interfaces implemented by us.

We have implemented our scheme of the privilege restriction schemes on Mica2 motes [5] in the TinyOS platform [95]. The programs were written in nesC [96], a C-like programming language used for developing applications in TinyOS. Our implementation does not include the privilege revocation schemes yet, since currently the basis of them, the GPSR scheme, is not implemented in TinyOS. We shall implement them when the GPSR scheme becomes available in TinyOS.

Our implementation is based on the tiny key management package (TinyKeyMan) provided by Liu and Ning at North Carolina State University. TinyKeyMan provides an implementation of the polynomial pool-based key predistribution in sensor networks. In our case, we adapted it to assign the shares of the same polynomial to the sensors and the MS based on node id. In addition, we added 2,987

lines of new code in implementing our privilege restriction scheme.

3.5.1 System Architecture

Fig. 3.12 depicts the components and the interfaces used by our implementation. Of the interfaces shown in the figure, the seven interfaces on the right are directly adopted from TinyOS (with most of them from the TinySec package). For instance, we use the RC5 block cipher [89] to provide the CBC-MAC [97].

The top four interfaces on the left are from the TinyKeyMan package. Among them, the *ComputerKey* interface is used for computing the pairwise key based on bivariate polynomial evaluation. The *SetupServer* interface assigns a polynomial share to a node based on its node id. For the MS, the polynomial share is assigned based on its id calculated using Eqn. 3.7. The *Channel* interface implements the channel module.

The bottom three interfaces on the left are interfaces implemented by us. The *Trajectory* interface implements the cell merging process; the *HashTree* interface implements the block compressing process; and the *authentication* interface implements the authentication procedure between a sensor node and the MS. The first two interfaces are implemented only in MS; whereas the *Authentication* interfaces are implemented in both MS and static sensor nodes. In the following, we describe these interfaces and the packet formats in details.

3.5.2 The Trajectory Interface

Currently, this interface includes four typical trajectories, i.e., triangle, polygon, ellipse and irregular shape. An array that consists of the ids of the cells on the trajectory is used to store one trajectory. For example, the ellipse trajectory shown in Fig. 3.10 covers 79 cells. Each *cell id* is represented by a dual-pair (i, j), where i and j are the cell's x and y axis, respectively. Thus, in our implementation, the ellipse trajectory is represented by an array that contains 79 aforementioned cell ids. Given a trajectory, the cell merging algorithm shown in Fig. 3.2 is called to merge the cells into blocks.

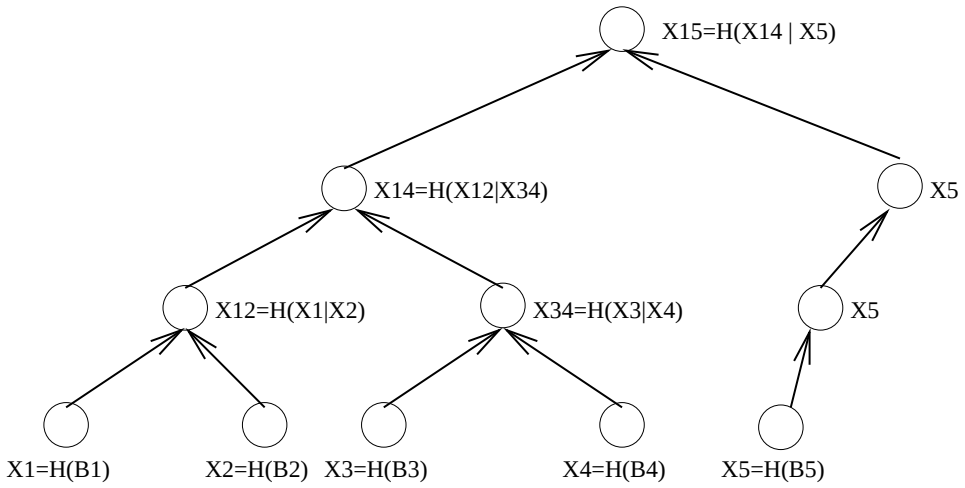


Figure 3.13. Unbalanced Merkle hash tree. X_5 does not have any immediate siblings with which to be combined to calculate the next generation. So, X_5 is promoted up the tree, without being rehashed, until it can be paired with value X_{14}. The value X_5 and X_{14} are then concatenated, and hashed, to produce the root hash X_{15}.

3.5.3 The HashTree Interface

Based on the set of blocks output from the *Trajectory* interface, a Merkle hash tree is constructed in the *HashTree* interface. We first blind the block ids using the one-way hash function. Then, we compute the levels of the tree recursively from the leaf nodes to the root. A non-leaf node in the tree is a hash of its two child nodes, that is $X_i = H(X_i\text{'s left child}|X_i\text{'s right child})$, where H is a collision-resistant hash function. We considered three important issues when implementing the Merkle hash tree in TinyOS. (These issues are first discussed in [98], where Merkle hash tree is used for file integrity verification.)

First, the strength of the hash tree construction is only as strong as the underlying hash algorithm H. Thus, we use a secure hash algorithm CBC-MAC [97] as the basis of the hash tree. CBC-MAC is efficient, and the fact that it relies on a block cipher minimizes the number of cryptographic primitives we must implement in the limited memory we have available. CBC-MAC is provably secure, however, the standard CBC-MAC construction is not secure for variably messages. Bellare *et al.* [97] suggested three alternatives for generating MACs for variable sized messages. In our implementation, we adopted the CBC-MAC implementation in TinySec, which is the variant that XORs the encryption of the message with the first plaintext block.

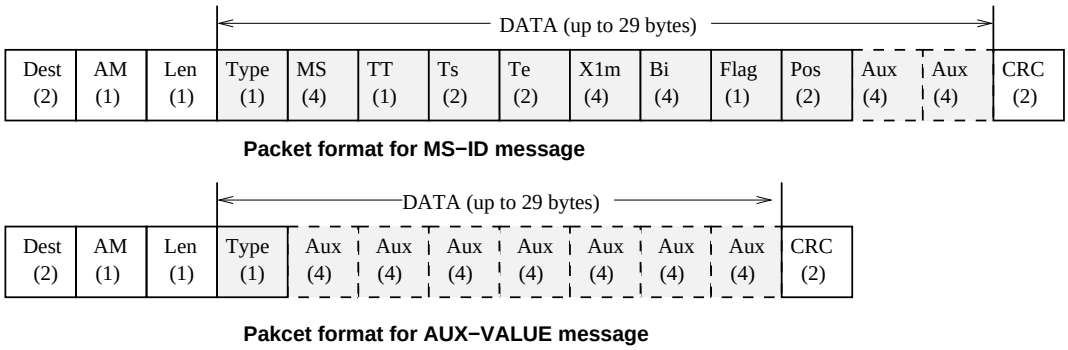


Figure 3.14. Packet Formats

Second, in order to reduce the collisions between leaf hashes and internal hashes, different hash constructions are used to hash the leaf nodes and the internal nodes. The same hash algorithm, CBC-MAC, is used as the basis of each construct, but a single '1' byte in network byte order, or 0x01 is appended to the input of the internal node hashes, and a single '0' byte, or 0x00 is appended to the input of the leaf node hashes.

Third, since a trajectory has arbitrary number of blocks, the constructed Merkle hash tree might be an unbalanced tree, i.e., the number of leaves is not power of 2. In such a case, the tree is constructed as follows. Interim hash values (of internal nodes) which do not have a sibling value to which they may be concatenated are promoted, unchanged, up the tree until a sibling is found. For example, Fig. 3.13 shows the hash tree for a trajectory made up of five blocks, B1, B2, B3, B4 and B5.

The *HashTree* interface can also generate the authentication path for a leaf node. The authentication path consists a set of auxiliary values in the Merkle hash tree that allows a sensor node in a certain leaf node (or block) to verify the MS's id efficiently.

3.5.4 Packet Formats

Our implementation involves two new message types: MS-ID message and AUX-Value message. The packet formats for the two messages, shown in Fig. 3.14, are based on the current packet format in TinyOS. The common fields are *destination address, active message (AM) type*, and *length*. Active message types are similar to

port numbers in TCP/IP. The AM type specifies the appropriate handler function to extract and interpret the message on the receiver. The data field is up to 29 bytes in a TinyOS packet. In these two packets, the *type* field designates the message type, which is 7 for the first one and 8 for the second one. Messages of type 1-6 are previously defined in the TinyKeyMan package.

3.5.4.1 MS-ID Message Packet Format

MS-ID message is used to send MS, TT, T_s, T_e, X_{1m}, B_i, flag, the auxiliary values' position information, and up to two auxiliary values (optional) on the authentication path to the sensor.

the auxiliary values' position information is needed for a sensor to correctly reconstruct X_{1m}, the root of the Merkle hash tree, using the authentication path. Recall that a non-leaf node is the hash over the concatenation of its left and right children, where the left child is before the right child. We will get a different hash if the right child is before the left child. Thus, the sensor must have the position (left or right) information of each auxiliary values; otherwise it cannot reconstruct X_{1m} successfully even if it has the correct authentication path.

The auxiliary values' position information is contained in the *Pos* field in a MS-ID message. Each bit of it represents the position of one auxiliary value. If the corresponding auxiliary should be on the left, the bit is set to 0; otherwise the bit is set to 1. Since *Pos* field is 2 bytes, it can be used to hold up to 16 auxiliary values. This means that the Merkle hash tree could have up to 64 thousand blocks, which should be big enough to store an enormous trajectory.

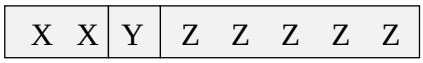


Figure 3.15. The Flag field in the MS-ID message

The *flag* field, shown in Fig. 3.15, is described as follows. The first 2 bits (XX) shows how many auxiliary values behind. XX could be 0, 1, or 2, which means that zero, one or two auxiliary values behind, separately. The last 5 bits ($ZZZZZ$) designates the total number of auxiliary values on the authenticate path. The third bit (Y) indicates whether all the auxiliary values are here in the MS-ID message or not. If the total number of auxiliary values is not larger than

two, $Y = 0$; otherwise $Y = 1$, which indicates that this message will be followed by an AUX-VALUE message.

3.5.4.2 AUX-VALUE Message Packet Format

AUX-VALUE message is used for sending auxiliary values on the authentication path. Each AUX-VALUE message can hold up to seven auxiliary values, considering each auxiliary value is 4 bytes. Combining the MS-ID message with one AUX-VALUE message, we can send nine auxiliary values using two messages. Thus, by piggybacking two auxiliary values in a MS-ID message, we can support the authentication of a trajectory of up to 512 blocks ($2^9 = 512$), which is often big enough to hold a typical trajectory.

3.5.5 The Authentication Interface

Lastly, the *Authentication* interface implements the message sending and receiving in the authentication procedure between a sensor node and MS. Its implementation on MS is different from that of on a static sensor node.

On MS, before sending any messages, the *Authentication interface* first calls the Trajectory interface to merge cells into blocks; then the *HashTree* interface is called to generate X_{1m}, which, in turn, is used to generate mobile sink's id MS. After that, MS sends a MS-ID message and a AUX-VALUE message to the current senor node u on the trajectory using the *Channel* interface. If node u can successfully verity the MS id, similar to the strawman scheme in Section 3.3.1, MS sends an authentication message (Eqn. 3.2) to node u. Later on, when MS receives the reply message (Eqn. 3.3) from node u, it can confirm whether node u has the same pairwise key by verifying the MAC in the reply message.

On a static sensor node, the *Authentication* interface processes the received MS-ID message and AUX-VALUE message to verify the id of MS and compute the pairwise key shared with MS by evaluating its polynomial share on MS's id. After it receives authentication message (Eqn. 3.2) from the MS, it can check whether it has the same pairwise key with MS by verifying the MAC in the message. If it can successfully verify the MAC, message (Eqn. 3.3) will be sent back to the MS.

3.5.6 TinyOS Code Size

We upload and run the code in Mica2 motes. Since the TinyOS code of a MS is different from that of a static sensor node, MS and static sensor node have different code sizes. In the case of MS, the memory used for code and data depends on the trajectory. A more complex trajectory often consists of a greater number of blocks. Since these block information need to be stored in the MS, a more complex trajectory often requires more ROM/RAM space. This situation can be observed from Table 3.1, which shows the usage of ROM/RAM for MS with four different trajectories. The four trajectories are previously showed in figures 3.8, 3.9, 3.10, and 3.11. For the most complex trajectory, the irregular shape one, the required code space is 25 KB (out of 128 KB) in ROM and about 800 Bytes (out of 4 KB) in RAM for the MS.

Table 3.1. The required ROM/RAM space for MS with different trajectories.

Trajectory	Triangle	Polygon	Ellipse	Irregular Shape
ROM (bytes)	24,792	24,806	24,916	24,926
RAM (bytes)	609	597	657	771

In the case of static sensor node, the required code space is 17.2 KB (out of 128 KB) in ROM and 672 Bytes (out of 4 KB) in RAM. From the above results, We can see that the memory requirements for the privilege restriction scheme are reasonable given that Mica2 motes have 4 KB of RAM and 128 KB of ROM for data and code storage.

3.6 Discussions

In this chapter, we identified new security challenges of using MSs; if a MS is given too many privileges, the compromise of the MS may break down the whole network; on the other hand, without some necessary privilege, the MS may not be able to accomplish the planned mission. Based on the *principle of least privilege*, we first proposed several efficient schemes to restrict the privilege of the MS without impeding its capability of carrying out any authorized operations for an assigned task. In addition, we present an extension to allow conditional trajectory change due to unexpected events. To further reduce the possible damage caused

by a compromised MS, we then proposed efficient message forwarding schemes for depriving the privilege assigned to the compromised MS immediately after its compromise has been detected. Detailed analysis, simulation, and implementation results show that our schemes are secure and efficient, and are highly practical for sensor networks consisting of the current generation of sensors.

To the best of our knowledge, this is the first work to address the MS privilege issues in wireless sensor networks. As the initial work, we do not expect to solve all the problems. In the future, we shall look into other revocation techniques tot balance the tradeoff between delay and message complexity.

Chapter 4

Attack-resilient Time Synchronization in Wireless Sensor Networks

The existing time synchronization schemes are designed without security in mind and are vulnerable to malicious attacks. In this chapter, we focus on a specific type of attack, called *delay attack*, which cannot be addressed by cryptographic techniques. We have proposed two approaches to filter the outlier data (caused by delay attack) using time transformation technique and statistical method, respectively.

4.1 Overview

Many sensor network applications require time to be synchronized within the network. Examples of such applications include mobile object tracking, data aggregation, TDMA radio scheduling, message ordering, to name a few. Consider the application of mobile object tracking [99], in which a sensor network is deployed in an area of interest to monitor passing objects. When an object appears, the detecting nodes record the detecting location and the detecting time. Later, these location and time information are sent to the aggregation node which estimates the moving trajectory of the object. Without an accurate time synchronization, the

estimated trajectory of the tracked object could differ greatly from the actual one. Similarly, we can see the importance of time synchronization for the operations of other sensor network applications.

All network time synchronization methods rely on some sort of message exchanges between nodes. Nondeterminism in the network dynamics such as physical channel access time or operation system overhead (e.g., system calls), makes the synchronization task challenging in sensor networks. In the literature, many schemes have been proposed to address the time synchronization problem [100, 101, 102, 103, 104]. These schemes involve the exchange of multiple time synchronization messages among multiple sensor nodes [100] or between two sensor nodes [101] to be synchronized. However, none of them was designed with security in mind, even though security has been identified as a major challenge for sensor networks [105]. Actually, even if an adversary is capable of destroying some or all sensor nodes, it may opt for other more severe attacks, since it is more dangerous to take actions based on some false sensor data than without any data. For example, if an adversary can attack the time synchronization protocol so that the estimated direction of a mobile object is contrary to its actual direction, a wrong or even risky action may be taken and many system resources may be wasted. Thus, when a sensor network is deployed in an adversarial environment such as a battlefield, the time synchronization protocol is an attractive target to the adversaries.

In this chapter, we first identify several security attacks an adversary can launch against a non-secure time synchronization protocol. For instance, an attacker can replay old synchronization messages, drop, modify, or even forge exchanged timing messages. Since many of these attacks can be addressed by employing appropriate cryptographic techniques, we focus on a specific type of attack called *delay attack* which cannot be addressed by the cryptographic techniques. In the delay attack, a malicious attacker (or a compromised node) deliberately delays the transmission of time synchronization messages to magnify the offset between the time of a malicious node and the actual time. All the current time synchronization schemes [100, 101, 102, 103, 104] are vulnerable to this attack in one way or another.

We propose two approaches to detect and accommodate the delay attacks. Our first approach uses the generalized extreme studentized deviate (GESD) algorithm to detect the outliers introduced by malicious nodes. If there is no malicious

node, the time offsets among the sensor nodes should follow the same (or similar) distribution or pattern. For their attacks to be effective, malicious nodes typically report their time offsets much larger than those from the benign nodes, leaving their reported values suspicious. Our second approach uses a time transformation technique, which enables every node to derive an upper bound of the time offset that is acceptable to it, thereby filtering out the outliers. We discuss the merits as well as the limitations of each approach, and evaluate the effectiveness of these two schemes through extensive simulations.

The rest of the chapter is organized as follows. In Section 4.3, we identify and discuss a new attack called *delay attack*. Section 4.4 presents the system model and assumptions. In Section 4.5, we present the GESD-based approach. Section 4.6 presents the threshold-based approach. The performance of these two approaches are evaluated in Section 4.7. Section 4.8 concludes the chapter.

4.2 Previous Work

4.2.1 Time Synchronization in Hostile Environments

The time synchronization problem has been studied for years and many protocols have been proposed in the literature [100, 101, 102, 103, 104, 106, 107]. A thorough survey on this problem can be found in [108]. However, most of the aforementioned protocols become vulnerable in hostile environments. Taking the RBS scheme as an example, an attacker may launch different kinds of attacks to break the protocol. The first attack is called *masquerade attack*. Suppose a node A sends out a reference beacon to its two neighbors B and C. An attacker E can pretend to be B and exchange wrong time information with C, disrupting the time synchronization process between B and C. A second attack is called *replay attack*. Using the same scenario in the first attack, the attacker E can replay B's old timing packets, misleading C to be synchronized to a wrong time. A third attack is called *message manipulation attack*. In this attack, an attacker may drop, modify, or even forge the exchanged timing messages to interrupt the time synchronization process. For the message dropping attack, the attacker can selectively drop the packets and thus prolong the converging time of the synchronization process. This can be done

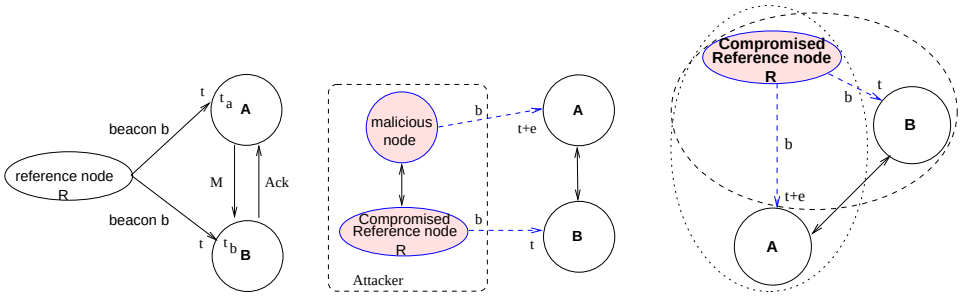
on a random or arbitrary basis, making it more difficult to be detected. For the message forging attack, the attacker can forge many reference beacon messages and flood the network. This not only incorrectly synchronizes the neighbors, but also causes those nodes to consume power to process these unwanted and faked timing messages. If some nodes run out of power, coverage holes or network partitions may appear.

We can certainly employ some cryptographic techniques to address the aforementioned attacks. For example, providing authentication of every exchanged message will prevent an outside attacker from impersonating other nodes or altering the content of an exchanged message. Adding a sequence number to beacon messages or other messages will prevent message replay attacks. Message dropping may be noticed by some misbehavior detection schemes [26].

4.2.2 Fault-Tolerance Time Synchronization

The previously proposed time synchronization schemes fall into the general field of fault-tolerance time synchronization, which has been studied for years in the past [109, 110, 111, 112, 113]. The algorithms mentioned in [110, 111] are based on an averaging process that involves reading the clocks of all the other processors. In [112], the nodes are assigned to one or more groups, then each node estimates the clock values of those nodes with which it shares a group. The algorithms in [109] work for arbitrary networks and can tolerate any number of processor or communication link failures as long as the correct processors remains connected by fault-free paths. The proposed scheme in [113] guarantees an upper bound of clock difference between any non-faulty nodes in a cluster, provided that the malicious nodes are no more than one third of the cluster.

Our proposed schemes differ from these schemes in several ways. First, in [110, 111, 112], it was assumed that two nonfaulty clocks never differ by more than a predefined threshold δ, but how to setup this threshold is not discussed. In our solution, we use the time transformation technique to derive the threshold and also give techniques to remove this assumption. Second, [109] requires an authentication mechanism such as digital signatures to ensure that no other node can generate the same message or alter the message without being detected. Our



(a) The RBS scheme (b) Collusion-based delay attack (c) Directional antenna based delay attack

Figure 4.1. The RBS scheme and the delay attacks

schemes do not have this requirement and can address delay attacks, which can not be handled by cryptographic techniques such as digital signatures, because nodes may be compromised. In addition, our scheme is complimentary to [113] and they can be used at the same time.

4.3 The Delay Attack Model

The time synchronization schemes proposed for wireless sensor networks are based on two models: the receiver-receiver model or the sender-receiver model. The reference broadcast synchronization scheme (RBS) [100] and its prototype protocol [114] falls into the receiver-receiver model. In the following, we simply use the RBS scheme to represent the receiver-receiver model. Schemes of the sender-receiver model include TPSN [101], LTS [102], and the tiny-sync and mini-sync schemes [103]. In the following, we will describe the delay attack model in the context of the RBS scheme [100].

The RBS scheme is based on a simple idea: using a third party for time synchronization. A node, which is a regular node acting as a *reference* node, broadcasts a reference beacon to its neighbors. Each neighboring node records the arrival time of the beacon based on its own clock. Since these receiving nodes are close to the reference node, we can assume the beacon arrives at both receivers at the same time. Therefore, the difference between the recording times of these receiving nodes is the time offset between them. By exchanging their recorded receiving times, they can calculate the clock offset, adjust and synchronize their clocks. As

shown in Fig. 4.1(a), nodes A and B have the recorded times t_a and t_b, respectively, and the time offset between them is $\delta = t_a - t_b$. To synchronize with node A, node B may increase its clock by δ, or both of them set their clocks to $(t_a + t_b)/2$. The *delay attack* is defined in Definition 2.

Definition 2 (Delay attack)**.** *The attacker deliberately delays some of the time messages, e.g., the beacon message in the RBS scheme, so as to fail the time synchronization process. We refer to this kind of attack as delay attack.*

Fig. 4.1(b) and (c) show two ways to launch the delay attack in the RBS scheme. In Fig. 4.1(b), two colluding nodes act as the reference node for nodes A and B. They send the reference beacon b to nodes A and B at different times. As a result, nodes A and B are deceived to believe that they receive the beacon at the same time, although they actually receive it at different times. Fig. 4.1(c) shows that a malicious node can launch the above attacks alone if it has a directed antenna [115] so that nodes A and B only hear one beacon message. The delay attack can also be launched when a benign node is synchronizing with a compromised node. The compromised node can add some delay to the beacon receiving time and send it the good node. This will mislead the good node to synchronize to a wrong time.

