Intrusion Prevention and Detection in Wireless Sensor Networks

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von

Ioannis Krontiris

aus Athen, Griechenland

Mannheim, 2008

Dekan: Professor Dr. Ing. Felix Freiling, Universität Mannheim
Referent: Professor Dr. Ing. Felix Freiling, Universität Mannheim
Korreferent: Professor Dr. Tassos Dimitriou, Carnegie Mellon University

Tag der mündlichen Prüfung: 13. November 2008

To the memory of my grandfather

Acknowledgements

A thesis is never solely the work of the author. Support and encouragement comes from different sources in various ways. In particular, I would like to thank Felix Freiling for accepting me as a Ph.D. student at the University of Mannheim under his supervision. His leadership, support, attention to detail and hard work have set an example I hope to match some day. Seemingly big problems have always turned into non-problems after discussing them with Felix. I am indebted to him for his unselfish commitment of time, trusted conscientious advice, sound guidance, and constant support.

I am also grateful to my co-advisor, Tassos Dimitriou, who is most responsible for introducing me to the field of sensor networks and helping me identify the challenging research problems of this thesis. Tassos has been a friend and mentor. He taught me how to write academic papers, had confidence in me when I doubted myself, and brought out the good ideas in me. Without his encouragement and constant guidance I could not have finished this dissertation. He was always there to meet and talk about my ideas, to proofread and mark up my papers and to ask me good questions to help me think through my problems.

I would also like to thank my colleague Zinaida Benenson for sharing her friendship with me from the first moment we met in Dagstuhl. Our discussions had always been stimulating and inspiring. She has gone out of her way in order to work with me on topics of this thesis, which made it considerably more enjoyable for me. Her strong theoretical background and ideas had a major influence on this work. Of no less importance to me was her moral support and encouragement.

Thanks also to the great master thesis students with whom I have been involved during this work, and especially to Thanassis Giannetsos whose excitement on the topic fueled my own during the last year. His hard work and programming skills contributed to the implementation and experimental evaluation of several ideas and his feedback was always valuable to me.

Moving toward more personal acknowledgments, I owe special gratitude to

my family and friends for continuous and unconditional support. My friend Alexis supported me from the very first moment I took the decision to start this Ph.D. until now. He has always taken interest in my work and believed in me. He is the person that I troubled the most whenever I faced difficulties and his understanding and encouragement was valuable to me. My friend Traianos was the one who supported me the most during my stays in Germany and made them a lot more fun. He made me feel at home and devoted a lot of time trying to teach me German. He was my point of reference for any difficulties I faced with my everyday life in Germany and acted as my main source of energy and excitement.

I am, of course, particularly indebted to my parents for their monumental, unwavering support and encouragement on all fronts. They have truly always been there for me, and without them none of this would have been even remotely possible.

Athens, *Ioannis Krontiris*
August 2008

Contents

Acknowledgements i

List of Publications vii

1 Introduction 1

1.1 Motivation . . . 1
1.2 Problem Statement . . . 2
1.3 Contributions . . . 4
1.3.1 Key Management . . . 5
1.3.2 Secure Network Programming . . . 5
1.3.3 The Intrusion Detection Problem . . . 5
1.3.4 An IDS Architecture . . . 6
1.4 Thesis Outline . . . 6
1.5 Notation . . . 7

2 Security in Sensor Networks 9

2.1 Introduction . . . 9
2.2 Obstacles to Sensor Network Security . . . 10
2.2.1 Constrained Hardware . . . 10
2.2.2 Wireless Communication . . . 11
2.2.3 Exposure to Physical Attacks . . . 11
2.2.4 Large Scale Deployment . . . 11
2.2.5 Aggregation Processing . . . 11
2.3 New Opportunities . . . 11
2.3.1 Broadcast Communication . . . 12
2.3.2 Massive Redundancy . . . 12
2.3.3 Sensors as Routers . . . 12
2.4 Threat Models . . . 12
2.4.1 Outsider Attacks . . . 12