The sender-receiver model protocols [101, 102, 103] are also vulnerable to the delay attack. In the sender-receiver model, the sender and the receiver exchange time synchronization packets, estimate the round-trip transmission time between them, and synchronize their clocks after finding the clock offset between them. Since only two nodes are involved in the process, this model does not suffer from the attacks introduced by a malicious reference node. However, a node can be deceived if the node it is synchronizing with is malicious. Therefore, these schemes are also subject to the aforementioned delay attacks.

4.4 System Model and Assumptions

4.4.1 Node, Network, and Security Assumptions

We consider a sensor network composed of resource-constrained sensor nodes such as the current generation of Berkeley Mica motes [116]. Every sensor node is

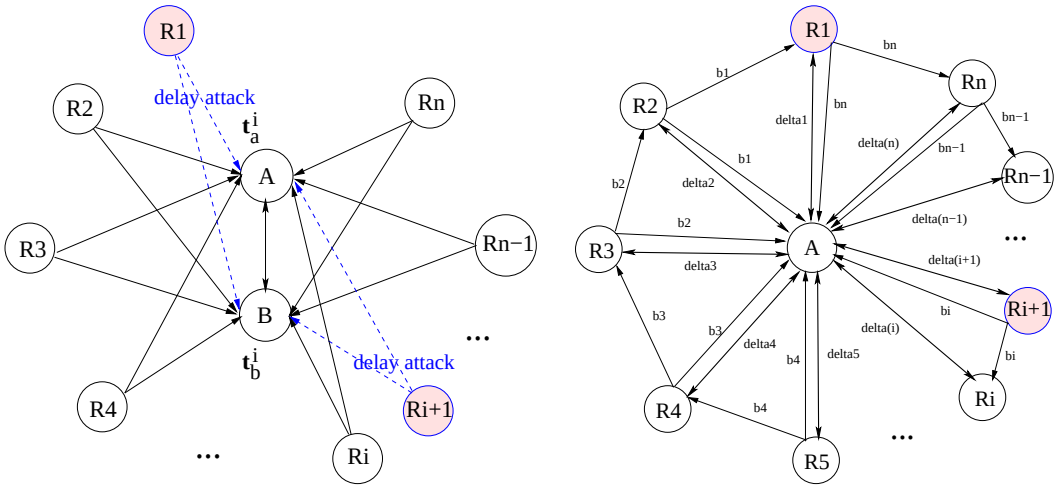
equipped with an oscillator assisted clock and powered by an external battery. The clock of a sensor starts to tick only after it is powered on. Since it is unlikely to power on all the sensor nodes at the same time, there may be large time offsets among sensor nodes initially. We assume that the sensor nodes deployed in a security critical environment is manufactured to sustain possible break-in attacks at least for a short time interval (say several seconds) when captured by an adversary [54]; otherwise, the adversary could easily compromise all the sensor nodes and then take over the network. To this end, we assume that there exists a lower bound on the time interval T_{min} that is necessary for an adversary to compromise a sensor node. We assume that the first time synchronization will be executed and finished within the time interval T_{min}. As a result, we can assume that all the sensor nodes are loosely synchronized.

Because of intrinsic clock drifts of sensor nodes, the time offsets among sensor nodes could become very large (e.g., in the order of seconds or even larger) unless time synchronization is performed once in a while. Hence, we assume that time synchronization is performed periodically. Clearly, the longer the time period, the larger the time offsets. We will discuss the selection of the appropriate synchronization interval in Section 4.7.3.4.

Each node is assigned a unique id before deployment and it can authenticate the messages sent/received with appropriate shared keys established through a key management protocol [54, 117, 118]. This ensures that no node can impersonate others during the exchange of timing messages and a malicious node can act as a reference at most once.

4.4.2 Models for Secure Time Synchronization

The general idea of defending against delay attacks is as follows. After collecting a set of time offsets from multiple involved nodes, we identify the malicious time offsets that are under delay attacks. The identified malicious time offsets will be excluded and the rest of the time offsets are used to estimate the actual time offset. Next, we present two models for collecting the time offsets: the two-node model and the neighboring-node model, which are described in the context of the RBS scheme.



(a) Two-node model **(b) Neighboring-node model**

Figure 4.2. Two models for secure time synchronization

The two-node model: In this model, one node needs to synchronize with another node. For example, in Fig. 4.2(a), node B is the cluster head and A is a node within the cluster. All nodes in the cluster are required to synchronize with B. Due to security concerns, node A only trusts the cluster head but not other nodes in the cluster. However, it has to use other nodes as reference nodes when using RBS. To deal with security attacks on time synchronization, node A uses multiple reference nodes to obtain a set of time offsets. For example, it can request R_1, R_2,... R_n to serve as reference nodes. Let $\langle t_a^i, t_b^i \rangle$ represent the two beacon receiving times obtained by using a reference node R_i and $\delta_i = (t_a^i - t_b^i)$ be the time offset between A and B. Node B obtains a set of n time offsets $\{\delta_1, \delta_2, \ldots, \delta_n\}$. Based on the collected time offsets, we can detect and exclude the malicious time offsets and obtain a more accurate estimation on the actual time offset between A and B.

The neighboring-node model: In some applications, a node may be required to synchronize with its neighbors to cooperate with each other. In this case, the two-node model is not enough since some neighbors may have been compromised and synchronizing with a malicious node is more vulnerable to attacks. Our solution is illustrated in Fig. 4.2(b). Suppose A has n neighbors: R_1, R_2, ..., R_n. We run the RBS scheme between A and each of its neighbors and each time we use a different node as reference to obtain a time offset. After collecting a set of n time

offsets, we can detect the outliers, exclude them, and make a good estimation on the actual time offsets.

In addition to the above two models, other models are possible too. However, all of them have one thing in common: they collect a set of time offsets, which may include the malicious time offsets. The focus of this chapter is to answer the following question: *Given a set of time offsets, how to identify the outliers and how to achieve an attack-resilient estimation?* In this chapter, we propose solutions in the context of RBS, although the solutions can also be applied to the sender-receiver based model.

4.5 The GESD-Based Delay Attack Detection

Intuitively, without delay attacks, the time offsets among nodes follow a similar distribution. The existence of delay attacks makes the malicious time offsets much different from the others; otherwise, the attack is not effective and can be tolerated by the time synchronization schemes. In statistics, these malicious time offsets are referred to as *outliers*, which is defined as "an observation which deviates so much from other observations as to arouse suspicious that it was generated by a different mechanism" [119]. Numerous schemes have been proposed to detect outliers [120] (see [120] for a survey). Among them, the generalized extreme studentized deviate many-outlier procedure (GESD procedure) [121] is proved to perform well under different conditions [120]. In the following, we introduce GESD and discuss how to apply it to our problem. After the outliers have been identified by GESD, we discuss how to exclude the outliers and obtain a more accurate estimation of the time offset.

4.5.1 The GESD Many-Outlier Detection Procedure

Before introducing GESD, let us first look at the extreme studentized deviate (ESD) test which is also called the Grubb's test. The ESD test is good at detecting one outlier in a random normal sample.

Definition 3 (ESD test)**.** *Given a data set* $\Gamma = \{x_1, x_2, \ldots, x_n\}$*, The* mean *of* Γ

is denoted as $\bar{x}$ *and the* standard deviation *of* Γ *is denoted as* s*. Let*

$$T_i = |x_i - \bar{x}|/s, \textit{where } i = 1, \ldots, n.$$

T_i *is also called the corresponding* T-value *of* x_i*. Let* x_j *be the observation that leads to the largest* $|x_i - \bar{x}|/s$*, where* $i = 1, \ldots, n$*. Then* x_j *is an* outlier *when* T_j *exceeds a tabled critical value* λ*.*

In principle, if T_j does not exceed the critical value λ, we need not single out x_j. Assuming this test finds an outlier, we then look for further outliers by removing observation x_j and repeating the process on the remaining $n - 1$ observations. However, the ESD test can only detect one outlier.

The GESD procedure is a modified version of the ESD test, which can find multiple outliers. Two critical parameters for GESD are r and λ_i, where r is the estimated number of outliers in the data set and λ_i is the two-sided $100 * \alpha$ percent critical value got from Eqn. 4.1.

$$\lambda_i = \frac{t_{n-i-1,p}(n-i)}{\sqrt{(n-i-1+t^2_{n-i-1,p})(n-i+1)}} \tag{4.1}$$

In Eqn. 4.1, $i = 1, \ldots, r$. $t_{v,p}$ is the $100 * p$ percentage point from the t distribution with v degrees of freedom, and $p = 1 - [\alpha/2(n-i+1)]$. Given α, n and r, the critical values λ_i, where $i = 1, \ldots, r$, can be calculated beforehand.

4.5.2 Using GESD for Delay Attack Detection

The GESD-based approach is formally defined as follows.

Definition 4 (GESD-based delay attack detection)**.** *Given the time offset set* $\Gamma = \{\delta_1, \delta_2, \ldots, \delta_n\}$*, all the time offsets* δ_i *that are identified as outliers by GESD are claimed to be under delay attack.*

In GESD, r is the number of estimated outliers in the data set, which is the estimated number of malicious time offsets in our settings. The choice of r plays an important role in GESD. If r is set to a small number and there are more than r malicious time offsets among the n time offsets, some of them cannot be detected

using GESD. On the other hand, if r is too large, it wastes time on checking the nodes that are in fact benign (good) ones. In our settings, since the number of time offsets is small (e.g., 20), we set r to be half of the total number of time offsets. We also assume that the number of malicious time offsets is less than half of the total number of time offsets. Without this assumption, GESD may not work since it may find the malicious time offsets to be benign and the benign ones to be malicious.

Definition 5 (Estimate r)**.** *Let the median of the time offset set* Γ *be* $\widehat{x}$ *and* s *be the standard deviation.* r *is defined as the number of time offsets* x_j *such that* $|x_j - \widehat{x}|/s > 2$, *where* $i = 1, \ldots, n$.

When the number of malicious nodes is small, i.e, less than 5% of the total, we can utilize the median of the time offsets to set r. As shown in Definition 5, r is the number of time offsets that are two standard deviations away from the median. In most cases, the data and time offsets are normally distributed, and then 95% of the values are at most two standard deviations away from the mean. In our case, we replace the mean with the median since the median serves better when there

Algorithm 1 Identifying outliers with GESD

input : r, Γ, λ
let j = 1, C and T be two arrays
loop
  calculate $\overline{x}$ and s over set Γ; find x_{k_j}
    which maximizes $|x_i - \bar{x}|, x_i \in \Gamma$;
  let $T[j] = \{|x_{k_j} - \bar{x}|/s\}, C[j] = x_{k_j}$;
    remove x_{k_j} from Γ;
  increase j; **decrease** r;
  if $r < 1$ **break**
end loop
let outlier set $\Omega = \emptyset, j = r$;
loop
  if $T[j] > \lambda^n[j]$ **then**
    $\Omega = \cup\{C[k]\}, \quad k = 1, \ldots, j$;
    return Ω
  else
    decrease j; **if** $j < 1$ **return** $\emptyset$
  end if
end loop

exists malicious data sets.

Algorithm 1 shows how to use GESD to identify outliers. The algorithm accepts three parameters: the estimated number of outliers r, the time offset data set Γ, and the critical value λ computed by Eqn. 4.1. λ can be pre-computed and stored in the sensors. In the following, we use λ^n to denote the critical values for a data set with n elements. Two array structures C and T, are used to save the candidate outlier information. C is used to keep the outliers and T is used to save the T value (Definition 3) corresponding to the candidate outliers. The T values of the candidate outliers are later used to compare with the critical values to decide whether the candidates are outliers or not.

Time complexity In GESD, two operations are time consuming: calculating the mean $\bar{x}$ and the standard deviation s. Among them, the most expensive operation is to calculate the standard deviation, which involves multiplications.

$$s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Given r and n, the first loop (Line 1-6) has $n + (n-1) + \ldots + (n-r) = nr - \frac{1}{2}r^2$ multiplications. In general, the time complexity of GESD is $O(nr)$. In the worst case, where $r = \frac{n}{2}$, the time complexity is $O(\frac{3}{8}n^2)$.

4.5.3 Delay Attack Accommodation

The goal of securing time synchronization is to synchronize the time in the presence of delay attacks. This can be achieved by first identifying the outliers (malicious time offsets) and then excluding them when estimating the true time offsets between nodes. We use the mean of the benign time offsets to approximate the true time offsets. The following definition can be used to approximate the time offset estimation $\hat{\delta}$.

Definition 6 (Estimate $\hat{\delta}$)**.** *Let Γ be the time offset data set and Ω be the outlier set. Then the benign time offset set is $\Gamma - \Omega$. $\hat{\delta}$ is defined as the mean of the set $\Gamma - \Omega$. Let the size of Γ be n and the size of Ω be k. $\hat{\delta}$ is calculated as follows.*

$$\hat{\delta} = \sum_{i=1}^{n-k} \frac{x_i}{n-k}, where\ x_i \in \Gamma - \Omega.$$

4.6 Threshold-Based Delay Attack Detection

One drawback of the GESD approach is that it needs to have enough reference nodes to detect the malicious nodes effectively. This has been verified by the simulation results shown in Section 4.7.2. In this section, we propose a threshold-based approach to detect the delay attacks based on the following observations. Without delay attacks, the time offset between two nodes should be bounded by a threshold value if the maximum clock drift rates can be bounded. With the threshold value, we can identify those time offsets that are larger than the threshold as malicious ones. Different from GESD, the threshold-based approach does not need that many reference nodes. Moreover, the threshold-based approach only needs to calculate the threshold once, and hence has less overhead.

In the following, we first present the time transformation technique and then present a method to determine the threshold based on the time transformation technique. After determining the threshold, we discuss how to use it to defend against delay attacks. Different from the previous work [122] where the time interval is used to order messages, we utilize the time interval to quantify the time offset upper bound between two nodes.

4.6.1 The Time Transformation Technique

Before presenting the time transformation technique, let us first look at the hardware oscillator assisted clock in Berkeley Mica motes [116], which implements an approximation $C(T)$ of the actual time T. $C(T) = k \int_{T_0}^{T} \omega(\eta) d_\eta + C(T_0)$ is a function of the real time T, which derives from the angular frequency $\omega(T)$ of the hardware oscillator. In this formula, k is a proportional coefficient and T_0 is the initial clock value.

For a perfect hardware clock, $\frac{d_C}{d_T}$ is equal to one. However, all hardware clocks are not perfect since they are subject to *clock drift*. We can only assume that the clock drift rate of the sensor clock does not exceed the maximum value of ρ. Thus, we have the following inequality: $1 - \rho \leq \frac{d_C}{d_T} \leq 1 + \rho$.

The idea of time transformation is to transform the real time difference Δ_T into

the sensor clock difference Δ_C and vice versa. These transformations are difficult because of the unpredictability of the sensor clock, but there exists some lower and upper bounds on the estimates. Based on the previous inequality, we can get: $1-\rho \leq \frac{\Delta_C}{\Delta_T} \leq 1+\rho$. This inequality can be transformed into $(1-\rho)\Delta_T \leq \Delta_C \leq (1+\rho)\Delta_T$ and $\frac{\Delta_C}{1+\rho} \leq \Delta_T \leq \frac{\Delta_C}{1-\rho}$, which means that the clock difference Δ_C can be approximated by the interval $[(1-\rho)\Delta_T, (1+\rho)\Delta_T]$. On the other hand, the real time difference Δ_T that corresponds to the sensor clock difference Δ_C can be approximated by the interval $[\frac{\Delta_C}{1+\rho}, \frac{\Delta_C}{1-\rho}]$.

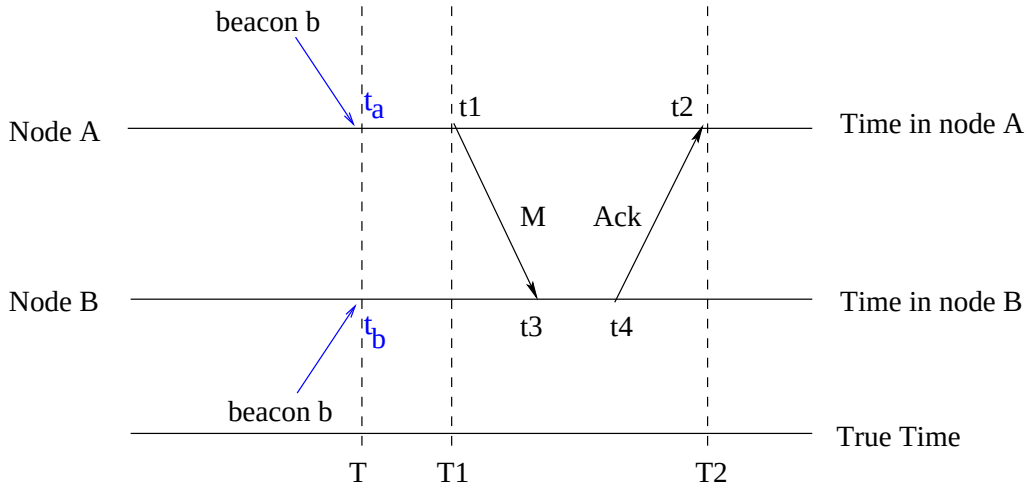


Figure 4.3. Time transformation

In order to transform a time difference Δ_{C_1} corresponding to one node N_1 with ρ_1, to a time corresponding to another node N_2 with ρ_2, Δ_{C_1} is first estimated by the real time interval $[\frac{\Delta_{C_1}}{1+\rho_1}, \frac{\Delta_{C_1}}{1-\rho_1}]$, which in turn is estimated by the sensor clock time interval $[\frac{1-\rho_2}{1+\rho_1}\Delta_{C_1}, \frac{1+\rho_2}{1-\rho_1}\Delta_{C_1}]$, relative to the local time of node N_2. As shown in Fig. 4.3, nodes A and B use RBS to do time synchronization. The maximum clock drift rates of A and B are denoted as ρ_a and ρ_b, respectively. Suppose A and B receive the reference beacon at time t_a and t_b, in terms of their own local clocks, respectively. After receiving the reference beacon, at time t_1, A sends a message M to B, telling B that it received the beacon at time t_a. Message M is received by B at time t_3, and then B sends back an Ack at time t_4 to confirm that it has received M. In the Ack, B piggybacks t_b, t_3, and t_4. After receiving the Ack, A can use the time transformation technique to transform the beacon receiving time t_b to a time interval $[t_{bL}, t_{bR}]$ relative to A's clock as follows.

$$\begin{aligned} t_{bL} &= t_2 - (t_4 - t_b)\frac{1+\rho_a}{1-\rho_b} - ((t_2 - t_1) - (t_4 - t_3)\frac{1-\rho_a}{1+\rho_b}) \\ t_{bR} &= t_2 - (t_4 - t_b)\frac{1-\rho_a}{1+\rho_b} \end{aligned} \tag{4.2}$$

4.6.2 Determining the threshold ξ

The threshold ξ is the upper bound of the time offsets between two nodes. We determine ξ based on the idea of time transformation shown above. A straightforward solution is to use $(t_{bR} - t_{bL})$ as ξ. However, $(t_{bR} - t_{bL})$ is a tight bound. If we use it to decide whether a time offset is malicious or not, it may identify benign time offsets as malicious. To effectively detect malicious time offsets, ξ should be a looser upper bound. Since t_{bL} and t_{bR} are the two boundaries of time t_b at node A, $max(|t_a - t_{bL}|, |t_{bR} - t_a|)$ should be the upper bound of the time offsets between A and B. Based on this observation, the time offset upper bound, ξ^{ab}, between A and B can be determined by Eqn. 4.3, which is a looser upper bound compared to $(t_{bR} - t_{bL})$. This can be explained as follows. If the clock drift rate of the two nodes are equal, t_a should fall inside $[t_{bL}, t_{bR}]$; otherwise, t_a may fall outside of $[t_{bL}, t_{bR}]$, leading to a looser upper bound based on Eqn. 4.3. Since the clock drift rates of two nodes are usually not equal, Eqn. 4.3 gives a looser upper bound compared to $(t_{bR} - t_{bL})$.

$$\xi^{ab} = \begin{cases} t_{bR} - t_a & \text{if } t_a < t_{bL} \\ \text{MAX}\{t_{bR} - t_a, t_a - t_{bL}\} & \text{if } t_a \in [t_{bL}, t_{bR}] \\ t_a - t_{bL} & \text{if } t_a > t_{bR} \end{cases} \tag{4.3}$$

The time offset upper bound between two neighboring nodes shown in Eqn. 4.3 is calculated only in the first time synchronization, which happens shortly after the sensor network deployment. Thus, the time offset caused by the clock drift is small in Eqn. 4.3. The clock drift time increases as time goes by. If the time synchronization interval is long, the clock drift time will be long and should be taken into consideration when determining the time offset upper bound.

Eqn. 4.4 gives the time offset upper bound between nodes A and B considering clock drift time.

$$\Delta^{ab} = \xi^{ab} + |\rho_a - \rho_b| \cdot T \tag{4.4}$$

In Eqn. 4.4, T is the time synchronization interval and Δ^{ab} is the upper bound

of the time offset between nodes A and B when they are synchronized using one reference node. To increase the accuracy of the estimation, we use n reference nodes to obtain a set of ξ^{ab}. The threshold ξ is defined as the maximum among them, as showed in Eqn. 4.5.

$$\xi = \text{MAX}\left\{\xi_i^{ab}\right\} + |\rho_a - \rho_b| \cdot T, \text{where } 1 \leq i \leq n. \tag{4.5}$$

With threshold ξ, we can detect malicious time offsets among a set of time offsets. The threshold-based approach is formally defined in Definition 7.

Definition 7 (Threshold-based delay attack detection)**.** *Given the time offset data set $\Gamma = \{\delta_1, \delta_2, \ldots, \delta_n\}$, all the time offsets bigger than ξ are claimed to be under delay attack and are identified as malicious time offsets.*

Time complexity Compared to GESD, the threshold-based approach involves two multiplications when calculating the interval $[t_{bL}, t_{bR}]$ in Eqn. 4.2. Given n reference nodes, the total number of multiplications are $2n$. Thus, the time complexity of the threshold-based approach is $O(2n)$, which is much less than that of the GESD approach, which is $O(nr)$. Further, the threshold is only calculated in the first time synchronization, but the GESD outlier detection algorithm is executed for each time synchronization. Thus, the GESD approach has much higher overhead than the threshold based approach.

4.6.3 Delay Attack Accommodation

After the malicious time offsets have been detected using the threshold, we can use the same strategy as that in Section 4.5.3 to exclude them and obtain a good estimation on the true time offset between two nodes.