2.4.2 Insider Attacks . . . 13
2.5 Routing Attacks against Sensor Networks . . . 13
2.5.1 The Sinkhole Attack . . . 14
2.5.2 The Wormhole Attack . . . 14
2.5.3 The Sybil Attack . . . 15
2.5.4 The HELLO Flood Attack . . . 15
2.6 Typical Security Requirements . . . 16
2.7 Issues in Sensor Network Security Research . . . 17
2.7.1 Key Establishment and Initial Trust Setup . . . 17
2.7.2 Resilience to Denial of Service Attacks . . . 18
2.7.3 Resilience to Node Compromises . . . 19
2.7.4 Routing Security . . . 20
2.7.5 Location Aware Security . . . 21
2.7.6 Secure Aggregation and Dissemination . . . 22
2.7.7 Link-layer Security . . . 23
2.7.8 Secure Network Programming . . . 25
2.7.9 Efficient Cryptographic Primitives . . . 26
2.8 Conclusions . . . 26

3 Key Management **29**

3.1 Introduction . . . 29
3.1.1 Related Work . . . 30
3.1.2 Chapter Contribution . . . 31
3.1.3 Chapter Outline . . . 32
3.2 Security Protocol . . . 32
3.2.1 Initialization . . . 33
3.2.2 Cluster Key Setup . . . 33
3.2.3 Secure Message Forwarding . . . 37
3.2.4 Addition of New Nodes . . . 40
3.3 Key Establishment for Secure Aggregation . . . 41
3.4 Key Establishment for Secure Dissemination . . . 43
3.4.1 Defending Against Impersonation Attacks . . . 44
3.5 Experimental Analysis . . . 45
3.6 Security Analysis . . . 46
3.7 Conclusions . . . 49

4 Secure Network Programming **51**

4.1 Introduction . . . 51
4.2 Network Programming . . . 52
4.3 Problem Definition . . . 53
4.4 Design Approaches and Related Work . . . 55
4.4.1 DOS-attack Resilience . . . 55
4.4.2 Signature Scheme . . . 56

4.5 An r-time Signature Scheme . . . 57
4.5.1 Public Key and Signature Size . . . 60
4.5.2 Security Level . . . 61
4.5.3 Signature Verification Time . . . 62
4.6 Implementation and Experiments . . . 64
4.6.1 Implementation . . . 64
4.6.2 Memory Requirements . . . 65
4.6.3 Experimental Evaluation . . . 67
4.6.4 Updating the Public Key . . . 70
4.7 Conclusions . . . 70

5 The Intrusion Detection Problem 71
5.1 Introduction . . . 71
5.2 Designing an IDS for Sensor Networks . . . 72
5.2.1 Intrusion Detection Techniques . . . 72
5.2.2 Intrusion Detection Architectures . . . 73
5.2.3 Decision Making Techniques . . . 74
5.3 The Watchdog Approach . . . 75
5.4 Requirements of IDS for WSN . . . 77
5.5 Existing Approaches . . . 78
5.6 System Model . . . 79
5.6.1 Sensor Nodes and Communication . . . 79
5.6.2 Attacker Model . . . 80
5.6.3 Alert Module . . . 80
5.7 The Intrusion Detection Problem . . . 81
5.8 Conditions for Solving Intrusion Detection . . . 82
5.8.1 Sufficient Conditions for Solving IDP . . . 83
5.8.2 Necessary and Sufficient Conditions for Solving IDP . . . 85
5.8.3 Byzantine Agreement vs. Intrusion Detection . . . 86
5.9 Conclusions . . . 87

6 A Cooperative Intrusion Detection Algorithm 89
6.1 Introduction . . . 89
6.2 System Model . . . 90
6.3 Threat Model . . . 91
6.4 The Algorithm . . . 91
6.4.1 Initialization Phase . . . 92
6.4.2 Voting Phase . . . 92
6.4.3 Publish Key Phase . . . 94
6.4.4 Exposing the Attacker . . . 96
6.4.5 External Ring Reinforcement Phase . . . 97
6.4.6 Responding to the Attack . . . 98
6.5 The IDS Architecture . . . 99

6.6 Simulation Results 101
6.7 Implementation 102
6.7.1 Memory Requirements 103
6.7.2 Experiments 104
6.8 Conclusions 106

7 Conclusions **109**
7.1 Summary of Main Results 109
7.2 Discussion and Future Research 110

Bibliography **113**

List of Publications

This Thesis is a monograph, which contains some unpublished material, but is mainly based on the following publications.