4.7 Performance Evaluations

4.7.1 Simulation setup

We evaluate the performance of the two approaches using the reference broadcast synchronization (RBS) scheme by simulation. In the simulation, each node has a maximum clock drift rate at microsecond level (10^{-6} second) [122]. The deviations of clock drift rates among nodes are also at microsecond level. To synchronize two

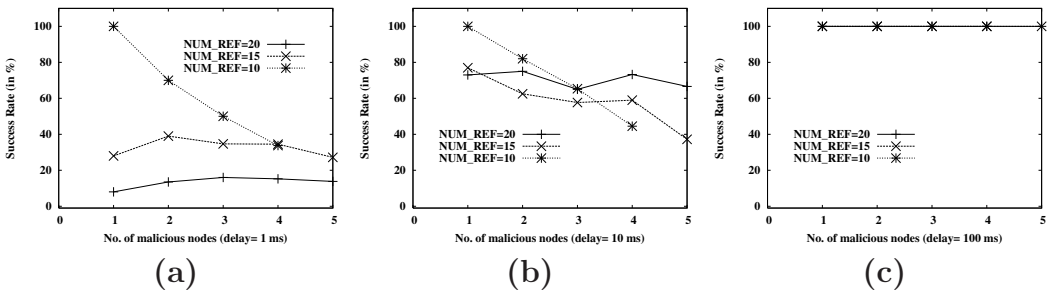


Figure 4.4. The successful detection rate of GESD

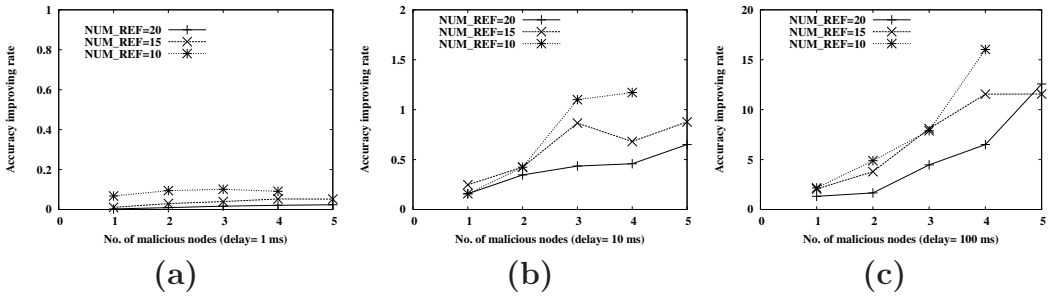


Figure 4.5. The accuracy improving rate of GESD

nodes, a number of reference nodes are generated varying from 10 to 20. Each reference node broadcasts a reference beacon to these two nodes, which record the beacon receiving times according to their clocks. The arrival times of the reference beacons follow Poisson distribution, and the beacon processing time follows normal distribution. Since the typical message size is 36 bytes in TinyOS [95], the beacon processing time is about 12 milliseconds which is the time required to process a 36-byte packet.

After a beacon has been processed, one node sends the beacon receiving time to the other, which calculates the time offset between them. After these two nodes get a set of time offsets, we randomly pick some of them as malicious time offset and assume they are under delay attacks. We also add a delay attack time which follows normal distribution. Based on a set of time offsets, the proposed schemes are evaluated with different levels of delay attack time and different number of malicious time offsets. All results are obtained by setting the synchronization interval to 5,000 seconds. The results are averaged over 100 runs. Most of the simulation parameters are listed in Table 4.1.

Table 4.1. Simulation parameters

Parameter	Value
Number of reference nodes	10 to 20
Number of malicious nodes	1 to 5
Beacon processing time mean (milliseconds)	12
Beacon arrival interval mean (milliseconds)	200
Clock drift rate mean (milliseconds)	0.005
Clock drift rate deviation (milliseconds)	0.001
Delay attack time (milliseconds)	1 – 100
Synchronization interval (seconds)	5,000

Three metrics are used to evaluate the effectiveness of the proposed schemes: the successful detection rate, the false positive rate, and the accuracy improving rate. In a network with delay attacks, the successful detection rate tells the percentage of malicious time offsets that can be successful detected. The false positive rate shows the percentage of time offsets that are reported as outliers but are not. The accuracy improving rate shows the accuracy improvement on the estimated time offset after the detected outliers have been excluded. Let $\hat{\delta}$ be the estimated time offset when the outliers have been excluded and δ_{bad} be the estimated time offset when the outliers have not been excluded. The accuracy improving rate is defined as $\frac{\delta_{bad}-\hat{\delta}}{\hat{\delta}} * 100\%$.

Finally, the synchronization interval is used to evaluate the impact of the synchronization interval on the successful detection rate of the threshold-based approach over different levels of delay attack time. The results are averaged over 100 runs.

4.7.2 Simulation Results of the GESD Approach

4.7.2.1 The Successful Detection Rate

Fig. 4.4 shows the successful detection rate as the delay attack time (*delay*), the number of malicious nodes, and the number of time offsets (NUM_REF) change. We did not show the successful detection rate when there are five malicious nodes and NUM_REF equals to 10, because GESD does not work when the number of malicious time offsets is equal or larger than that of the benign nodes.

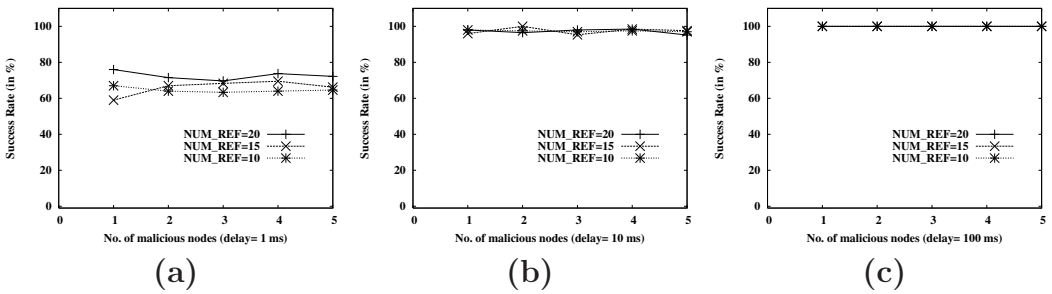


Figure 4.6. The successful detection rate of the threshold-based approach

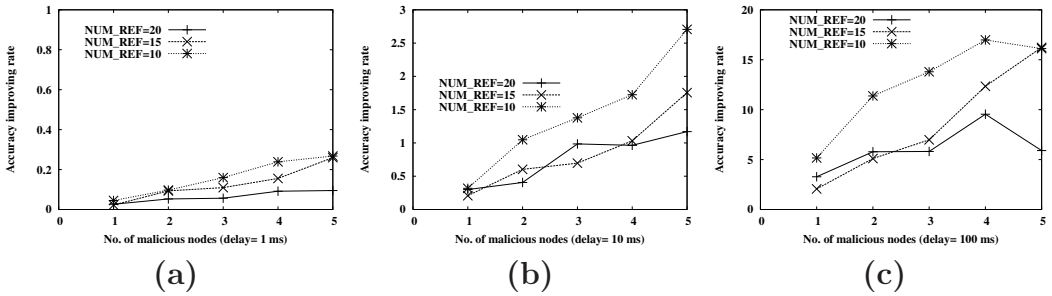


Figure 4.7. The accuracy improving rate of the threshold-based approach

Based on the figure, we have the following observations. First, when the delay attacks are at levels of 1ms 10ms, the successful detection rate is pretty low in most cases. Since the time synchronization interval is 5,000 seconds, the clock drift time between two nodes can be as large as 10ms. It is difficult to detect the delay attacks when the delay attack time is not significantly larger than the clock drift time, resulting in low successful detection rate. In Fig. 4.4(a), since the delay attack time is one level smaller than the clock drift time, increasing the number of reference nodes does not help improving the successful detection rate.

Second, as shown in Fig. 4.4(b), when the number of malicious time offsets increases, the successful detection rate decreases due to the masking problem in outlier detection. Masking occurs when an outlier goes undetected because of the presence of other outliers. GESD is not robust against the masking problem since it is based on the *mean* value, which is affected by the outliers. As an exception, when NUM_REF is 20 and the number of malicious time offsets increases from three to four, the successful detection rate increases. This can be explained as follows. If one malicious time offset is not detected in both cases, the successful detection

rate will be about 66% when the number of malicious time offsets is three and 75% when the number of malicious time offsets is four, which shows an increase in terms of successful detection rate.

Third, Fig. 4.4(b) also shows that the successful detection rate increases as the number of time offsets increase in general. Given a number of malicious time offsets, we will have more benign time offsets with a larger set of time offsets; and the more benign nodes we have, the higher the successful detection rate is. Thus, when there are multiple outliers, GESD is more effective if more time offsets are available.

Fourth, as long as the delay attack time is much larger than the clock drift during the synchronization interval, the successful detection rate increases dramatically. For example, as shown in Fig. 4.4(c), the successful detection rate reaches 100% when the delay attack is at 100ms level. As the delay attack time is larger than the clock drift time, the malicious time offsets can be easily identified. Although not shown in the figure, GESD keeps the 100% successful detection rate when the the delay attack time is larger than 100ms.

4.7.2.2 The False Positive Rate

The simulation results show that the false positive rate of GESD is almost 0 in our system settings. This is because a benign time offset will not be identified as outlier when there really exists malicious nodes. Thus, GESD works well in terms of false positive rate.

4.7.2.3 The Accuracy Improving Rate

Fig. 4.5 shows the accuracy improving rates with different level of delay attacks. From the figure, we can see that the accuracy improving rate is low when the delay attacks are at levels of 1ms and 10ms. This is because the delay attack time is relatively small compared to the clock drift time during the 5,000-second interval. Thus, excluding the malicious time offsets cannot have too much improvement. However, as the delay attack time increases, excluding the malicious time offsets can significantly improve the accuracy improvement rate. For example, when the delay attack time is 100ms, the accuracy improving rate can be increased by as much as 16 times (see Fig. 4.5c).

4.7.3 Simulation Results of the Threshold-based Approach

4.7.3.1 The Successful Detection Rate

Fig. 4.6 shows the successful detection rates with different level of delay attacks when the synchronization interval is 5,000-second. As shown in Fig. 4.6(a) and (b), when the delay attack time is 1ms, the threshold-based scheme can achieve a higher successful detection rate compared to GESD (Fig. 4.4ab). For example, when NUM_REF is 20, the successful detection rate of the THRESHOLD-based approach (80% on average) is seven times higher than that of GESD (10% on average). This shows that the threshold-based approach is effective even when the delay attack time is small compared to the clock drift rate. In the threshold-based approach, the threshold reflects both the maximum time offset that two nodes can have when there is no delay attack and the time offset caused by clock drift during the synchronization interval. Thus, even though delay attack time is not large compared to the clock drift time, it can still be detected at a high rate. Similar to GESD, the threshold-based approach achieves a 100% successful detection rate when the delay attack time is 100ms.

Fig. 4.6 also shows that the successful detection rate does not change too much as the number of malicious time offsets increases. Different from GESD, the threshold is affected neither by the outlier masking problem nor by the number of malicious time offsets.

In summary, the threshold-based approach can achieve a better successful detection rate than GESD. The threshold-based scheme performs well even when the delay attack time is small compared to the clock drift time and it is robust against multiple delay attacks.

4.7.3.2 The False Positive Rate

Simulation results show that the false positive rate of the threshold-based approach is always 0 in different settings. This is due to the reason that the threshold is determined in such a way that no benign time offsets will be identified as malicious. From the false positive rate point of view, both the GESD approach and the threshold-based approach perform well.

4.7.3.3 The Accuracy Improving Rate

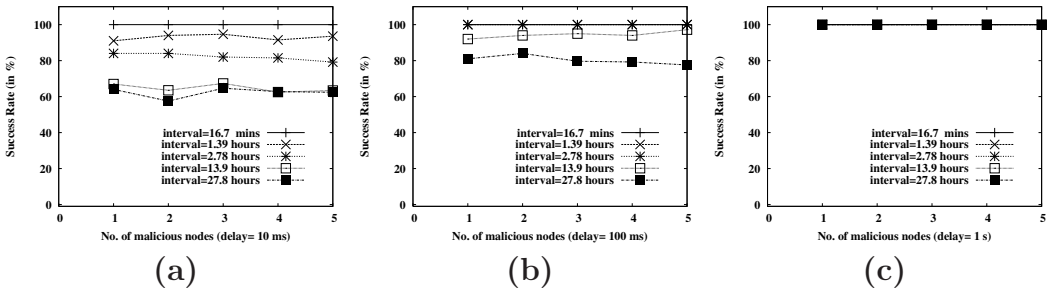


Figure 4.8. Delay attacks and the synchronization interval

Fig. 4.7 shows the accuracy improving rates with different level of delay attacks. As shown in Fig. 4.7(a), when the delay attack time is 1ms, the accuracy improving rate is below 30% most of time, because the delay attack time is small compared to the clock drift time. However, the accuracy improving rate achieved in the threshold-based approach is much higher than that of GESD. This can be explained by the fact that the threshold-based approach can achieve a much higher successful detection rate than GESD. As the delay attack time increases, the improvement on the accuracy also increases as shown in Fig. 4.7(b) and (c). In terms of the accuracy improving rate, the threshold-based approach performs better than GESD, which is consistent with the results of the successful detection rate.

4.7.3.4 The Synchronization Interval

Fig. 4.8 shows the impact of the synchronization interval on the successful detection rate under different level of delay attacks. The results are obtained when the number of time offsets is 10. As shown in the figure, given a certain level of delay attack, the successful detection rate decreases as the synchronization interval increases. For example, when the delay attack time is 10ms, the threshold-based approach can almost reach 100% detection rate when the synchronization interval is less than 3,000 seconds. When the synchronization interval is larger than 50,000 seconds (or 13.9 hours), the successful detection rates drops to about 60% on average.

Fig. 4.9 shows the tolerable synchronization interval with different levels of delay attacks. We define the *tolerable synchronization interval* as the maximum

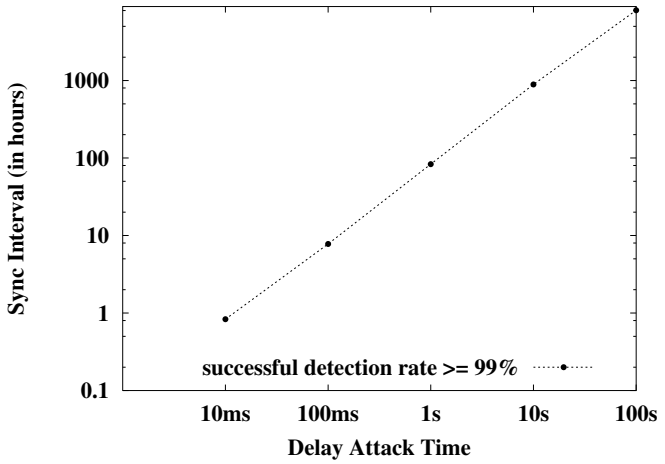


Figure 4.9. The tolerable synchronization interval

synchronization interval with which the threshold-based scheme achieves a 99% or higher successful detection rate. Here, we still use ten reference nodes. We observe that the tolerable synchronization interval increases as the delay attack time increases. Fig. 4.9 shows that the tolerable synchronization intervals are 0.83, 7.78, 83.33, 888.89, and 8055.56 hours when the delay attacks are 10ms, 100ms, 1s, 10s, and 100s respectively. Thus, given a delay attack time, we can select the appropriate synchronization interval without sacrificing the successful detection rate. Since the threshold-based approach can tolerate 83.33-hour synchronization interval on 1-second level delay attacks, we claim that our threshold-based scheme is very effective on defending against delay attacks in sensor networks.

4.8 Discussions

In this chapter, we identified various attacks that are effective to several representative time synchronization schemes, and focused on dealing with the delay attack. We proposed two solutions to detect and accommodate the delay attacks. Our first approach uses the generalized extreme studentized deviate (GESD) algorithm to detect multiple outliers introduced by the compromised nodes and our second approach uses a threshold derived using a time transformation technique to filter out the outliers. Extensive simulation results show that both schemes are effective in defending against delay attacks. However, the GESD approach needs more refer-

ence nodes to effectively detect the malicious nodes. The threshold based approach relaxes this assumption and outperforms GESD in terms of successful detection rate, false positive rate, and accuracy improving rate.

Chapter 5

A Sensor-network-based Vehicle Anti-Theft System

In the near future, sensors are anticipated to be produced in large quantities at a very low cost. They are also expected to be miniaturized into a cubic millimeter package (e.g., smart dust) in order to be stealthy in a hostile environment. On the other hand, practical, sensor-based applications need to be defined to stimulate the large-scale deployment of wireless sensor networks. In this chapter, we present a vehicle theft detection system using sensor networks, which could be a potential killer application for WSN. More specifically, we propose one count-based vehicle theft detection scheme and one signature-based scheme. This system is fully implemented and tested using the state-of-the-art sensor *Mica2 motes*.

5.1 Overview

Today, vehicles have been an essential part of our daily life. Unfortunately, we are also facing the high possibility of vehicle theft. For example, based on an article [123] published in USAToday, the National Insurance Crime Bureau reported that nearly 1.3 million vehicles were stolen in 2003. The vehicle theft rate held steady at about 433 cars stolen per 100,000 people in 2003.

Because of the high theft rate, vehicle tracking/alarming systems become more and more popular. Generally, these systems can be classified into three types: *lock devices*, *alarm systems*, and *vehicle tracking/recovery systems*. The commonly used lock device is the steering wheel lock. Although it is relatively cheap, it is

inconvenient to use and may be easily disarmed by skilled thieves. Car alarm systems (prices range $100 to $500) are very popular these days. However, the vast majority of blaring sirens are false alarms and people have been used to the alarms and do not care about them.

The commonly used vehicle tracking/recovery systems are based on radio signals such as the Lojack tracking system, the ProScout GPS Vehicle Tracking System, the TravelEyes2 Vehicle Tracking System and so on. After a vehicle has been stolen, the owner can report the problem to the police or the GPS tracking office. The wireless transmitter or the GPS device in the car will send wireless signals which can be picked up by the tracking device. The wireless signals can be used to pinpoint the location and lead police to rapid recovery. However, there are several disadvantages. First, these systems have high cost of $500 to $1300. Although the up-front purchase price keeps decreasing, the maintenance cost remains high. For example, these systems often come with a monthly monitoring fee. Second, GPS-based systems do not work indoors and terrain interference may occur in dense urban areas. Third, GPS-based tracking systems are easy to defeat since the thief knows where device is located. The thief can simply break off the antenna or cover it with metal, and then the GPS tracking system will become useless.

To address the limitations of existing vehicle tracking/alarming systems, we propose a Sensor-network-based Vehicle Anti-Theft System (SVATS). In SVATS, each vehicle is equipped with some sensors. All sensors in vehicles parked in the same parking area form a sensor network. For each parking area, one separate sensor network is formed and one base station (BS) is installed. SVATS relies on the sensors to detect vehicle theft and notifies the police through the BS. In this chapter, we will present the design, implementation, and evaluation of SAVTS.

The rest of the chapter is organized as follows. In Second 5.2, we give the system overview of SVATS. Section 5.3 presents the design of SVATS and techniques used in SVATS. In Section 5.4, we discuss implementation issues. Section 5.5 presents experimental results of SVATS. Section 5.6 concludes the chapter.

5.2 System Overview

Our system design is motivated by the requirements of a typical vehicle anti-theft application. The general objective of such an application is to alert the security

officer (or the vehicle owner) as soon as possible when a vehicle theft happens. To identify a vehicle theft requires that the application detects unauthorized vehicle move with high detection rate and low false positive rate.

5.2.1 SVATS Overview

In SVATS, each vehicle has a wireless sensor node which can be connected to the power source of the vehicle. All sensors in vehicles parked in the same parking area such as shopping centers, schools, hospitals, airports, residential areas, form a sensor network. For each parking area, one separate sensor network is formed and one base station (BS) is installed.

Within a sensor network, each node is monitored by its neighbors, which identify possible vehicle thefts by detecting unauthorized vehicle movement. When an unauthorized movement is detected, an alert will be reported to the BS, which in turn automatically sends a warning message to the security officer. The vehicle owner can also be notified at their choice.

In the following, we use an example to illustrate how SVATS works within a residential parking area. As shown in Fig. 5.1, suppose Emily comes back home and wants to park her car in the residential parking lot. Before she leaves the car, she powers on the sensor node in her car by a remote controller. The sensor node broadcasts an authenticated "join" message to sensors in the neighboring cars. After joining the network, it periodically broadcasts an authenticated "alive" message to its neighbors. Commanded by the remote controller, it sends an authenticated "leave" message to neighbors when the car leaves; the sensor is then turned off. If a thief moves the car, without sending an authenticated "leave" message, the neighboring sensors can detect the car movement. If the thief destroys the sensor in the car, the neighbors will not receive authenticated "alive" messages from the sensor, thus detecting the abnormal phenomenon. They will report the problem to the BS connected to the parking office which in turn automatically sends a warning message to the police.

The aforementioned basic SVATS system is enough to detect stolen vehicle. To track the stolen vehicle, we enhance SVATS by using the wireless nodes or access points deployed along major streets and around the intersections. Note that this roadside wireless access points may not be an extra requirement of SVATS. Many

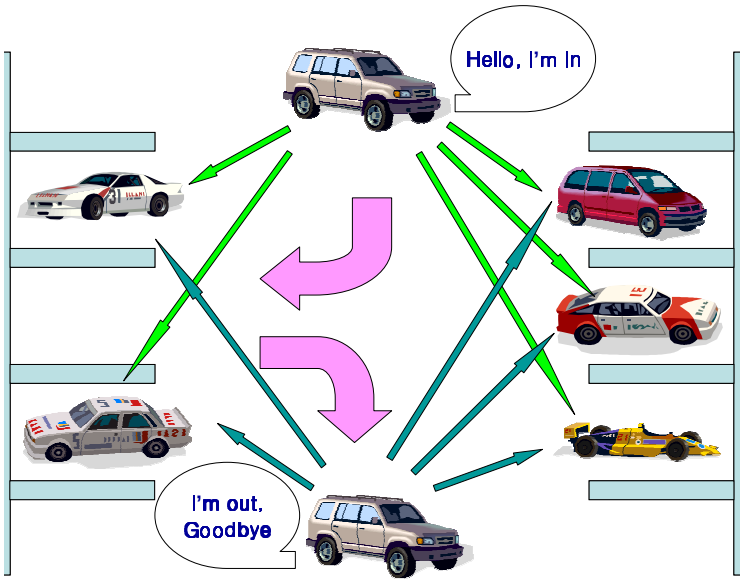


Figure 5.1. An example SVATS within a residential parking area

vehicular ad hoc networks [124, 125, 126] need to access this road side devices to improve road safety and support many commercial applications [127]. This roadside wireless access points can be used to communicate with the sensors within the passing-by vehicles. In case a car is stolen, the sensor node within the car can detect its own unauthorized movement by using movement sensors or by measuring neighboring car's sensor signal, and hence report problems to the roadside wireless devices. In this way, the vehicle can be tracked city-wide as long as it is within the area where SVATS system has been deployed.

Since the sensor is attached to the vehicle power, its position is known and may be destroyed by the thief, and then cannot report problems for tracking. To address this problem, we deploy more sensor nodes, referred to as the *slave sensors*, inside each vehicle. The slave sensors should be put at several hidden places inside the vehicle so that the thief cannot locate them in a short time. Their function is to monitor the status of the original sensor node, referred to as the *master sensor*. The slave nodes are normally in sleep. After the master sensor node sends "join" message, the slave nodes start to monitor the master sensor. After the master sensor sends "leave" message, the slave nodes go to sleep again. If the thief destroys the master sensor before it sends "leave" message, these slave nodes will report problems to roadside wireless devices that the vehicle is stolen. In case

the slave sensors cannot communicate with the roadside wireless device or such devices do not exist, the slave nodes can send the alert to sensors in passing by vehicles, which can carry the alert to a BS or directly send to the police through its vehicular network [127].

5.2.2 Why SVATS?

Compared with the traditional tracking-alarming systems, SVATS has the following advantages. First, unlike the tracking system where the transmitter must be powerful enough to send signal far away, sensors rely on multi-hop routing to send data. As a result, sensor nodes could be much cheaper. The state-of-art technology could eventually allow the price of a sensor node (e.g., smart dust) to be less than US $1 per unit [128]. Note that we do not rely on the advanced (hence expensive) sensing modalities such as GPS or accelerator, which come in a separate sensing board, but merely exploit the inherent capability of a sensor node such as radio for theft detection. Second, our proposed system is a decentralized scheme and the decision is made by a set of neighbors, which is more reliable. Third, recent advances in technology allow the size of a sensor node to be smaller than a cubic centimeter [6]. With such a small size, the slave sensors can be easily hidden inside the vehicle. Thus, it is hard for the thief to locate and compromise all the slave sensors in one vehicle within a short period of time. Therefore, it is extremely difficult for thieves to disable SVATS. This feature makes our scheme better than the GPS-based tracking system, where the anti-theft mission can be defeated easily by destroying the GPS device. Fourth, SVATS can directly and promptly notify the security office or vehicle owner regarding any possible vehicle theft via instant messaging techniques (such as SMS text messaging).

SVATS can be incrementally deployed. At the beginning, only a small percentage of the vehicles in a city may be equipped with SVATS sensor nodes. Although the effectiveness of SVATS is somehow limited initially, as long as most vehicles parked in the same parking area are equipped with SVATS nodes, SVATS will be effective within the parking area. Many rental offices can be the early deployers since the cost of SVATS is not high, and can be used as an attractive feature for the tenants. Further, the SVATS nodes can be used to replace the parking permit. As long as enough roadside wireless access points are deployed due to the emerging

of vehicular ad hoc networks [124, 125, 126], stolen vehicles can be tracked. This will motivate many people to install SVATS sensor nodes.

5.3 The SVATS System Design

To make SVATS work, we mainly address the following technical challenges:

Network Topology Management: Vehicles join/leave the parking lot frequently, and hence the network topology keeps changing. We rely on power control techniques to maintain a network topology so that a vehicle has enough neighbors to monitor it. Although the master sensor is connected to the power source, always using the maximum power level is not a good solution due to interference to other sensor nodes. We propose techniques to quickly reach the right power level.

Vehicle Theft Detection: By measuring the signal strength, sensors can defer the distance between two vehicles, and hence detect the unauthorized movement. We propose two techniques for vehicle theft detection: one is based on the missing of alive messages and the other is based on the change of measured distance between two vehicles.

Intra-vehicle Networking: When the master sensor is destroyed, the slave sensors have to report the problem and hence the stolen vehicle can be tracked. Since the slave sensors have to be hidden from the thief, they run by battery. We propose techniques to extend the battery life of these slave sensors and make sure that the slave sensors can detect and report problems when the master sensor is destroyed.

Alert Reporting and Warning Delivery: When a possible vehicle theft is detected, an alert message should be generated and forwarded to the BS. We propose techniques to let monitoring sensors send report to the BS.

Security management: The security of SVATS is critical for its success because the goal of an attacker will be to evade the detection of the system. We will need to deal with various attacks and secure the communications among sensor nodes.

In the following subsections, we describe our solutions to address these technical challenges.

5.3.1 Network Topology Management

In SVATS, sensors[1] are connected to the power source of the vehicle, and hence power consumption is not a major concern. However, since all the sensors share the same media to send messages, message collision is unavoidable. An effective way to mitigate collision is to reduce the signal transmission range by decreasing the transmission power level. On the other hand, if the transmission power level is too low, a vehicle may not be able to find enough neighbors to monitor itself. In this section, we use power control techniques to mitigate the effect of collisions and to maintain a required level of connectivity.

The goal of the network topology management (or power management) is to find the minimum required transmission power level at which the sensor in the vehicle has the required connectivity as well as the lowest possible interference to other nodes in a quick and efficient way.

We formulate the power management as a localized and distributed process. Each sensor checks its number of neighbors periodically (and adjusts its transmission power level if necessary) to maintain a desired number of neighbors with the minimum needed transmission power level. Specifically, it has three phases: initial power-level estimation, neighbor discovery and neighbor maintenance. The first two phases are executed when a node first joins the network; whereas the last phase is executed periodically after the sensor has joined the network.

Initial Power-level Estimation After joining the network, the sensor first estimates its transmission power level. Although using the estimated transmission power level cannot guarantee that the sensor will find enough number of neighbors, this phase can speed up the neighbor discovery process.

After a car has been parked, the sensor node inside the vehicle (e.g., node A) is triggered to power on, and it listens to and collects "alive" messages from neighboring nodes. After A has collected enough "alive" messages, it counts the number of neighbors that it can hear. In addition, A retrieves the transmission power levels from the "alive" messages and order them, from low to high, to form a power-level list. If the number of neighbors that it can hear is larger than the desired number of neighbors, the sensor can choose a power level that meets

[1]Sensors are actually master sensors in the paper unless specified otherwise.

the desired number of neighbors. Otherwise, the sensor should at least use the maximum power level of all received "alive" messages as the estimated transmission power level.

Neighbor Discovery With the estimated initial transmission power level, the joining node initiates the neighbor discovery phase by broadcasting a *join* packet to its neighbors. The *join* packet includes a neighbor list; i.e., the list of nodes that it can hear in the previous phase.

When a node broadcasts a *join*, some neighbors that are not in the neighbor list can still receive the *join* packet, which are referred to as *unidirectional neighbors*. The nodes that receive *join* will check whether they are within the neighbor list. If not, they just ignore the join request. Otherwise, they mark themselves as a neighbor node of the new joining node and send "reply".

If the joining sensor node can receive enough replies, the neighbor discovery process terminates. Otherwise, it has to find more neighbors. In this case, it has to increase its transmission power level and send a neighbor discovery message. This process is repeated until it either finds enough number of neighbors or reaches the maximum transmission power level. It may take a long time to find enough neighbors if the transmission power level is linearly increased. Using the Mica2 mote as an example, which has 255 power levels, the joining node may have to try 255 times to find enough neighbors in the worst case.

To reduce the neighbor discovery time, the joining node increases the transmission range using a strategy similar to binary-search, hoping to pinpoint the minimum needed power level quickly. Specifically, we want to use binary-search to find the power level within the range from the estimated transmission power level to the maximum power level.

Unfortunately, it is difficult to adjust the transmission range in a binary-search way for the current generation sensor nodes such as Mica2 motes. The reason is that the radio transmission power level (or signal strength RSSI) is measured by ADC counts (from 1 to 255) in stead of by dBm. Further, the relationship between transmission power level and transmission range is nonlinear.

Next, we derive the relationship between ADC count (ADC) and transmission range (d) (see Eqn. 5.6) to facilitate binary-search based on the *log normal shadowing model* [129]. This model is a statistical model for variations in the received

signal amplitude due to blockage. It combines the effect of both path loss and shadowing. In this model, the received signal power (P_r) is, in dB, given by

$$\left[\frac{P_r(d)}{P_r(d_0)}\right]_{dB} = -10\beta \log\Big(\frac{d}{d_0}\Big) + X_{dB}. \tag{5.1}$$

In Eqn. 5.1, β is called the path loss exponent, which is often between 2.7 to 5, and is usually empirically determined by field measurement. X_{dB} is a Gaussian random variable with zero mean and standard deviation σ_{dB}. σ_{dB} is called the shadowing deviation, which is between 4 to 12 outdoors, and is obtained by measurement. $P_r(d_0)$ can be calculated using Eqn. 5.2 [130].

$$P_r(d) = \frac{P_t \lambda^2}{(4\pi)^2 d^2}, \text{where } \lambda \text{ is the wavelength.} \tag{5.2}$$

From Eqn. 5.1, we can get the expression of d as

$$d = d_0 \cdot 10^{\frac{\frac{P_r(d)}{P_r(d_0)} - X_{dB}}{-10\beta}}. \tag{5.3}$$

On the other hand, for Mica2 motes with 915 MHz radio, the conversion from ADC counts to signal strength (RSSI) is given by

$$V_{RSSI} = V_{batt} \times ADC/1024 \tag{5.4}$$

$$RSSI(dBm) = -50.0 \times V_{RSSI} - 45.5 \tag{5.5}$$

Combining Eqn. 5.3, 5.4 and 5.5, we can get Eqn. 5.6, which shows the relationship between transmission range and ADC.

$$d = d_0 \cdot 10^{\frac{\frac{V_{batt} \cdot ADC + 932.75}{20.5 \cdot P_r(d_0)} + X_{dB}}{10\beta}}. \tag{5.6}$$

Let the maximum transmission range be D and the corresponding ADC counts be ADC_{max}. Let ρ be the fractions for binary search, i.e., $\frac{1}{2}$, $\frac{1}{4}$, $\frac{3}{4}$, $\frac{1}{8}$, $\frac{3}{8}$, and so on. From Eqn. 5.6, we get

$$\rho \cdot D = d_0 \cdot 10^{\frac{\frac{V_{batt} \cdot ADC_\rho + 932.75}{20.5 \cdot P_r(d_0)} + X_{dB}}{10\beta}}$$

$$= \rho \cdot d_0 \cdot 10^{\frac{\frac{V_{batt} \cdot ADC_{max} + 932.75}{20.5 \cdot P_r(d_0)} + X_{dB}}{10\beta}}. \quad (5.7)$$

Simplifying Eqn. 5.7, we get

$$ADC_\rho = ADC_{max} + \frac{10\beta \log\rho \cdot P_r(d_0)}{V_{batt}}. \quad (5.8)$$

Now suppose we need to jump to the transmission range of $\rho \cdot D$ in a binary search (i.e., ρ can be $\frac{1}{2}$, $\frac{1}{4}$, $\frac{3}{4}$, $\frac{1}{8}$, $\frac{3}{8}$, and so on), we can adjust the transmission power level accordingly following Eqn. 5.8.

If the joining node still cannot find enough neighbors after it increases its power level to maximum, it asks those unidirectional neighbors to join. In this round of join request, if a unidirectional neighbor is willing to serve as a monitoring node, it can notify the joining node. Since the unidirectional neighbor cannot send data directly to the joining node, it either increase its power level or ask other nodes to forward its reply.

Neighbor Maintenance To maintain a desired number of neighboring nodes, each sensor checks its number of neighbors periodically and adjusts its transmission power level if necessary.

5.3.2 Vehicle Theft Detection

In this section, we present two vehicle theft detection techniques: count-based and signature-based.

5.3.2.1 Count-based Theft Detection

After joining the sensor network, the sensor node keeps broadcasting "alive" message (also referred to as node advertisement) periodically. Each neighbor can measure the signal strength (or RSSI) of the received alive message. The RSSI is associated with the sending node's transmission power level and the distance between the sender and the receiver. If a monitoring node does not receive a certain number of node advertisement messages from the neighbor that it is monitoring, the monitoring node would assume that the neighbor has moved. The theft detection consists of three components: theft detection, theft attestation, and distributed voting.

Theft detection: For each monitee, a monitor keeps a counter that counts the number of node advertisements (NA) missed by that node. The count (NA) is increased by one when the advertisement timer runs out. When NA reaches a MAX_ADV_MISSES, the monitor suspects that the monitee as un-reachable and initiates the verification process.

Theft attestation: The attestation phase is necessary because situations other than theft could also result in the miss of node advertisement from the given monitee. For example, passing-by objects (e.g., vehicles or human beings) that temporarily block the communication path between the monitor and the monitee.

Using this component, the monitor can attest the vehicle theft and reduce the false alarm rate. The monitor either confirms or voids its detection based on the attestation results. More specifically, the monitor sends a challenge to the monitee and waits for a response. If the monitor does not receive a response from the monitee within a given timeout period, the monitor claims that the monitee is moved and a vehicle theft is detected. The challenge-response process can be executed several times. If none of the challenges gets through, the attestation confirms a vehicle theft detection; otherwise, the theft detection should be canceled.

The attestation at a single monitor node could still result in a false alarm due to the unreliability of the measurement method. Specifically, since the theft detection is based on RF signal which is subject to the surrounding environments and errors, a monitor could still make false detections if something has permanently blocked the communication between itself and the monitee. Thus, the detection should be confirmed by a set of nodes (i.e., through a voting process) and the confirmation procedure should be executed in a distributed fashion.

Using a centralized scheme such as cluster-head-based schemes to conduct the voting process may have problem. In the cluster-head-based voting scheme, the cluster head collects the detection reports from all the monitors and makes a final decision on the detection. However, if the cluster-head is compromised, the whole detection process fails. Further, a compromised cluster-head can falsely accuse any legitimate node in its cluster. Thus, we cannot depend on one node to make the final decision. Instead, the final-decision making process should be distributed.

Distributed voting: In this phase, every node that confirms a detection should broadcast a detection announcement to others. Based on the received distinct

detection announcements, each monitor makes a final decision on whether the monitee is stolen or not. In our system setting, if a monitor receives three or more detection announcements from different nodes, it claims that a vehicle theft detection is verified. In case that there are less than three monitors for a monitee, a monitor can still confirm a detection if it detects the theft continuously for several rounds (say three) of testing. After the detection has been verified, it sends an alert message to the BS. Note that the voting is distributed; i.e., the verification is done at each node independently. The distributed property avoids the security weakness of depending on a single node to make the final decision.

This count-based scheme has one disadvantage: The detection time is relatively long. The monitors start to miss node advertisements sent from the monitee only when the vehicle has moved out of the transmission range. Thus, the vehicle theft will not be detected until the vehicle has been moved for a distance of the sensor's transmission range, which could be as large as 70 meters for Mica2 motes. As a result, the response time for theft detection is long. Next, we present a signature-based technique that can detect vehicle theft much more quickly.

5.3.2.2 Signature-based Theft Detection

For the signature-based scheme, the movement detection is based on the change of measured RSSI[2] recorded at either the monitors or the monitee. Without vehicle theft (or movement), the monitee has an invariant signature. For example, its location, will not change and the distance measurements at each of its neighboring nodes should not change. Similarly, from the monitee's point of view, the distance measurements for each of its neighboring nodes should be the same. If the monitee is moved away from the original location without first sending a leave message, a monitor or the monitee can detect the theft by noticing the difference in the distance measurements. In other words, the distribution of the distance measurements before and after the vehicle theft is different.

More specifically, if a monitor (or monitee) makes *before* and *after* distance measurements on the monitee (or monitor), it can calculate the difference between

[2]Note that other localization techniques, such as time of arrival [131], time difference of arrival [132], and angle of arrival [133] can also be used here to measure the distance, and our scheme does not rely on a specific one.

the *before* and *after* distance-measurement pairs and determine whether the monitee has been moved or not. The *before* measurement, which consists of a set of distance measurements, is collected right after the vehicle is parked, and we denote the set of data in *before* measurement as the *location signature*. The *after* measurement is done at sometime thereafter and we denote the collected data set at this time as the *fingerprint*.

Several statistical techniques can be used to test/compare the distributions of two sets of data. In our system, we adopt the *unpaired observations* technique [134] to detect vehicle theft. Next, we first present the signature-based scheme at the monitors and then apply the same technique on the monitee to detect vehicle theft.

Theft Detection On The Monitor Side

In this scheme, a monitor first collects the location signatures for its monitee. To test whether a monitee is moved, the monitor collects a fingerprint and compares it with the location signature. If the two sets of data fail the test of unpaired observations, a vehicle theft is detected.

Specifically, our scheme has three phases: a *training phase*, a *detecting phase*, and a *verification phase*. The training phase starts shortly after the sensor (or vehicle) joins the sensor network and is executed only once; whereas the detection and verification phases should be executed periodically so as to detect the vehicle theft. The training phase collects the location signature, which corresponds to the *before* measurement in the unpaired observations; whereas the detection phase collects the fingerprints, which correspond to the *after* measurement. The verification is in a distributed way based on the results of other monitors. In the following, we describe the three phases in detail.

The Training Phase: In this phase, a monitee first discovers its monitors by broadcasting a *join* packet. A neighboring node receiving the join packet should send back a *join-reply* packet to the monitee. The monitee waits for a period of time until it receives enough number of join replies. After the monitee has received enough join replies, the monitor discovery process terminates. Otherwise, to discover more monitors, the monitee increases its transmission power level and sends another join message. This process is repeated until it either finds enough number of monitors or reaches the maximum transmission power level.

After discovering its monitoring nodes, the monitee broadcasts N beacon mes-

sages to them. Each monitor measures and records the measured distance of the N messages as a set $\{D_{i1}, D_{i2}, \cdots, D_{iN}\}$, which is the *location signature* for theft detection.

The Detecting Phase: After the training phase, a monitor (node i) collects a series of n distance measurements $\{d_{i1}, d_{i2}, \cdots, d_{in}\}$ for each of its monitees, respectively. Each set of data is denoted as a *fingerprint*. Then the unpaired observation technique is used to test the two sets of data (the location signature and the fingerprint) for vehicle theft detection. If they fail the test, the monitor claims that the monitee is stolen, since the difference between the two sets of data is significant. To reduce the false positive due to environmental dynamics and measurement errors, the monitor can collect multiple fingerprints and run the test several times before claiming the detection of a vehicle theft.

A monitor can make a decision on whether another vehicle has been moved or not using the detection technique presented above. However, the monitors may still arrive at wrong decisions and raise false alarms, because of measurement errors or malicious attacks. For example, if the measurement errors is due to external reasons (e.g., the moving of another vehicle between the monitor and the monitee), the monitor may still arrive at a wrong decision even if the distance measurements are correct. As another example, the attackers can forge data or replay old data to mislead a monitor to a wrong decision.

To mitigate this problem, we use the generalized extreme studentized deviate (GESD) algorithm [121] to pre-process the data before applying the unpaired observation scheme. The purpose is to detect and filter out the outliers introduced by measurement errors or attacks. To further decrease the false alarm rate, the detection phase is followed by a verification phase.

The Verification Phase: When a monitor detects a vehicle theft, it broadcasts its detection decision (or called *detection report*) to its neighbors using the maximum transmission power level. Upon the receipt of a detection report, each monitor (for the same monitee) broadcasts its detection report.

Each monitor collects the detection reports from other monitors. If a monitor can collect a threshold number of detection confirmations from others (or the majority of the received detection reports say that vehicle theft is detected), it claims that the monitee is stolen. Then the monitor can take further actions such

as reporting the event to the base station and so on.

Theft Detection On The Monitee Side

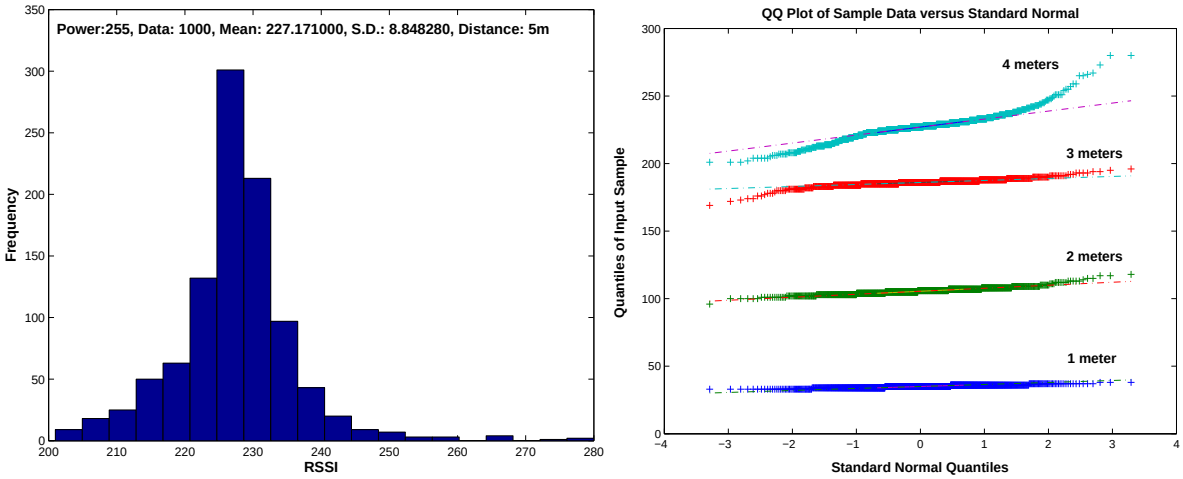
If the monitee has a movement measurement sensor, it can easily know whether there is an unauthorized vehicle movement. If such sensor does not exist, it can use techniques similar to the theft detection on the monitor side, which has three phases: the training phase, the detection phase, and the verification phase, except that the data measurement is done inversely. That is, instead of collecting the location signature and the fingerprint with regards to the monitee at the monitor side, these data are collected with regard to the monitors at the monitee side. The unpaired observation technique can be used here for theft detection. By noticing the distance changes to its neighbors, the monitee can indirectly detect its own movement (or theft).

However, since the master sensor is connected to the power source, it is easy to find and the thief can easily destroy it. That is why we rely on the neighboring monitors to detect vehicle theft instead of relying on the master sensor inside the stolen vehicle.

Distance Measurement Normality Test

In SVATS, we adopt the *unpaired observations* technique [134] to detect vehicle theft. This technique has been widely used in testing a hypothesis based on the difference between sample means. This technique, however, relies on the assumption that the observations come from a population that has normal distribution. To verify the normality of the distance measurements, we conducted an experiment using Mica2 motes. In the experiments, two sensors are placed 5 meters away from each other. One sensor broadcasts beacon signal periodically and the other node measures the received beacon RSSI. Fig. 5.2(a) shows the histogram of the measured RSSI, which indeed follows a normal distribution.

Fig. 5.2(b) illustrates the quantile-quantile (Q-Q) plot of RSSIs at different distances. Q-Q plot displays the sample quantile of RSSIs versus the theoretical quantile from a normal distribution. If the distribution of RSSIs is normal, the plot should be close to linear. From Fig. 5.2(b) we can see that the RSSIs for different distances are close to linear. Based on Fig. 5.2, we claim that the distribution of distance measurements follows a normal distribution.



(a) The histogram (b) The Q-Q plot

Figure 5.2. Test the normality of the distance measurements. One figure shows the histogram of the measured RSSI and the other one shows the quantile-quantile plot of the RSSIs at difference distances.