Book Chapters

1. T. Dimitriou, I. Krontiris, and F. Nikakis. Fast and scalable key establishment in sensor networks. In S. Phoha, T. F. L. Porta, and C. Griffin, editors, *Sensor Network Operations*, pp. 557–570. Wiley-IEEE Press, 2006.

2. T. Dimitriou and I. Krontiris. Secure in-network processing in sensor networks. In Y. Xiao, editor, *Security in Sensor Networks*, chap. 12, pp. 275–290. CRC Press, 2006.

3. I. Krontiris and T. Dimitriou. Secure network programming in wireless sensor networks. In Y. Xiao and Y. Pan, editors, *Security in Distributed and Networking Systems*, chap. 12, pp. 289–310. World Scientific Publishing Co., 2007.

4. I. Krontiris, T. Dimitriou, H. Soroush, and M. Salajegheh. WSN link-layer security frameworks. In J. Lopez and J. Zhou, editors, *Wireless Sensors Networks Security*, chap. 6, pp. 142–163. IOS Press, 2008.

5. T. Giannetsos, I. Krontiris, T. Dimitriou, and F. C. Freiling. Intrusion detection in wireless sensor networks. In Y. Zhang and P. Kitsos, editors, *Security in RFID and Sensor Networks*. CRC Press, Taylor&Francis Group, 2008.

Journals

6. T. Dimitriou and I. Krontiris. GRAViTy: Geographic routing around voids. *International Journal of Pervasive Computing and Communica-*

tions, Special Issue on Key Technologies and Applications of Wireless Sensor and Body-area Networks, vol. 2(4):pp. 351–361, 2006.

Conferences

7. I. Krontiris and T. Dimitriou. Authenticated in-network programming for wireless sensor networks. In T. Kunz and S. S. Ravi, editors, *AD-HOC Networks & Wireless – ADHOC-NOW*, vol. 4104 of *Lecture Notes in Computer Science*, pp. 390–403. Springer, 2006.

8. I. Krontiris, T. Dimitriou, and F. C. Freiling. Towards intrusion detection in wireless sensor networks. In *Proceedings of the 13th European Wireless Conference*. Paris, France, April 2007.

9. I. Krontiris, T. Dimitriou, and T. Giannetsos. LIDeA: A distributed lightweight intrusion detection architecture for sensor networks. In *Proceeding of the 4th International Conference on Security and Privacy for Communication (SECURECOMM '08)*. Istanbul, Turkey, September 2008.

10. I. Krontiris and T. Dimitriou. Security issues in biomedical sensor networks. In *First International Symposium on Applied Sciences in Bio-Medical and Communication Technologies (ISABEL '08)*. Aalborg, Denmark, October 2008.

Workshops

11. T. Dimitriou and I. Krontiris. Autonomic communication security in sensor networks. In I. Stavrakakis and M. Smirnov, editors, *Autonomic Communication – WAC*, vol. 3854 of *Lecture Notes in Computer Science*, pp. 141–152. Springer, 2005.

12. T. Dimitriou and I. Krontiris. A localized, distributed protocol for secure information exchange in sensor networks. In *Proceedings of the 5th IEEE International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN '05)*. April 2005.

13. I. Krontiris and T. Dimitriou. A practical authentication scheme for in-network programming in wireless sensor networks. In *Proceedings of the 2nd ACM Workshop on Real-World Wireless Sensor Networks (REALWSN '06)*, pp. 13–17. Uppsala, Sweden, June 2006.

14. I. Krontiris, T. Dimitriou, T. Giannetsos, and M. Mpasoukos. Intrusion detection of sinkhole attacks in wireless sensor networks. In M. Kutylowski, J. Cichon, and P. Kubiak, editors, *Algorithmic Aspects of Wireless Sensor Networks – ALGOSENSORS*, vol. 4837 of *Lecture Notes in Computer Science*, pp. 150–161. Springer, 2007.

15. I. Krontiris and T. Dimitriou. Launching a sinkhole attack in wireless sensor networks; the intruder side. In *First International Workshop on Security and Privacy in Wireless and Mobile Computing, Networking and Communications*. Avignon, France, October 2008.

Posters

16. T. Dimitriou, I. Krontiris, F. Nikakis, and P. G. Spirakis. SPEED: Scalable protocols for efficient event delivery in sensor networks. In N. Mitrou, K. P. Kontovasilis, G. N. Rouskas, I. Iliadis, and L. F. Merakos, editors, *NETWORKING*, vol. 3042 of *Lecture Notes in Computer Science*, pp. 1300–1305. Springer, 2004.