5.3.3 Intra-Vehicle Networking

Normally, master sensors can be used to communicate with roadside wireless access points for tracking the stolen vehicular. Since the sensor is attached to the vehicle power, its position is known and may be destroyed by the thief. At this time, slave sensors can be used to track the stolen vehicular. The slave nodes are normally in sleep. After the master sensor node sends "join" message, the slave nodes start to monitor the master sensor. After the master sensor sends "leave" message, the slave nodes will go to sleep again. If the thief destroys the master sensor before it sends "leave" message, these slave nodes will report problems to roadside wireless points that the vehicle is stolen. Since the slave sensors have to be hidden in some places hard to find, they have to run their own power. As a result, energy efficiency of these nodes is an important issue.

To extend the battery life, the slave sensors should keep sleep most of time. They also need to monitor the status of the master sensor in case the master sensor is destroyed. There are many existing work on designing sleep schedules to save power. However, SVATS is different from a typical sensor network, where sensors are used to regularly collect information from the deployed field. Instead of monitoring the external environment, the slaves monitor the beacon messages sent from the master sensor, and hence the slaves must be awaken when the master sensor

is sending messages such as alive, leave, join, etc. As a result, the sleep scheduling of the slave sensors should be synchronized to the master sensor's message sending behavior. Next, we present a sleep scheduling scheme for the slave sensors in the intra-vehicle network.

The sleep scheduling is designed based on two observations. First, to function properly, the slaves should be synchronized with the message sending of the master sensor. Second, the slave sensors should wake up at different times to reduce the theft detection delay and save power. To save power, the sleep-and-wake cycle of slave sensors is much longer than the beacon interval of the master sensor. If all slaves follow the same schedule, the detecting interval for slaves will be the sleep-and-wake cycle. On the other hand, if we can schedule the slaves to wake up evenly within the sleep-and-wake cycle, the detecting interval will be shortened linear to the number of slaves.

These observations lead to our two-phase scheme, consisting a bootstrap phase and a duty-cycling phase. In the bootstrap phase, all the slaves synchronize to the master sensor (in several respects such as message sending, time, etc.) and define their sleep scheduling. In the duty-cycling phase, every slave follows the sleep scheduling defined before to monitor the master accordingly.

Bootstrap Phase: Each time the master sensor turns on, it wakes up the slaves. Gu and Stankovic [135] proposes a remote radio triggered hardware, which extracts energy from specific radio signals without using an internal energy source, to provide wake up signals. This technique can be used here to wake up the slaves. Other remote wake up techniques can be used here as well. Note that the slave sensors will keep awake during the bootstrap phase.

After waking up, all slaves synchronize their clocks to that of the master sensor, using time synchronization protocol such as RBS [100]. Later on, the slaves can periodically synchronize to the master, since they will hear the beacon message at each sleep-and-wake cycle. Thus, we can assume that the slaves and the master have approximately synchronized clocks.

Then the master sensor announces its beacon interval I_b and the expected cycle interval N (in terms of number of beacon intervals). Based on I_b and N, each slave calculates its wakeup time utilizing a random number generator (RNG). More specifically, the slave first generates a random number based on its node

id, denoted as RNG(id); then its wakeup time within the cycle is calculated as: (RNG(id) mod N) $* I_b$.

The wakeup times assigned to all the slaves are not optimal due to the limitations of RNG. Some slaves may end up with the same wakeup time. To mitigate this problem, we should pick a good RNG and choose a prime number for N. Another reason for the sub-optimal sleep scheduling is due to a unique characteristic of slave sensors: They only passively listen and do not exchange packets with others. One reason of not exchanging packets with each other is to protect these slave sensors. Since they only passively listen during normal time, the thief cannot locate them based on electrical signal.

Besides wakeup time, we also need to decide how long the slave sensor should be awake in each cycle. For a sensor network consists of Mica2 motes, the time to send a typical packet of 36-byte is about 7.5 milliseconds, considering the date rate is 38.4 kbps. Thus, we set the duty time to be 10 milliseconds.

Since many slaves may be at sleep when the master sends the leave message, they will not be able to receive the leave message successfully. To solve this problem, the master sensor is required to send the leave message at the beacon interval (as sending the alive message) for a period of time (e.g., for two sleep-and-wake cycles). In this way, every slave will receive the leave message at least once. Note that, when a slave sensor receives a leave message, it will power itself off to save power.

Duty-cycling Phase: After calculating the wakeup time, each slave starts the duty-cycling phase based on its computed wakeup time.

Battery Lifetime: We investigate the average life time of a slave Mica2 sensor in intra-vehicle communication. According to Shnayder et al. [136], for a Mica2 sensor in receiving state, its power consumption $p_R = 10.2$ mA; for a sensor in sleeping sleeping state, its power consumption $p_S = 0.11$ mA. Two National AA-size batteries have a capacity of $C = 2 * 2870$ mAh, which fails to support the sensor when 85% of the power has been consumed. We know that in each cycle, the average power consumption for a slave sensor can be computed as

$$\bar{p}_{slave} = \frac{p_R * t_d + (I_b \cdot N - t_d) \cdot p_S}{I_b \cdot N}, \tag{5.9}$$

where $t_d = 10$ ms is the wakeup time during each cycle. Note that N also denotes

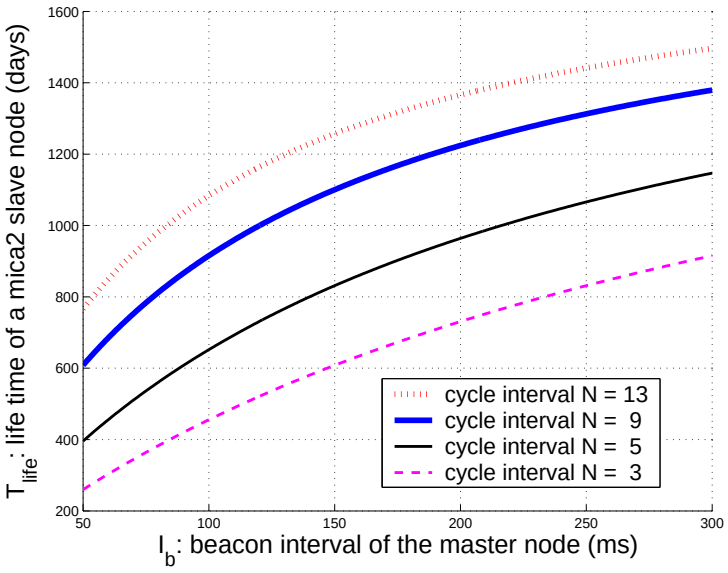


Figure 5.3. Battery life time for slave sensors

the number of slave senors. Then, we derive the average life time T_{life} for a Mica2 sensor as follows:

$$T_{life} = \frac{C \cdot I_b \cdot N \cdot 0.85}{p_R * t_d + (I_b \cdot N - t_d) \cdot p_S}. \tag{5.10}$$

Fig. 5.3 illustrates T_{life} as a function of the beacon interval I_b and the number of slave nodes N. The figure clearly demonstrates that the battery life time T_{life} becomes longer as the number of slave nodes increases. Each sensor can sleep longer when there are more slave nodes. Similarly, increasing the beacon interval I_b can extend the batter life time.

5.3.4 Alert Reporting

When a vehicle theft is detected, an alert should be sent back to the BS. For the sake of presentation, we refer to the nodes that detect a vehicle theft as *reporting nodes*. Note that the detecting node may also be the master sensor inside the stolen vehicle. If the reporting node can reach the BS, it will report the problem to the BS directly. In most cases, the detecting node may be multi-hops away from the BS due to the short wireless transmission range. Then, a routing path to the BS has to be found. Different from a general sensor network where the topology does not change frequently, our system is highly dynamic since cars may come and go frequently.

One way for sensors to set up routes back to the BS is to use a multi-hop routing protocol such as DSDV (destination sequence distance vector) [137], DSR (dynamic source routing) [138] and AODV (ad-hoc on demand distance vector routing) [139]. These protocols, however, are designed for general purpose networks where any node can route message to any other nodes. In our system, the report message is only sent to the BS, and hence the solution can be simplified. Based on these considerations, we adapt the distributed depth-first-search technique [140] to build a hierarchical topology, called *level tree*, and enhance it to handle frequent node joins and leaves. As we will see shortly, the level tree can also facilitate multi-path routing to improve the reliability.

The level tree is constructed as follows. The BS is at level 0 and it initiates the level-tree construction by broadcasting a discovery packet to its neighbors. On receiving the discovery packet, the BS's first-hop neighbors will assign themselves as level-1 nodes (one greater than the level they receive), and mark the BS as their *upstream node*. After setting their levels and upstream node, the level-1 nodes broadcast a discovery packet containing their own level. Similarly, the nodes that receive the discovery from level-1 nodes will assign themselves as level-2 nodes. In addition, since level-2 nodes can overhear the discovery packets from several different level-1 nodes, each level-2 node keeps a threshold number of level-1 nodes as its *upstream-node set*, which can be later used for multi-path forwarding. This process is executed recursively until all nodes get a level number and an upstream-node set. If a node does not have an assigned level (such as a joining node), it gets the level number by broadcasting a request packet to its neighbors and assign itself a level number which is one greater than the smallest level of the received packets and constructs an upstream-node set as well.

Using the level tree, a reporting node can send alert messages to the BS along the tree—from high level to low level. It first sends the alert message to a certain number of upstream nodes. After receiving the alert message, each upstream node will wait for a short, random period of time before forwarding the alert. During this period of time, by listening to its neighbors' transmission, each upstream node will forward the message to a node in its own upstream-node set to which no other neighbors have forwarded the alert message. Thus, in each round, the alert packet will be forwarded to a certain number of distinct upstream nodes. This process

is repeated until the alert messages arrive at the BS. Thus, an alert message is forwarded to the BS using highly likely multiple paths. Multiple-path forwarding makes our alert forwarding scheme robust and resilient to collisions and attacks such as message dropping and delaying attacks.

When an alert arrives at the BS, the BS should notify the security office. The owner of the vehicle can choose to provide personal information such as cellular phone number if he (or she) wants to be notified. If the owner does not want to provide personal information for privacy issues, he (or she) can still query the BS for the SVATS system to track the stolen car.

5.3.5 Security Management

The last component of SVATS is security management. Below we describe the security design, followed by security analysis.

5.3.5.1 Security Mechanisms

In SVATS, all the packets exchanged among nodes or with the BS are secured to prevent attacks. For all packets, authentication is required, whereas confidentiality may only be required for some types of packets. For example, routing control and keep alive information usually does not require confidentiality, whereas alert messages reported by sensor nodes may require confidentiality. As such, our first task is to determine the types of keys that will be used, which are in turn determined by the communication patterns in the system. In SVATS, the packets exchanged among sensors (including the BS) correspond to four communication models: hop-by-hop unicast (one-hop, one-to-one), end-to-end unicast (one-hop or multi-hop, one-to-one), local broadcast (one-hop, one-to-many), global broadcast (multi-hop, one-to-all).

We observe that no single keying mechanism can provide appropriate security and performance for all the above communications. Therefore, we adopt the key management framework from LEAP+ [141] to provide the following keys: *individual key*, shared between a node and the BS and used for secure communication between the node and the BS, *pairwise key*, shared between a node and a neighbor for securing the exchange of its decisions with its neighbors during a majority voting phase, *one-hop broadcast key*, shared by a node and all its neighbors and

used for authenticating keep-alive messages. Among these keys, individual key is preloaded into a sensor, and pairwise keys and one-hop broadcast key are established at run time.

How to distribute the keys to sensors nodes is an important issue in SVATS. It is trivial if SVATS is deployed in a small parking area for a department or a company because of the closed control. It will be challenge for a city-scale deployment. We may depend on a key distribution center (KDC)—the department of transportation (DOT) of each state is such a good candidate—to take the responsibility of preloading keys into sensors for sensor initialization. Other organizations who want to deploy SVATS at their owned or controlled parking areas can get sensors from the KDC.

5.3.5.2 Attack Analysis

Despite the variety of security attacks that might exist against SVATS, we consider the ultimate goal of a (rational) attacker is to evade its detection. In this spirit, we describe several general attacks against SVATS and show how SVATS address them.

For example, DoS attacks may be launched to jam the communication channel used by the sensor network. Under DoS attacks, the sensor nodes in vehicles will not be able to form a network, the network may be effectively disabled, or critical information such as alarm messages may be prevented from arriving at the BS. To survive under jamming-based DoS attacks, we may adopt the technique of service mapping in the JAM protocol [66] to locate the jammed area, and the nodes outside the jammed area then report to the BS, so that the network operator could take necessary actions to remove the attack. In the worst case, slave sensors can be used to detect/track the stolen vehicle, and report to roadside wireless access devices or sensors in other vehicles, which in turn report to police.

An attacker may deploy some sensors into the neighborhood of a victim car and let them become the monitors of the victim. These attack sensors will not report the theft event. For this attack to succeed, these attack sensors must be compromised nodes. Both our count-based and signature-based detection schemes rely on a threshold number of positive votes to decide whether to report the event or not. Therefore, despite the number of surrounding compromised nodes, vehicle

theft can be detected as long as the number of noncompromised monitors is above the threshold. Clearly, the choice of the threshold requires a tradeoff between detection accuracy and false alarm rate. Again, slave sensor can also be used to track the stolen vehicle to deal with this attack.

Another type of attack is message dropping and delaying attacks. When a malicious sensor node controlled by the attacker is part of a forwarding path between a victim vehicle and the BS, it may intentionally drop an alarm message to mask the event. It may also delay the forwarding of the alarm message so that the attacker has enough time to drive the vehicle far away before the BS (and the owner) is notified. The level-tree based routing adopted in SVATS essentially provides multiple paths; thus it is relatively robust against such attacks.

5.3.5.3 User Privacy

We note that there may also be privacy concerns. In SVATS, each vehicle is equipped with multiple sensors. Since the sensor's id is embedded in its message, an attacker may be able to track the location of the vehicle (and hence the owner). Users' privacy may also be violated by legal business companies. For example, since a user may park his car in various parking areas in a day; if all the BSs record the presence of the car and report this location information to a network controller (NC), the NC will be able to track the user's locations and infer his activities. Given the rich literature in privacy and anonymity research, we will identify the proper solutions for our application. Note that, the distributed nature of our solution provides a layer of protection for privacy.

5.4 SVATS Prototype Implementation

This section we report our prototype implementation of SVATS. As SVATS is a large system, we have not fully implemented all the details. Rather, as a practicability study, our prototype implementation is focused on the core operations from the networking perspective. Previously we showed security is critical for SVATS, but we can only integrate it into SVATS in the next phase of our project.

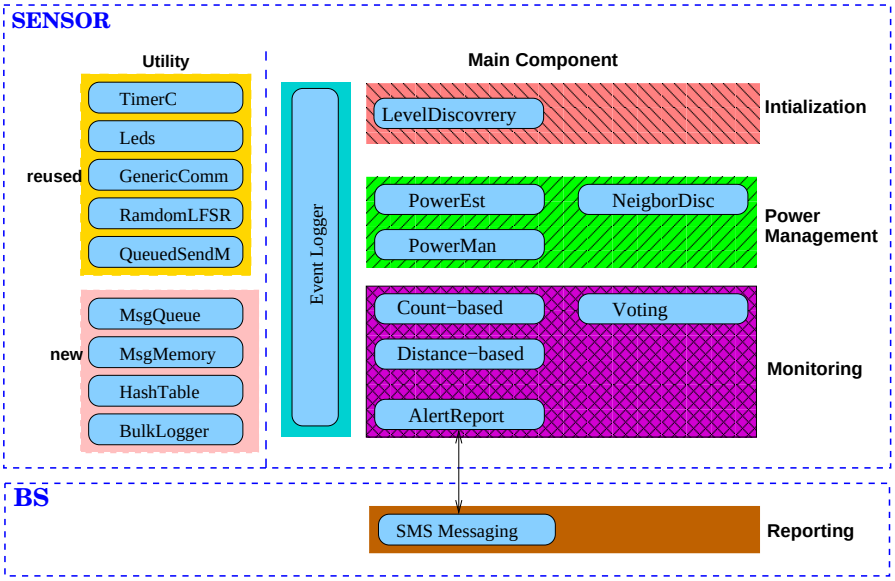


Figure 5.4. The components and interfaces used in implementing SVATS

5.4.1 Software Architecture

A prototype of SVATS was implemented using Mica2 motes [116] on the TinyOS platform [95]. Mica2 is widely available. Its radio chip (916MHz ChipCon) has low-power mode. It has programmable transmission power (255) levels and it provides direct sampling of the received signal strength.

Fig. 5.4 depicts the components and the interfaces used in our implementation. The system components are divided into six major groups: initialization, power management, monitoring, reporting, event logging, and general utilities. Initialization is responsible for setting up the level information of a node (on the level tree rooted at the BS). The power management component finds the optimal transmission power level for the master sensor. The monitoring component supports the vehicle theft detection and alert forwarding. SMS message module (only available at the BS) performs the function of sending warning messages to the security office or vehicle owner. The event logging component logs important events (such as node join time, leave time, theft detection time, etc.) into spacial EEPROM. These information is collected for experimental purpose. We also implemented some utilities (such as MsgQueue, HashTable, BulkLogger, etc.) which will be used by other five components.

The programs were written in nesC [96], a C-like programming language used

Table 5.1. The required RAM space as a function of the sampling size.

s	10	15	20
RAM (bytes)	2,733	3,123	3,613

for developing applications in TinyOS. The total code size is about 4,800 lines of nesC code. When the code is loaded into Mica2 mote, the code space is 25.4 KB (out of 128 KB) in ROM. The usage of data space in RAM (4 KB in total) depends on the sampling size s. Table 5.1 shows the usage of RAM as a function of the sampling size s.

5.4.2 System Design

5.4.2.1 Theft Detection

Fig. 5.5 illustrates a typical SVATS software's theft monitoring and detection control path, which includes three of the major components in Fig. 5.4. When power is on, a sensor first finds its level in the level tree by starting a self-discovery timer. If a level discovery packet is received during this time period, it cancels the timer, enters "Levelled" state, and sets its level as one plus the level in the received level discovery packet. Otherwise, if the timer fires, the sensor sends out a level discovery request. During this time, if it receives a level reply from another sensor or hears a level discovery packet directly, it enters "Levelled" state and sets its level as one plus the level within the received level reply message (or the level discovery packet). In addition, it starts to send alive message periodically.

The next part is the "Power Management" component, in which the levelled sensor starts a join timer. Before the timer expires, the sensor collects alive messages and estimates the power level by which it should use to send join message (i.e., neighbor discovery message). When the join timer expires, a join request is sent out. If it receives a join reply from others, it enters the "Joined" state.

The third part is the "Monitoring" component. A sensor enters the "Monitored" state when it receives enough number of join replies. Then, it starts a timer. Before this timer fires, the sensor keeps on collecting alive messages from its neighbors, and the RSSIs of these alive messages will be the signature for theft detection in the signature-based scheme. The sensor enters "Detection Mode" when the timer expires and a detection timer is started. In this mode, the number of missed alive messages will be counted in the count-based scheme, whereas the

received alive messages will be fed to the signature-based scheme, in which they serve as the fingerprint. When the detection timer fires, the fingerprint and the signature will be compared to detect possible theft. If a theft is detected by either scheme, a verification process is triggered. If the verification confirms a theft detection, the sensor starts a theft reporting process by sending alert messages to its upstream nodes. In turn, the upstream nodes forward the alert to their upstream nodes, and finally reaches the BS.

After the alert report, each sensor should remove the theft node from its neighbor list (nodes that monitor it) and monitor list (nodes that it monitors). As a result, the state of the sensor could be changed to either "Levelled", or "Joined", or still "Detection Mode", depending on the nodes' neighbor list after removing the theft node. This uncertain state is referred to as the "Adjusted State" in Fig. 5.5.

5.4.2.2 Event Logger

The original event logger provides line access (16 bytes) to EEPROM (512 Kbytes). In our case, an event is logged as a descriptive string, such as "at time i, node n is joined," "at time j, node n is monitored," etc., which may need multiple lines. To make the logger more efficient, we implemented a BulkLogger, which can divide a string into several lines automatically and write them into the EEPROM one by one. A high-level design of our event logger is shown in Fig. 5.6.

In our event logger, the EEPROM is divided into two parts: the application part and the event log part. The application part is reserved so that normal application can access the EEPROM without modification, enabling our event logger to be used in other applications as well. Depending on the application, the size of the application part varies. In SVATS, we set the application part to be 10 lines (i.e., 160 bytes). Note that the first two bytes of the event log keeps the lines of logged messages.

Read and write requests are connected to the BulkLoggerRead and BulkLoggerWrite interfaces through line converter. The function of the line converter is to convert the line number, which is used in the BulkLogger interfaces to the read line number in LogToRom. Line converter also validates the line number from different interfaces. The LogToRom component accesses EEPROM through the LogRead and LogWrite interfaces provided in TinyOS.

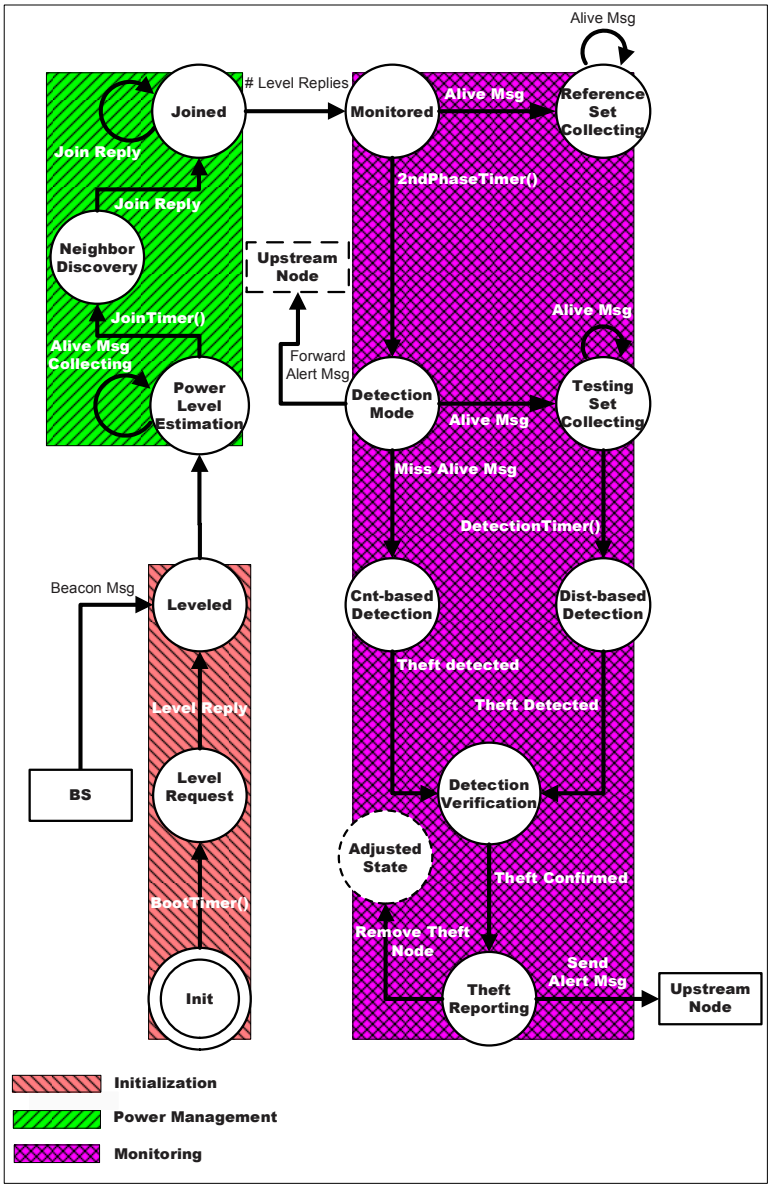


Figure 5.5. The control path of SVATS on a typical Mica2 mote. It illustrates three of the major components showed in the SENSOR part in Fig. 5.4.

5.4.3 Design Issues

When implementing the above architecture, we have to address several challenges:

Message Collisions: In our first few experiments, we notice that the count-based detection scheme has many false alarms. The investigation shows that the false alarms are caused by the collisions of alive messages among neighboring nodes. Due to collision, the neighboring nodes may miss the alive messages, and report

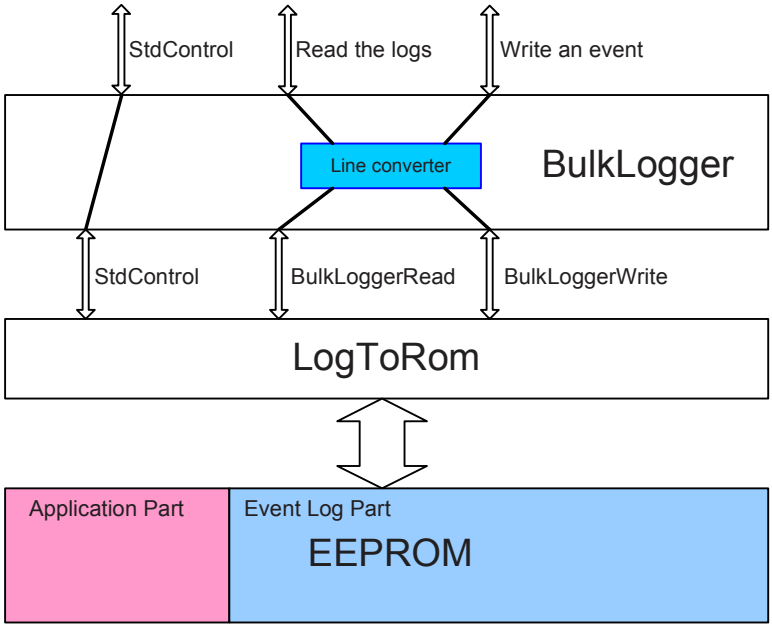


Figure 5.6. The event logger

theft detection. Several techniques can be used to mitigate the collision problem. First, when a node joins the network, it should observe the neighbors' alive message sending schedules and choose a different sending time to avoid collisions. The second technique is to add a random delay when sending an alive message. In this way, the sending of alive messages from different nodes will be most likely at different time, thus avoiding message collisions. This can help reduce other message collisions (e.g., leave message) as well. We adopted the second technique in SVATS.

Message Queuing: By default, TinyOS does not support packet buffering. If several packets arrive at the same node back to back, all but one packet will be dropped. This unexpected packet dropping will significantly affect the performance. We implemented a queuing technique to address this problem. Specifically, message queuing is used to queue the received packets or the packets to send at each node. This technique and the previous collision avoidance technique, together, ensure that our system works well in a message exchange based, mutual monitoring environment.

Time Synchronization: To analyze the events collected by the event loggers at different sensor nodes, logged events are time-stamped. Thus, time synchro-

Figure 5.7. A snapshot of the field test area

nization is needed. Since the detection time is in seconds, we don't need fine grained time synchronization protocol [100]. Instead, we designed a simple scheme for our coarse grained synchronization. The beacon message of the BS contains the number of beacons sent out so far. We refer it to as the *beaconClock*. The beaconClock information is also included in alive, level discovery, and join reply messages. When a sensor joins the network, it retrieves the current beaconClock from a beacon message from the BS, a level discovery message from its neighbors, or a join reply message. Each sensor maintains a *heartBeat* timer, which increases the beaconClock at the same beacon interval as the BS. To mitigate the clock drifting problem, whenever a node hears a beacon message from the BS or a level discovery message, it synchronizes its beaconClock to that contained in the message. In this way, all the sensors in the network maintain a beaconClock that is synchronized to the BS's.

5.5 Experimental Results

To evaluate the performance of SVATS, we developed a prototype based on Mica2 motes. The evaluation is done in a parting slot shown in Fig. 5.7. Various number of Mica2 motes are used in the experiments, one sensor node per vehicle. A laptop with a mote is used as the BS, which can send alert report, and can be used for message collection and analysis.

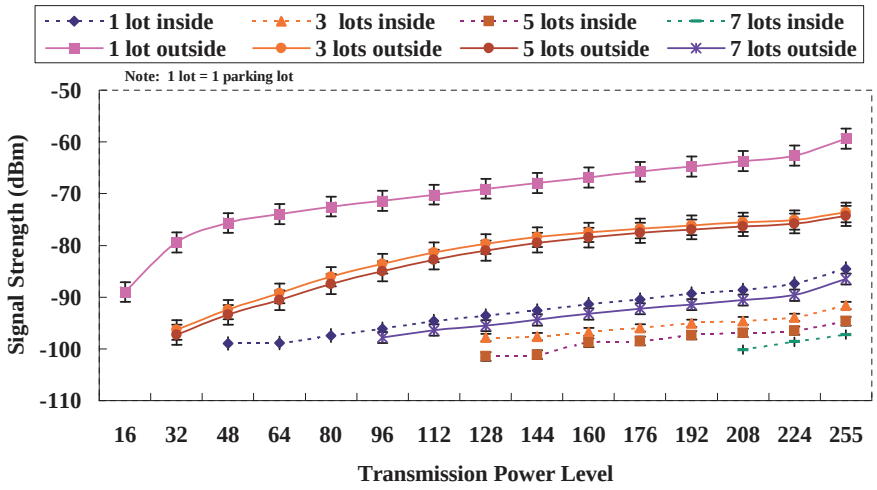


Figure 5.8. The relationship between transmission power level and signal strength, with different number of parking lots between. Sensors may be inside or outside of vehicles.

In the experiments, the BS sends beacons at an interval of one second and each sensor sends alive messages at the interval of 200 milliseconds. A sensor can monitor up to seven other nodes and should be monitored by at least three other sensors. The threshold for the missed number of alive messages (i.e., MAX_ADV_MISSES) in the count-based scheme is set to be six. Next, we present the experimental results.

5.5.1 Communication Characteristics of the Mica2 Radio

The communication range of a Mica2 mote depends on the environment. Since sensors are within vehicles, we want to measure the effect of vehicle body on the transmission range. We also measure the signal strength under different sending power levels with one sender and one receiver. Both vehicles are mid-sized cars. The distances between the sender and receiver are from 1 parking lot to 7 parking lots. We measure and compare the signal strength when the sensors are put inside and outside of the vehicle. Fig. 5.8 only shows the results for distances of 1, 3, 5, 7 parking lots so that the figure is readable. The first observation is that, as expected, the signal strength decreases as the number of parking lots between them increases. We can also clearly see that the signal strength is much lower when sensors are inside the vehicles.

Fig. 5.9 shows nine scenarios (scenario A to I) when a vehicle is stolen and

driven away and Fig. 5.10 shows the signal strength captured at the Guardian node when a vehicle is stolen from parking lot C (i.e., scenario C). Different transmission power levels are used for monitoring purpose. Note that the theft starts at time 20 (seconds).

Using stopwatch, we find that it often takes 10 seconds for the vehicle to be driven out of the range of the Guardian node. Fig. 5.10 confirms that the signal strength starts to change dramatically at about time 28.8 (seconds), at which the vehicle starts to get out of the range of the Guardian node. We call this time period (between theft starting and the time when vehicle is out of range) the *detection window*. The count-based scheme detects the theft at the end of the detection window whereas the signature-based scheme can detect the theft within the detection window. Fig. 5.11 shows the signal strength captured at the Guardian node for different vehicle theft scenarios, from A to I, sampling at rate of 1 per 3.2 seconds. We can see that the detection window is about 12 seconds for all scenarios.

To illustrate the change of signal strength during the theft process more clearly, we conducted another experiment using the same scenarios as in Fig. 5.9, in which only power level 255 is used and the sampling rate is 5 per second. Fig. 5.12 shows the results for three of the scenarios. This figure clearly shows that the signal strength is relative stable before theft and changes greatly when theft happens.

Note that other factors such as the length of the antenna, the elevation above the ground, and object (e.g., people, vehicles) blocking may also affect the measured signal strength. Although the absolute values may vary in different environments, we can still draw similar general observations about Mica2.

5.5.2 The Effectiveness of the Detection Schemes

In this section, we evaluate the count-based and signature-based schemes to quantify the successful detection rate, the false positive rate, and the detection delay. We deploy 6 sensors in 6 passenger cars and choose sensor 4 to be stolen. We drive vehicle 4 away without sending leave message. We evaluate the performance using three different deployment scenarios and three of them are reported below.

Experiment 1: Fig. 5.13 shows the sensor deployment, where vehicle 4 is monitored by five other sensors. At time 35.5 seconds, we drive away vehicle 4

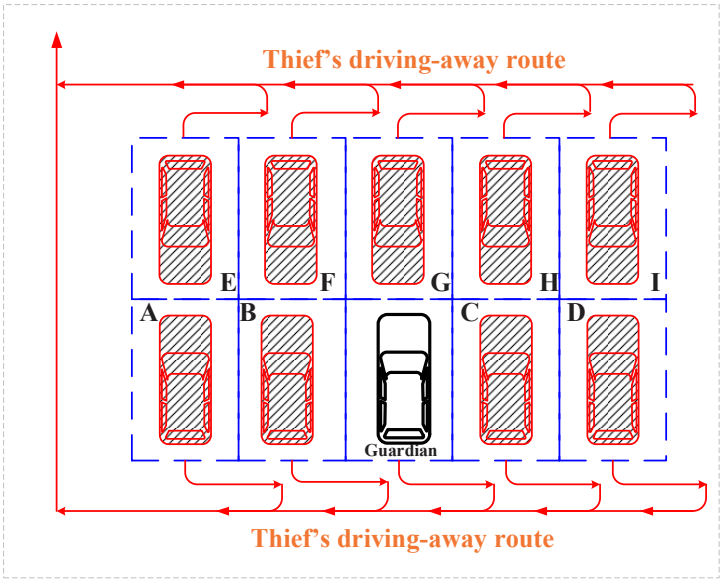


Figure 5.9. Nine scenarios (from scenario A to I) when a vehicle is stolen and driven away from node "Guardian."

(without sending a leave message). Fig. 5.14 shows that the detection delay is between 4 to 9 seconds for the signature-based scheme and is between 13 to 27 seconds for the count-based scheme. The results show that the signature-based scheme is pretty effective.

The false positives of the two detection schemes are shown in Fig. 5.15 and Fig. 5.16, respectively. The y axis shows the node id and the x axis shows the time. Each point in the figure shows the detection of movement by another node (some are false detections). For example, in Fig 5.15, node 3 detects that node 1 is moved at time 40, and hence a cross representing node 3 is plotted near (40, 1) Note that the cross representing node 3 is put near $y = 1$ so that several detections from different nodes can be accommodated.

The results show that both signature-based and count-based schemes have no false positives. Note that a false positive occurs only when three or more nodes confirm a detection within the same detection period. In the signature-based scheme, some node (i.e., node 1) has been detected being moved, but actually not. For example, node 1 was detected as moved by node 3 at time 40. However, due to the use of verification, no false positives are raised against node 1 since there are not enough neighbors to confirm the movement of node 1.

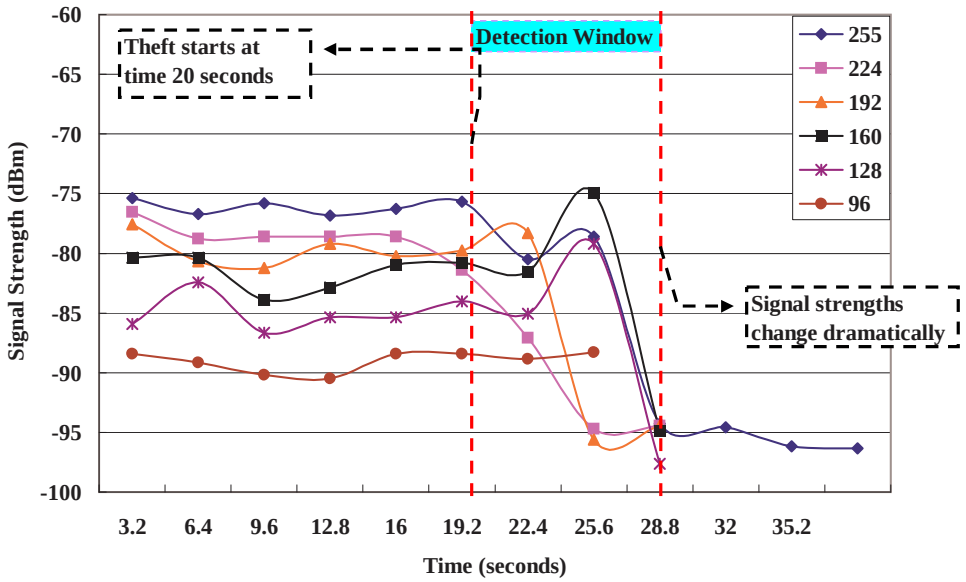


Figure 5.10. The signal strengths measured at node "Guardian" for scenario A, with different transmission power levels.

Experiment 2: In experiment 1, the sensors are in the same row. In this experiments, the vehicles are parked in two adjacent rows and head to head, as shown in Fig. 5.17. The theft starts at 85.3 seconds. The detection delay results are shown in Fig. 5.18. The detection time is about 8 seconds on average for the signature-based scheme and 15 seconds for the count-based scheme. As in the first experiment, there is no false positive, as shown in Fig. 5.19 and Fig. 5.20. Further, there is no false accusation in both schemes before the verification phase. This implies that having the monitoring nodes at two adjacent rows instead of in the same row may achieve better monitoring and detection effects. This observations is confirmed by the next experiment.

Experiment 3: In this experiment (Fig. 5.21), the monitoring vehicles are not adjacent to each other. The theft starting time is 53 seconds. The detection delay results are shown in Fig. 5.22. The results show that the signature-based scheme can detect the theft effectively (all five nodes detect node 4's theft within 5 seconds). The count-based scheme has a much longer detection delay (22 seconds on average). Both of them have no false positives as well (see Fig. 5.23 and Fig. 5.24.

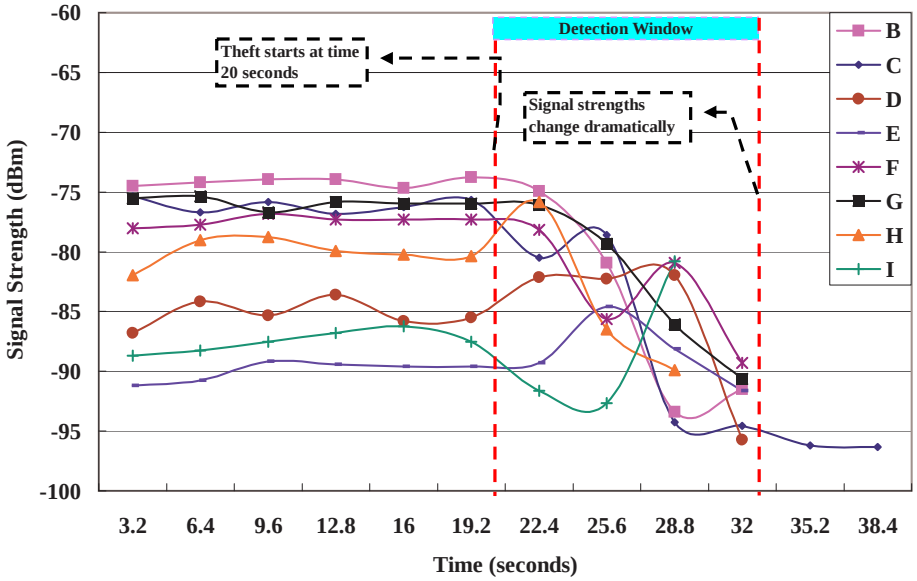


Figure 5.11. The signal strengths measured at node "Guardian" for scenarios A to I, where the transmission power level is 255 and the sampling rate is 1 per 3.2 seconds.

5.5.3 Discussions

Current SVATS still has several limitations. The biggest one is caused by network partitions in a sparse parking lot. In the extreme case no neighbors can be found even if a sensor has tried its maximum power level. This is not a concern if the sensor can communicate with the BS directly. For a large parking space, this may not always hold. As the result, this vehicle cannot receive any protection.

One possible solution is to deploy extra sensors inside the parking area securely (or invisibly), which can be used to monitor the parked vehicles and forward alert messages. However, the cost of the system will be increased. Note that even when there are not enough neighbors, the slave sensors inside the stolen vehicle can still provide tracking service as long as there are existing road side wireless nodes. Also, the slave sensors can send the alert to sensors in passing by vehicles, which can carry the alert to a BS or directly send to the police through its vehicular network.

So far, we have experimented with SVATS in a very restricted environment with a small number of sensors. We have not got enough resources to test SVATS in a large parking area with hundreds of vehicles. The road-side detection component is neither examined although we do not expect much complication.

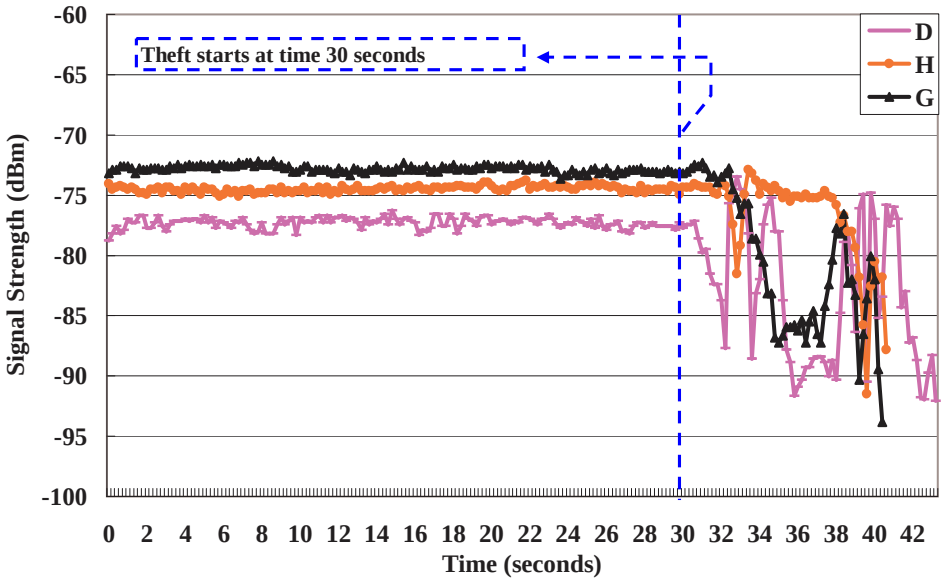


Figure 5.12. The signal strengths measured at node "Guardian" for scenarios D, H, and G, where the transmission power level is 255 and the sampling rate is 5 per second.

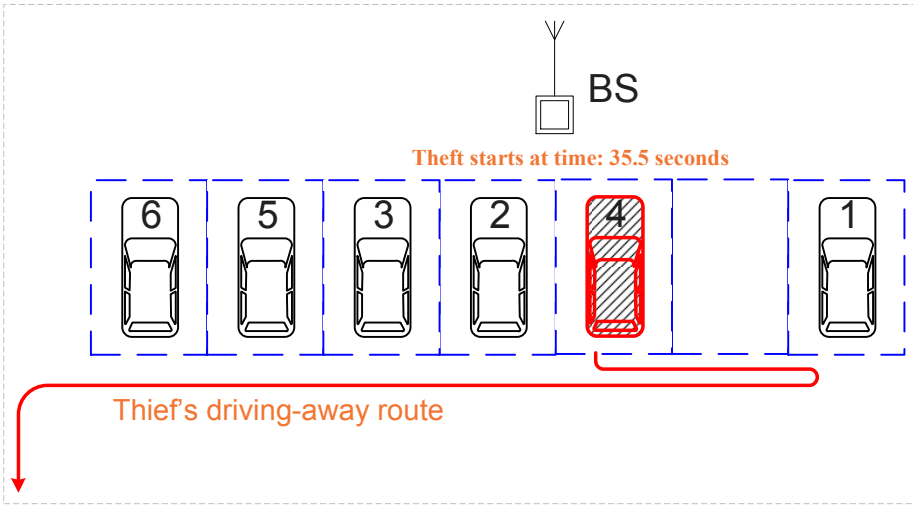


Figure 5.13. Experiment 1

5.6 Discussions

This chapter presented a sensor-network-based vehicle anti-theft system (SVATS), which can detect unauthorized vehicle movement and track the stolen vehicle. SVATS is a large system; clearly, its success will require more than techniques. In this chapter, we focused on the technical issues specific to the system such as connectivity management, movement detection, intra-vehicle networking, alert

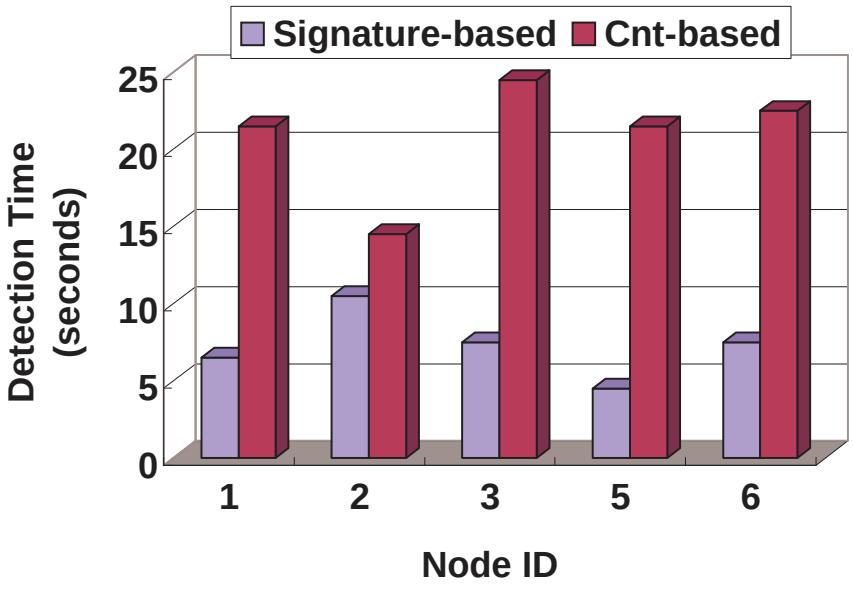


Figure 5.14. Detection delays (for experiment 1)

reporting and security management. A prototype has been deployed to test the design. Experimental results show that SVATS can detect vehicle theft effectively. Results also show that the signature-based scheme has much shorter detection delay compared to the count-based scheme.

SVATS can be deployed incrementally. Its nice features such as rapid response and resilience to attacks will motivate many people to install SVATS sensor nodes. If successfully deployed, SVATS will become the largest practical sensor network application, and will provide a platform for testing many of the techniques proposed for wireless sensor networks.

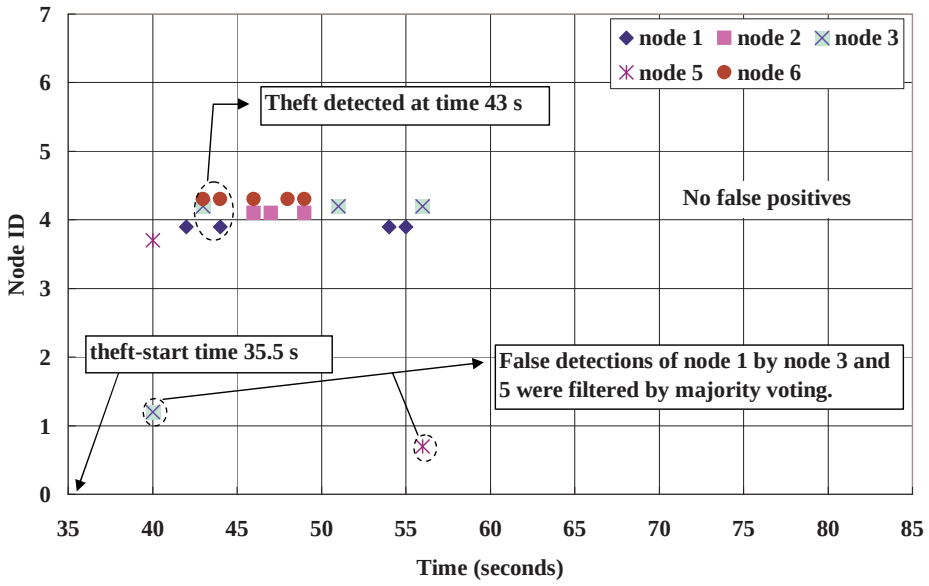


Figure 5.15. False positives of signature-based scheme (for experiment 1)

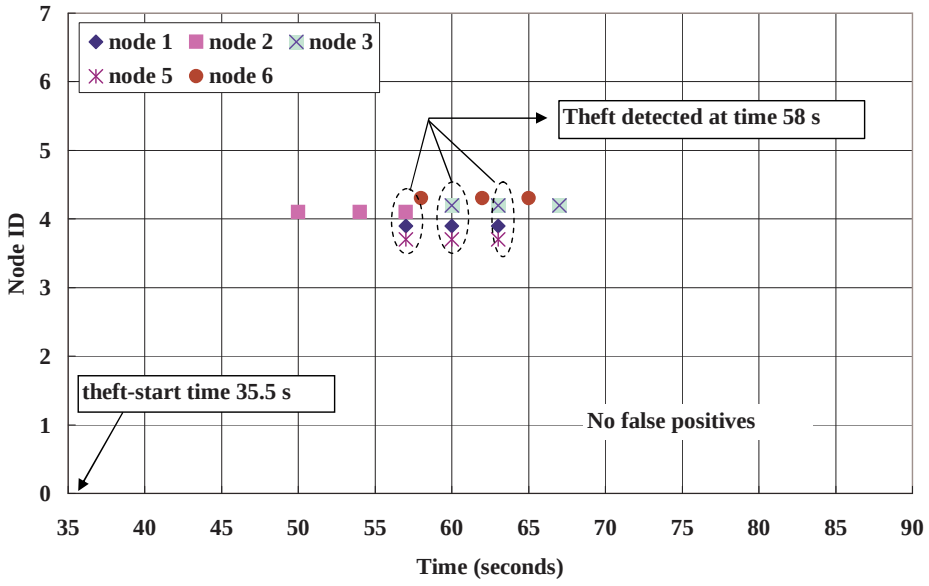


Figure 5.16. False positives of count-based scheme (for experiment 1)

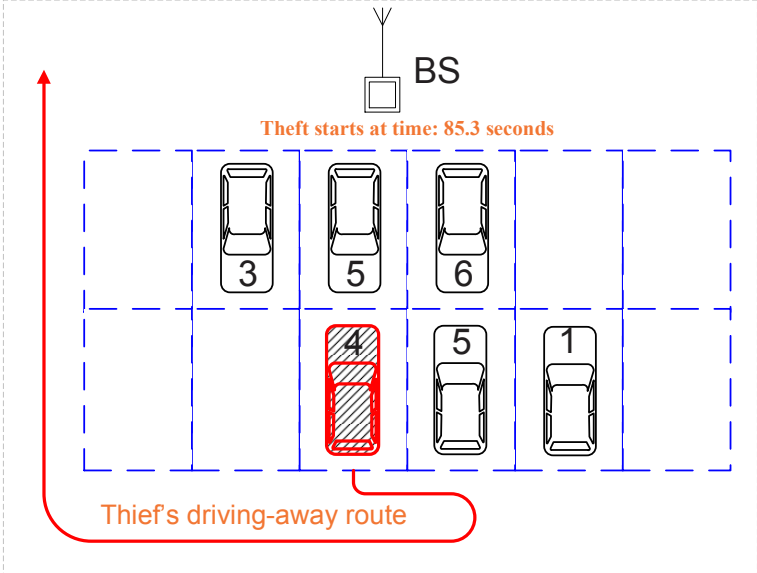


Figure 5.17. Experiment 2

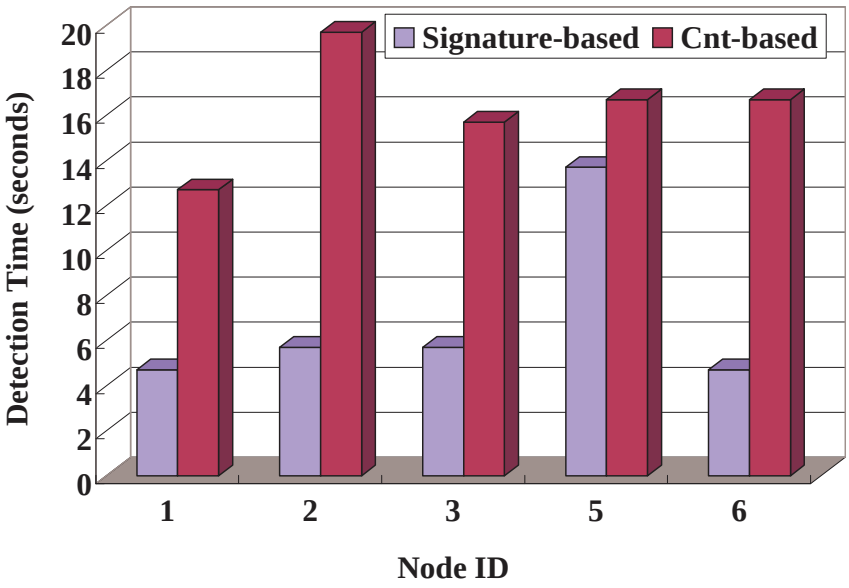


Figure 5.18. Detection delay (for experiment 2)

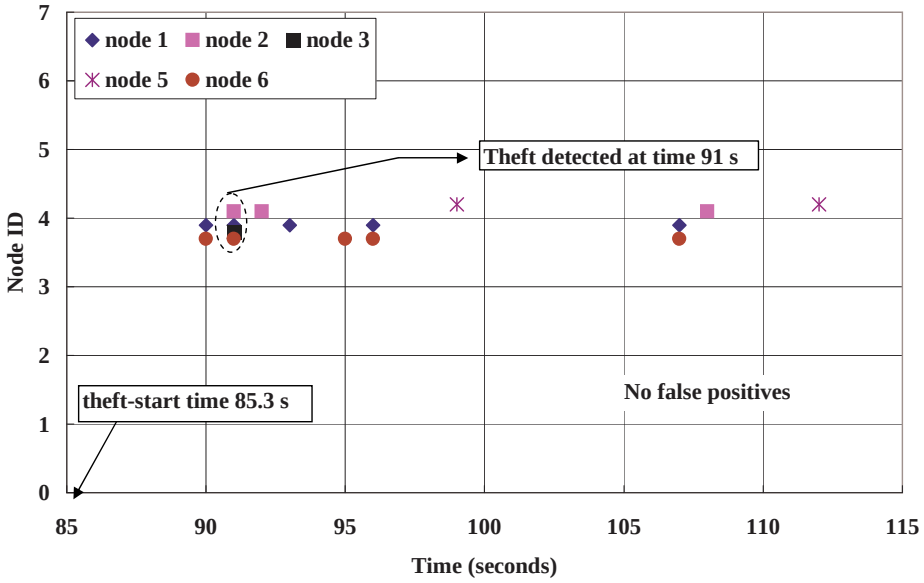


Figure 5.19. False positives of signature-based scheme (for experiment 2)

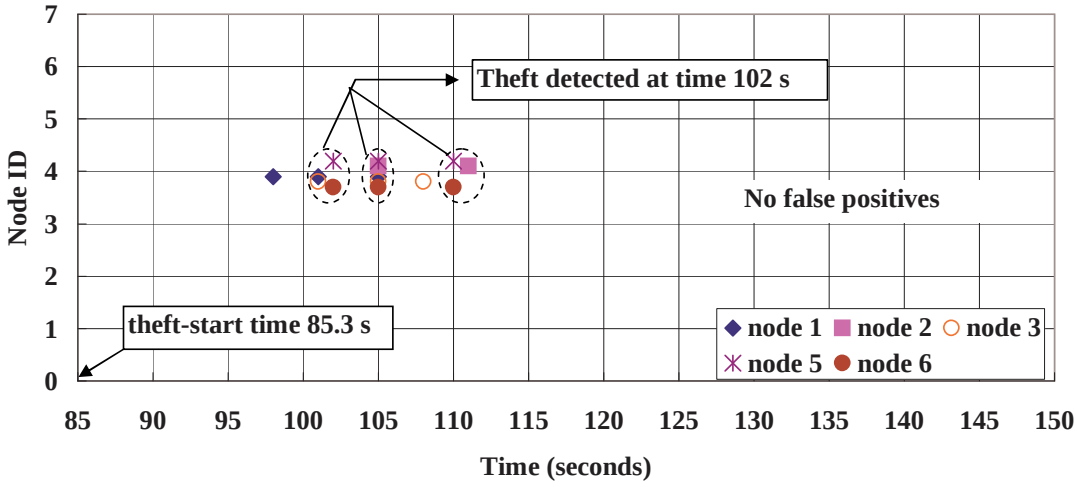


Figure 5.20. False positives of count-based scheme (for experiment 2)

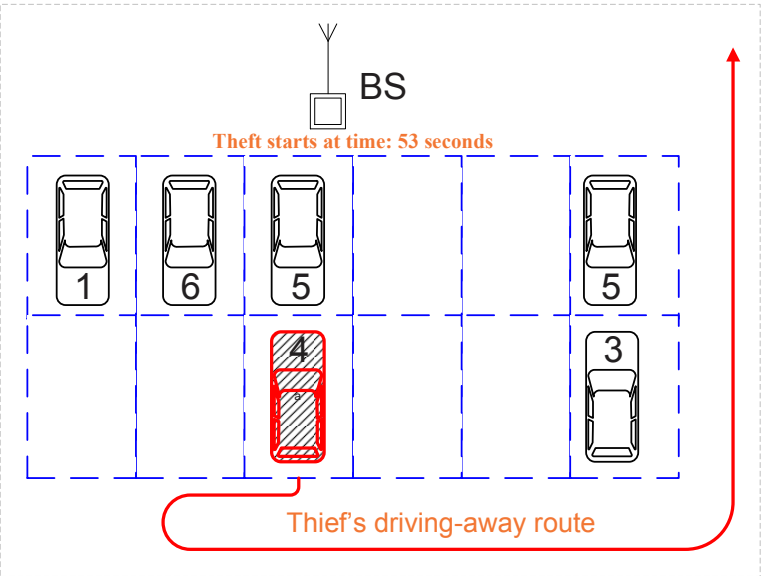


Figure 5.21. Experiment 3

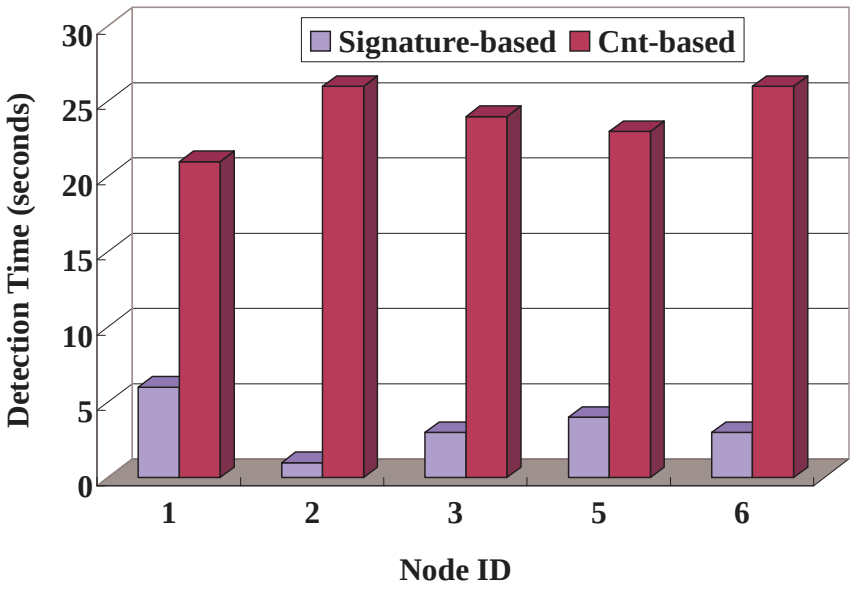


Figure 5.22. Detection delay (for experiment 3)

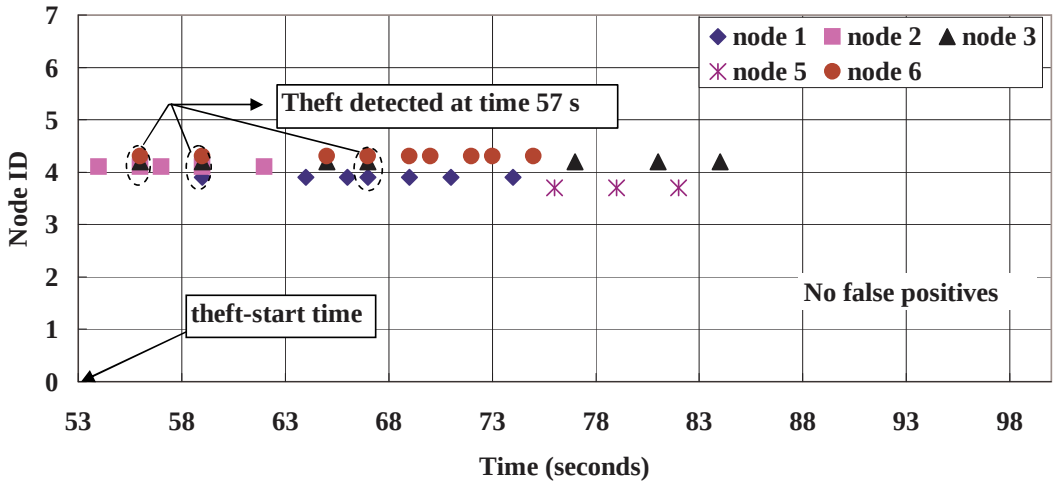


Figure 5.23. False positives of signature-based scheme (for experiment 3)

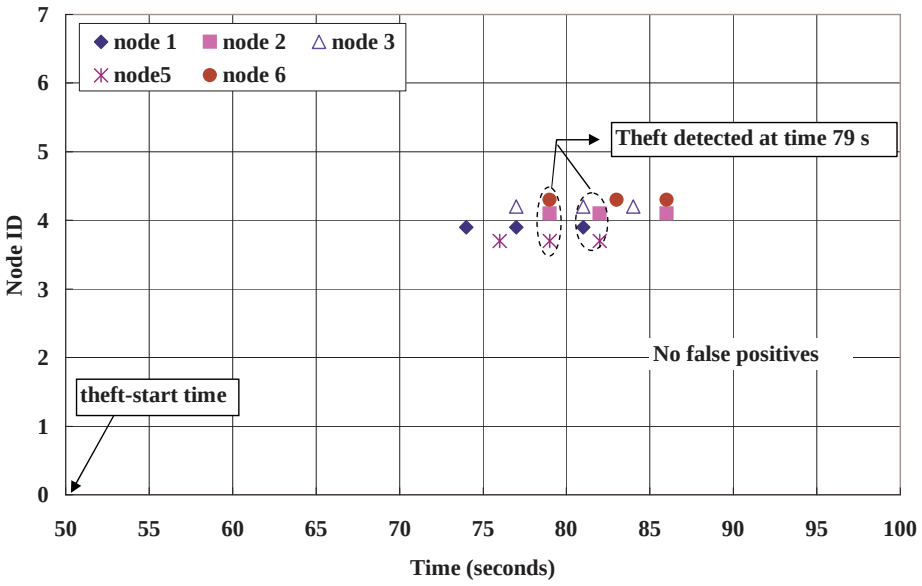


Figure 5.24. False positives of count-based scheme (for experiment 3)

Chapter 6

Conclusions and Future Work

6.1 Summary

In this thesis, we have presented our efforts on securing two of the building blocks of a sensor network, i.e., mobile sink and time synchronization, and have described a vehicle anti-theft system which is built upon sensor network.

To secure the mobile sink building block, we first propose an efficient scheme (based on the the principle of least privilege) to restrict the privilege of a mobile sink without impeding its capability of carrying out any authorized operations. We then propose several efficient message forwarding schemes for depriving the privilege assigned to a compromised mobile sink immediately after its compromise has been detected.

To secure the time synchronization block, we mainly focus on detecting and accommodating delay attack, an attack that cannot be handled by cryptographic techniques. Lastly, we propose a secured wireless sensor network application, where sensor network is used for vehicle theft detection. Extensive experiments as well as simulations have been conducted, which have confirmed the feasibility and effectiveness of our proposed solutions.

6.2 Future Work

Stemming directly from my current thesis, in the near future, I will focus on the following three aspects. First, for the mobile sink compromise problem, I will consider the following two issues. We will support policies more flexible than the

trajectory-based one. For example, both space and time may have different degrees of granularity and logical operations such as OR and AND may be needed. We also notice that, in a sensor application multiple mobile sinks may be dispatched for either the same or different tasks with overlapped time periods or trajectories. Thus, we will examine its impact on both security and performance, and see if some level of performance optimization is possible.

Second, I intend to enhance the current design of SVATS by incorporating more security considerations. For example, we would like to bind each vehicle (or sensor) to a single identity to prevent Sybil or other spoofing attacks. Meanwhile, it is also important to maintain users' privacy when providing strong authentication. How to do secure key distribution efficiently and effectively in a large scale is the next critical issue to investigate. We will also enhance our theft detection protocols to make them more resilient against collusion attacks. In addition, we will extend SVATS to provide stolen vehicle tracking capability.

Third, for time synchronization, my current solution enables a node to achieve attack-resilient time synchronization with one node or with its neighboring nodes. I will extend this work and consider a network-wide secure synchronization algorithm that has minimized the number of messages exchanged among nodes in the network.

We notice that, to secure sensor network, we must integrate security into each of its components (because components designed without security can become a point of attack) and provide security in every aspect of system design. Thus, in the long run, I plan to explore low-level security primitives such as secure key distribution and trust setup in large-scale networks, resilience to communication denial of service attacks, resilience to node capture attacks, etc. I will also investigate high-level security services, including secure data aggregation, intrusion detection, and secure group management.

Bibliography

[1] Klein, L. (1993) "Sensor and Data Fusion Concepts and Applications," *SPIE Optical Engr Press, WA*.

[2] Pottie, G. J. (1998) "Wireless Sensor Networks," *ITW, Killamey, Ireland*.

[3] Tilak, N. Abu-Ghazaleh, and W. Heinzelman, S. (2002) "A Taxonomy of Wireless Micro-Sensor Network Models," *ACM SIGMobile Computing and Communications Review (MC2R)*, **6**.

[4] Akyildiz, W. Su, Y. Sankarasubramaniam, and E.Cayirci, I. (2002) "Wireless Sensor Networks: A Survey," *Computer Networks*, **38**(4).

[5] (2004), "CROSSBOW TECHNOLOGY INC," .

[6] Pottie, G. and W. Kaiser (2000) "Wireless Integrated Network Sensors," *Communications of the ACM*, **43**(5).

[7] Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton and J. Zhao, A. (2000) "Application Driver for Wireless Communications Technology," *ACM SIGCOMM Workshop on Data Communication in Latin America and Caribben*.

[8] Pirretti, M., S. Zhu, V. Narayanan, P. McDaniel, M. Kandemir, and R. Brooks (2005) "The Sleep Deprivation Attack in Sensor Networks: Analysis and Methods of Defense," in *Proceedings of the Innovations and Commercial Applications of Distributed Sensor Networks Symposium (ICA DSN 2005)*.

[9] Seshadri, A., A. Perrig, L. van Doorn, and P. Khosla (2004) "SWAtt: Software-based Attestation for Embedded Devices," in *Proceedings of the IEEE Symposium on Security and Privacy*.

[10] Park, T. and K. Shin (2005) "Soft tamper-proofing via program integrity verification in wireless sensor networks," in *IEEE Trans. Mob. Comput.*

[11] Shaneck, M., K. Mahadevan, V. Kher, and Y. Kim (2005) "software-based attestation for wireless sensors," in *Proceedings of the 2nd European Workshop on Security and Privacy in Ad Hoc and Sensor Networks (ESAS)*.

[12] Du, W., J. Deng, Y. S. Han, and P. K. Varshney (2003) "A Witness-Based Approach for Data Fusion Assurance in Wireless Sensor Networks," in *Proc. of IEEE Global Telecommunications Conference (GLOBECOM '03)*, vol. 3, San Francisco, CA, USA, pp. 1435–9.

[13] Hu, L. and D. Evans (2003) "Secure aggregation for wireless networks," in *Proceedings of Workshop on Security and Assurance in Ad hoc Networks*.

[14] Przydatek, B., D. Song, and A. Perrig (2003) "SIA: Secure Information Aggregation in Sensor Networks," in *Proceedings of ACM SenSys 2003*.

[15] Wagner, D. (2004) "Resilient Aggregation in Sensor Networks," in *Proceedings of ACM Workshop on Security of Ad hoc and Sensor Networks (SASN'04)*.

[16] Castelluccia, C., E. Mykletun, and G. Tsudik (2005) "Efficient Aggregation of Encrypted Data in Wireless Sensor Networks," in *Mobile and Ubiquitous Systems: Networking and Services MobiQuitous 2005*.

[17] Yi, Y., X. Wang, S. Zhu, and G. Cao (2006) "SDAP: A Secure Hop-by-Hop Data Aggregation Protocol for Sensor Networks," in *Proceedings of the ACM Mobihoc*.

[18] Chan, H., A. Perrig, and D. Song (2006) "Secure Hierarchical In-Network Aggregation in Sensor Networks," in *Proceedings of ACM CCS'06*.

[19] Roy, S., S. Setia, and S. Jajodia (2006) "Attack-Resilient Hierarchical Data Aggregation in Sensor Networks," in *Proceedings of ACM SASN Workshop*.

[20] Perrig, A., R. Szewczyk, V. Wen, D. Culler, and J. Tygar (2001) "SPINS: security protocols for sensor netowrks," in *Proceedings of ACM Mobile Computing and Networking (Mobicom'01)*, pp. 189–199.

[21] Liu, D. and P. Ning (2003) "Efficient Distribution of Key Chain Commitments for Broadcast Authentication in Distributed Sensor Networks," in *Proceedings of the 10th Annual Network and Distributed System Security Symposium (NDSS'03)*, pp. 263–276.

[22] Liu, D., P. Ning, S. Zhu, and S. Jajodia (2005) "Practical Broadcast Authentication in Sensor Networks," in *Proceedings of International Conference on Mobile and Ubiquitous Systems (Mobiquitous'05)*.

[23] Zhu, S., S. Setia, S. Jajodia, and P. Ning (2004) "An Interleaved Hop-by-hop Authentication Scheme for Filtering of Injected False Data in Sensor Networks," in *Proceedings of IEEE Symp. on Security and Privacy*, pp. 259–271.

[24] Ye, F., H. Luo, S. Lu, and L. Zhang (2004) "Statistical En-route Detection and Filtering of Injected False Data in Sensor Networks," in *Proceedings of IEEE Infocom'04*.

[25] Yu, Z. and Y. Guan (2007) "A Dynamic En-route Scheme for Filtering False Data in Wireless Sensor Networks," in *Proceedings of IEEE INFOCOM*.

[26] Marti, S., T. Giuli, K. Lai, and M. Baker (2000) "Mitigating Routing Misbehavior in Mobile Ad Hoc Networks," in *Proceedings of ACM MOBICOM*.

[27] Ganeriwal, S. and M. Srivastava (2004) "Reputation-based Framework for High Integrity Sensor Networks," in *Proceedings of ACM workshop SASN'04*.

[28] Gruteser, M., G. Schelle, A. Jain, R. Han, and D. Grunwald (2003) "Privacyaware location sensor networks," in *Proceedings of 9th USENIX-Workshop on Hot Topics in Operating Systems (HotOS IX)*.

[29] Deng, J., R. Han, and S. Mishra (2004) "Intrusion Tolerance Strategies in Wireless Sensor Networks," in *Proceedings of IEEE 2004 International Conference on Dependable Systems and Networks (DSN'04)*.

[30] Kamat, P., Y. Zhang, W. Trappe, and C. Ozturk (2005) "Enhancing Source-Location Privacy in Sensor Network Routing," *Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS)*.

[31] Shao, M., S. Zhu, W. Zhang, and G. Cao (2007) "pDCS: Security and Privacy Support for Data-Centric Sensor Networks," in *Proceedings of IEEE INFOCOM*.

[32] Cardenas, A., S. Radosavac, and J. Baras (2004) "Detection and Prevention of MAC Layer Misbehavior for Ad Hoc Networks," in *Proceedings of ACM Workshop on Security of Ad hoc and Sensor Networks (SASN'04)*.

[33] Raya, M., J. Hubaux, and I. Aad (2004) "Domino: A system to detect greedy behavior in ieee 802.11 hotspots," in *Proceedings of the Second International Conference on Mobile Systems, Applications and Services(MobiSys2004)*.

[34] Wood, A. and J. Stankovic (2002) "Denial of service in sensor networks," *IEEE Computer*, pp. 54–62.

[35] Gu, W., X. Wang, S. Chellappan, K. Schosek, and D. Xuan (2005) "Lifetime Optimization of Sensor Networks under Physical Attacks," in *Proceedings of IEEE International Conference on Communications (ICC)*.

[36] Gu, W., X. Wang, S. Chellappan, D. Xuan, and T. Lai (2005) "Defending against Search-based Physical Attacks in Sensor Networks," in *Proceedings of IEEE Mobile Sensor and Ad-hoc and Sensor Systems (MASS)*.

[37] Eschenauer, L. and V. Gligor (2002) "A Key-Management Scheme for Distributed Sensor Networks," in *Proceedings of ACM CCS'02*.

[38] Chan, H., A. Perrig, and D. Song (2003) "Random Key Predistribution Schemes for Sensor Networks," in *Proceedings of IEEE Security and Privacy Symposim'03*.

[39] Du, W., J. Deng, Y. Han, and P. Varshney (2003) "A Pairwise Key Predistribution Scheme for Wireless Sensor Networks," in *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS'03)*, pp. 42–51.

[40] Liu, D. and P. Ning (2003) "Establishing Pairwise Keys in Distributed Sensor Networks," in *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS '03)*, pp. 52–61.

[41] Pietro, R., L. Mancini, and A. Mei (2003) "Random Key Assignment for Secure Wireless Sensor Networks," in *Proceedings of ACM Workshop SASN*.

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

[43] Liu, D. and P. Ning (2003) "Location-Based Pairwise Key Establishments for Static Sensor Networks," *Proceedings of 2003 ACM Workshop on Security in Ad Hoc and Sensor Networks (SASN '03)*.

[44] Chan, H. and A. Perrig (2005) "PIKE: Peer Intermediaries for Key Establishment in Sensor Networks," in *The 24th Conference of the IEEE Communications Society (Infocom 2005)*.

[45] Wacker, A., M. Knoll, T. Heiber, and K. Rothermel (2005) "A New Approach for Establishing Pairwise Keys for Securing Wireless Sensor Networks," in *Proceedings of ACM SenSys*.

[46] Simonova, K., A. Ling, and X. Wang (2006) "Location-aware key predistribution scheme for wide area wireless sensor networks," in *Proceedings of ACM SASN Workshop*.

[47] Watro, R., D. Kong, S. Cuti, C. Gardiner, C. Lynn, and P. Kruus (2004) "TinyPK: Securing Sensor Networks with Public Key Technology," in *Proceedings of ACM SASN Workshop*.

[48] Liu, A. and P. Ning (2006), "TinyECC: Elliptic Curve Cryptography for Sensor Networks (Version 0.2)," .

[49] Piotrowski, K., P. Langendoerfer, and S. Peter (2006) "How Public Key Cryptography Influences Wireless Sensor Node Lifetime," in *Proceedings of ACM SASN Workshop*.

[50] Karlof, C. and D. Wagner (2003) "Secure Routing in Sensor Networks: Attacks and Countermeasures," in *Proceedings of First IEEE Workshop on Sensor Network Protocols and Applications*.

[51] Parno, B., M. Luke, E. Gaustad, and A. Perrig (2006) "Secure Sensor Network Routing: A Clean-Slate Approach," in *Proceedings of the 2nd CoNext Conference*.

[52] Wood, A., L. Fang, J. Stankovic, and T. He (2006) "SIGF: A Family of Configurable, Secure Routing Protocols for Wireless Sensor Networks," in *Proceedings of ACM SASN Workshop*.

[53] Yin, J. and S. K. Madria (2006) "SecRout: A Secure Routing Protocol for Sensor Networks." in *20th International Conference on Advanced Information Networking and Applications (AINA 2006), 18-20 April 2006, Vienna, Austria*, pp. 393–398.

[54] Zhu, S., S. Setia, and S. Jajodia (2003) "LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks," in *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS '03)*, pp. 62–72.

[55] Blundo, C., A. Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung (1993) "Perfectly-Secure Key Distribution for Dynamic Conferences," in *Advances in Cryptology, Proceedings of CRYPTO'92*, LNCS 740, pp. 471–486.

[56] Huang, D., M. Mehta, D. Medhi, and L. Harn (2004) "Location-aware key management scheme for wireless sensor networks," in *Proceedings of Workshop on Security of Ad Hoc and Sensor Networks*.

[57] Blom, R. (1985) "An Optimal Class of Symmetric Key Generation Systems," in *Advances in Cryptology, Proceedings of EUROCRYPT'84*, LNCS 209, pp. 335–338.

[58] Deng, J., R. Han, and S. Mishra (2003) "Security Support For In-Network Processing in Wireless Sensor Networks," in *Proceedings of First ACM Workshop on the Security of Ad Hoc and Sensor Networks (SASN'03)*.

[59] Capkun, S. and J.-P. Hubaux (2005) "Secure positioning of wireless devices with application to sensor networks," in *IEEE INFOCOM'05*, pp. 1917–1928.

[60] Lazos, L., R. Poovendran, and S. Capkun (2005) "ROPE: robust position estimation in wireless sensor networks," in *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, p. 43.

[61] Anjum, F., S. Pandey, and P. Agrawal (2005) "Secure Localization in Sensor Networks using Transmission Range Variation," in *IEEE MASS'05*.

[62] Lazos, L. and R. Poovendran (2005) "SeRLoc: Robust localization for wireless sensor networks," *ACM Trans. Sen. Netw.*, **1**(1), pp. 73–100.

[63] Liu, D., P. Ning, and W. Du (2005) "Detecting Malicious Beacon Nodes for Secure Location Discovery in Wireless Sensor Networks," in *Proceedings of the The 25th International Conference on Distributed Computing Systems (ICDCS '05)*.

[64] Li, Z., W. Trappe, Y. Zhang, and B. Nath (2005) "Robust statistical methods for securing wireless localization in sensor networks," in *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, p. 12.

[65] Liu, D., P. Ning, and W. Du (2005) "Attack-Resistant Location Estimation in Sensor Networks," in *ACM IPSN*.

[66] Wood, A., J. Stankovic, and S. Son (2003) "JAM: A Mapping Service for Jammed Regions in Sensor Networks," *IEEE RTSS*.

[67] Kansal, A., A. A. Somasundara, D. D. Jea, M. B. Srivastava, and D. Estrin (2004) "Intelligent fluid infrastructure for embedded networks," in *ACM MobiSys '04*, pp. 111–124.

[68] Tirta, Y., Z. Li, Y. Lu, and S. Bagchi (2004) "Efficient Collection of Sensor Data in Remote Fields Using Mobile Collectors," *IEEE ICCCN'04*.

[69] Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang, F. (2002) "A Two-Tier Data Dissemination Model for Large-scale Wireless Sensor Networks," *ACM MOBICOM*, pp. 148–159.

[70] Zhang, W., G. Cao, and T. F. LaPorta (2007) "Data Dissemination with Ring-Based Index for Wireless Sensor Networks," *IEEE Transactions on Mobile Computing, to appear.*

[71] Stajano, F. and R. Anderson (1975) "The Protection of Information in Computing Systems," in *Proceedings of the IEEE.*

[72] Bergbreiter, S. and K. Pister (2003) "CotsBots: An Off-the-Shelf Platform for Distributed Robotics," in *IROS'03.*

[73] McMickell, M. B., B. Goodwine, and L. A. Montestruque (2003) "MICAbot: A Robotic Platform for Large-Scale Distributed Robotics," in *Proceedings of the 2003 IEEE International Conference on Robtics & Automation.*

[74] Sibley, G., M. Rahimi, and G. Sukhatme (2002) "Robomote: A tiny mobile robot platform for large-scale ad-hoc sensor networks," in *Proceedings of the 2002 IEEE International Conference on Robtics & Automation*, vol. 2, Washington D.C., USA, pp. 1143–1148.

[75] Xu, J. Heidemann and D. Estrin, Y. (2001) "Geography Informed Energy Conservation for Ad Hoc Routing," *ACM MOBICOM'01.*

[76] Zhang, W. and G. Cao "DCTC: Dynamic Convoy Tree-Based Collaboration for Target Trackin g in Sensor Networks," *IEEE Transactions on Wireless Communication, in press, also available at: http://www.cse.psu.edu/∼gcao.*

[77] Priyantha, N. B., A. Chakraborty, and H. Balakrishnan (2000) "The Cricket location-support system," in *ACM MobiCom*, pp. 32–43.

[78] Savvides, A., C. Han, and M. Srivastava (2001) "Dynamic Fine-grained Localization in Ad-hoc Networks of Sensors," in *ACM MobiCom*, pp. 166–179.

[79] Priyantha, N. B., A. K. Miu, H. Balakrishnan, and S. Teller (2001) "The cricket compass for context-aware mobile applications," in *ACM MobiCom.*

[80] Ganeriwal, S., R. Kumar, and M. Srivastava (2003) "Timing-sync protocol for sensor networks," in *Proceedings of ACM SenSys'03.*

[81] Song, H., S. Zhu, and G. Cao (2007) "Attack-Resilient Time Synchronization for Wireless Sensor Networks," *Ad Hoc Networks*, **5**(1), pp. 112–125.

[82] Sun, K., P. Ning, and C. Wang (2006) "Secure and resilient clock synchronization in wireless sensor networks," *IEEE Journal on Selected Areas in Communications*, **24**(2), pp. 395–408.

[83] Capkun, S. and J. Hubaux (2004) *Secure Positioning in Sensor Networks*, *Tech. rep.*, Technical report EPFL/IC/200444.

[84] Capkun, S., M. Cagalj, and M. Srivastava (2006) "Securing Localization With Hidden and Mobile Base Stations," in *IEEE INFOCOM '06*, Barcelona, Spain.

[85] Anjum, F., S. Pandey, B. Kim, and P. Agrawal (2005) "Secure localization in sensor networks using transmission range variation," in *IEEE MASS*, pp. 195–203.

[86] Ganeriwal, S., S. Capkun, C.-C. Han, and M. B. Srivastava (2005) "Secure time synchronization service for sensor networks," in *WiSe '05: Proceedings of the 4th ACM workshop on Wireless security*, ACM Press, New York, NY, USA, pp. 97–106.

[87] Goldreich, O., S. Goldwasser, and S. Micali (1986) "How to Construct Random Functions," *Journal of the ACM*, **33(4)**, pp. 210–217.

[88] Merkle, R. (1989) "A certified digital signature," in *Proceedings of Advances in Crypto-89*, pp. 218–238.

[89] Rivest, R. (1994) "The RC5 encryption algorithm," in *Proceedings of the 1st International Workshop on Fast Software Encryption*, pp. 86–96.

[90] Ko, Y. and N. Vaidya (2000) "GeoTORA: A Protocol for Geocasting in Mobile Ad Hoc Networks," *International Conference on Network Protocols (ICNP)*.

[91] Huang, C. Lu, and C. Roman, Q. (2003) "Spatiotemporal Multicast for Sensor Networks," *The First ACM Conference on Embedded Networked Sensor Systems (SenSys)*.

[92] Bose, P., P. Morin, I. Stojmenovic, and J. Urrutia (1999) "Routing with guaranteed delivery in ad hoc wireless networks," in *DIALM '99: Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications*, ACM Press, New York, NY, USA, pp. 48–55.

[93] Karp, B. and H. Kung (2000) "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks," *ACM Mobicom*.

[94] Kuhn, F., R. Wattenhofer, and A. Zollinger (2003) "Worst-Case optimal and average-case efficient geometric ad-hoc routing," in *MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, ACM Press, New York, NY, USA, pp. 267–278.

[95] Hill, J., R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister (2000) "System Architecture Directions for Networked Sensors," in *Architectural Support for Programming Languages and Operating Systems*, pp. 93–104.

[96] Gay, D., P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler (2003) "The nesC language: A holistic approach to networked embedded systems," in *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, ACM Press, New York, NY, USA, pp. 1–11.

[97] Bellare, M., J. Kilian, and P. Rogaway (2000) "The security of the cipher block chaining message authentication code," *J. Comput. Syst. Sci.*, **61**(3), pp. 362–399.

[98] Chapweske, J. and G. Mohr (2002), "Tree Hash EXchange format (THEX)," Available online at http://open-content.net/specs/draft-jchapweske-thex-01.html.

[99] Zhang, W. and G. Cao (2004) "Optimizing Tree Reconfiguration for Mobile Target Tracking in Sensor Networks," *IEEE INFOCOM*.

[100] Elson, J., L. Girod, and D. Estrin (2002) "Fine-grained network time synchronization using reference broadcasts," *SIGOPS Oper. Syst. Rev.*, **36**(SI), pp. 147–163.

[101] Ganeriwal, S., R. Kumar, and M. B. Srivastava (2003) "Timing-sync protocol for sensor networks," in *Proceedings of the 1st Int'l Conf. on Embedded Networked Sensor Systems*, ACM Press, pp. 138–149.

[102] van Greunen, J. and J. Rabaey (2003) "Lightweight time synchronization for sensor networks," in *ACM WSNA*, pp. 11–19.

[103] Sichitiu, M. L. and C. Veerarittiphan (2003) "Simple, Accurate Time Synchronization for Wireless Sensor Networks," *Wireless Communications and Networking (WCNC'03), IEEE*, **2**, pp. 16–20.

[104] Li, Q. and D. Rus (2004) "Global clock synchronization in sensor networks," in *IEEE INFOCOM*.

[105] Chen, M., W. Cui, V. Wen, and A. Woo (2000), "Security and Deployment Issues in a Sensor Network," .

[106] Dai, H. and R. Han (2004) "TSync: a lightweight bidirectional time synchronization service for wireless sensor networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, **8**(1), pp. 125–139.

[107] PalChaudhuri, S., A. K. Saha, and D. B. Johnson (2004) "Adaptive clock synchronization in sensor networks," in *IPSN'04: Proceedings of the third international symposium on Information processing in sensor networks*, ACM Press, New York, NY, USA, pp. 340–348.

[108] Sundararaman, B., U. Buy, and A. D. Kshemkalyani (2005) "Clock synchronization for wireless sensor networks: a survey." *Ad Hoc Networks*, **3**(3), pp. 281–323.

[109] Halpern, J. Y., B. Simons, R. Strong, and D. Dolev (1984) "Fault-tolerant clock synchronization," in *ACM PODC*, pp. 89–102.

[110] Lamport, L. and P. M. Melliar-Smith (1984) "Byzantine clock synchronization," in *ACM PODC*, pp. 68–74.

[111] Lamport, L. and P. Melliar-Smith (1985) "Synchronizing clocks in the presence of faults," *J. ACM*, **32**(1), pp. 52–78.

[112] Olson, A. and K. G. Shin (1994) "Fault-Tolerant Clock Synchronization in Large Multicomputer Systems," *IEEE Trans. Parallel Distrib. Syst.*, **5**(9), pp. 912–923.

[113] Sun, K., P. Ning, and C. Wang (2005) "Fault-tolerant cluster-wise clock synchronization for wireless sensor networks," *IEEE Trans. Dependable and Secure Computing*, **2**(3), pp. 177–189.

[114] Elson, J. and D. Estrin (2001) "Time Synchronization for Wireless Sensor Networks," in *Proceedings of the 15th International Parallel & Distributed Processing Symposium*, IEEE Computer Society, p. 186.

[115] Santivanez, C. and J. Redi (2003) "On the Use of Directional Antennas for Sensor Networks," in *IEEE MILCOM*, .

[116] Crossbow Technology Inc. (2004) "Wireless sensor networks," in *http://www.xbow.com/*, Accessed in November.

[117] Perrig, A., R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler (2002) "SPINS: security protocols for sensor networks," *Wirel. Netw.*, **8**(5), pp. 521–534.

[118] Zhang, W. and G. Cao (2005) "Group Rekeying for Filtering False Data in Sensor Networks: A Predistribution and Local Collaboration-Based Approach," *IEEE INFOCOM*.

[119] Hawkins, D. M. (1980) *Identification of outliers*, New York: Chapman and Hall.

[120] Iglewicz, B. and D. C. Hoaglin (1993) *How to detect and handle outliers*, ASQC basic references in quality control.

[121] Rosner, B. (1983) "Percentage points for generalized ESD many-outlier procedure," in *Technometrics*.

[122] Romer, K. (2001) "Time synchronization in ad hoc networks," in *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*.

[123] USA Today (2004), "Top car-theft areas in each state," .
URL `http://www.usatoday.com/news/nation/2004/11-29-car-thief-table.htm`

[124] Bychkovsky, B. Hull, A. Miu, H. Balakrishnan, and S. Madden, V. (2006) "A Measurement Study of Vehicular Internet Access Using In Situ Wi-Fi Networks," *ACM Mobicom*.

[125] Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden, B. (2006) "CarTel: A Distributed Mobile Sensor Computing System," *ACM Sensys*.

[126] Jiang, V. Taliwal, A. Meier, and W. Holfelder, D. (2006) "Design of 5.9 GHz DSRC-Based Vehicular Safety Communication," *IEEE Wireless Communications Magazine*.

[127] Zhao, J. and G. Cao (2006) "VADD: Vehicle-Assisted Data Delivery in Vehicular Ad Hoc Networks," *IEEE INFOCOM*.

[128] Kris Pister et al. (2004), "SMART DUST: Autonomous sensing and communication in a cubic millimeter," .
URL `http://robotics.eecs.berkeley.edu/ pister/SmartDust/`

[129] Seidel, S. Y. and T. S. Rapport (1992) "914 MHz path loss prediction Model for Indoor Wireless Communications in Multi-floored buildings," *IEEE Trans. on Antennas & Propagation*.

[130] Friis, H. T. (1946) "A note on a simple transmission formula," in *Proc. IRE*, vol. 34.

[131] Hofmann-Wellenhof, B., H. Lichtenegger, and J. Collins (2001) *Global Positioning Systems: Theory and Practice*, Springer Verlag, 5th edition.

[132] Girod, L. and D. Estrin (2001) "Robust Range Estimation Using Acoustic and Multimodal Sensing," *IROS 2001*.

[133] Niculescu, D. and B. Nath (2003) "Ad Hoc Positioning System (APS) Using AoA," in *IEEE INFOCOM*, pp. 1734–1743.

[134] NIST/SEMATECH, "e-Handbook of Statistical Methods," http://www.itl.nist.gov/div898/handbook/toolaids/pff/prc.pdf.

[135] Gu, L. and J. A. Stankovic (2005) "Radio-Triggered Wake-Up for Wireless Sensor Networks," *Real-Time Syst.*, **29**, pp. 157–182.

[136] Shnayder, V., M. Hempstead, B. rong Chen, G. W. Allen, and M. Welsh (2004) "Simulating the power consumption of large-scale sensor network applications," in *SenSys*, pp. 188–200.

[137] Perkins, C. and P. Bhagwat (1994) "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers," in *ACM SIGCOMM*, pp. 234–244.

[138] Johnson, D. B. and D. A. Maltz (1996) "Dynamic Source Routing in Ad Hoc Wireless Networks," in *Mobile Computing*.

[139] Perkins, C. F. and E. M. Royer (1999) "Ad-hoc on-demand distance vector routing," in *IEEE WMCSA*, New Orleans, LA, USA.

[140] Awerbuch, B. (1985) "A New Distributed Depth-First-Search Algorithm." *Inf. Process. Lett.*, **20**(3), pp. 147–150.

[141] Zhu, S., S. Setia, and S. Jajodia (2006) "LEAP+: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks," *ACM Transaction on Sensor Networks*.

Vita

Hui Song

Hui Song was born in Hubei, China. He received the B.S. degree in Computer Science from Beijing Jiaotong University, Beijing, China, in July 1997, and the M.E. degree in Computer Science from Tsinghua University, Beijing, China, in July 2000. He enrolled in the Ph.D. program in Computer Science and Engineering at The Pennsylvania State University in August 2001. He was a recipient of the Annual Best Research Assistant Award for 2005 in the CSE Department at Penn State. He is a student member of ACM and IEEE.