17. M. Salajegheh, H. Soroush, A. Thomos, T. Dimitriou, and I. Krontiris. .Sense, a secure framework for sensor network data acquisition, monitoring and command. In *Proceedings of the 2nd ACM Workshop on Real-World Wireless Sensor Networks (REALWSN '06)*, pp. 101–102. Uppsala, Sweden, June 2006.

Chapter 1

Introduction

1.1 Motivation

During the past few years there has been an explosive growth in the research devoted to the field of wireless sensor networks (WSN), covering a broad range of areas, from understanding theoretical issues to technological advances that made the realization of such networks possible. These networks use hundreds to thousands of inexpensive wireless sensor nodes (motes) over an area for the purpose of monitoring certain phenomena and capture geographically distinct measurements over a long period of time. Nodes employed in sensor networks are characterized by limited resources such as storage, computational and communication capabilities. The power of sensor networks, however, lies exactly in the fact that their nodes are so small and cheap to build that a large number of them can be used to cover an extended geographical area.

The pervasive interconnection of such devices has given birth to a broad class of exciting new applications in several areas of our lives, including environment and habitat monitoring, healthcare applications, home automation, and traffic control. However, as every network, sensor networks are exposed to security threats which, if not properly addressed, can exclude them to be deployed in the envisaged scenarios. Their wireless and distributed nature and the serious constraints in node battery power prevent previously known security approaches to be deployed and has created a large number of vulnerabilities that attackers can exploit, in order to gain access in the network and the information transferred within.

For example, in an *outsider attack*, where the attacker node is not an authorized participant of the sensor network, it may inject useless packets in the network in order to exhaust the energy levels of the nodes, or passively eavesdrop on the network's traffic and retrieve secret information. Even worse, in an *insider attack*, the attacker has compromised a legitimate sensor node and uses the stolen key material, code and data in order to communicate with the rest

of the nodes, as if it was an authorized node. With this kind of intrusion, an attacker can launch more powerful and hard to detect attacks that can disrupt or paralyze the network.

Securing sensor networks against these threats is a challenging research area, necessary for commercially attractive deployments. Unfortunately, while sensor networks were in their infancy, the main research focus was on making sensor networks feasible and useful, and less emphasis was placed on security. A number of new protocols have been designed for *TinyOS* [Hil00], which is an operating system specially designed for wireless sensor networks. Most of these protocols are built assuming a trusted environment and are very vulnerable against security attacks. Addressing these vulnerabilities can become very complex. Different applications employ different types of protocols, and different protocols have different attacks and weaknesses that require different security mechanisms. Therefore, our focus should not only be on how to secure sensor networks, but also on how this can be done through generic and independent solutions.

1.2 Problem Statement

The broadcast nature of the transmission medium in wireless sensor networks makes information more vulnerable than in wired applications. Thus, security mechanisms such as encryption and authentication are essential to protect information transfers. However, existing network security mechanisms are not feasible in this domain, given the limited processing power, storage, bandwidth and energy resources. Public-key algorithms, such as RSA are undesirable, as they are computationally expensive. Instead, symmetric encryption/decryption algorithms and hashing functions are between two to four orders of magnitude faster [Car00], and constitute the basic tools for securing sensor networks communications.

To develop security mechanisms and protocols for sensor networks, a necessary requirement is *key management*, i.e., the establishment and maintenance of shared keys between pairs of communicating nodes. However, bootstrapping secure communications between sensor nodes, i.e., setting up secret keys among them, can become a challenging task. If we knew which nodes would be in the same neighborhood before deployment, keys could be decided a priori. Unfortunately, most sensor network deployments are random, therefore such a priori knowledge does not exist. There are also some other requirements that need to be considered while designing a key management protocol. A desirable feature is resistance to node capture. Even if a node is compromised and its key material is revealed, an adversary should not be able to gain control of other parts of the network by using this material. Therefore the compromise of nodes should result in a breach of security that is constrained within a small, localized part of the network.

Another problem that must be handled well by key management schemes is that of simple message broadcast. Usually nodes establish pairwise keys with

their one-hop neighbors, since in sensor network applications, nodes communicate with their immediate neighbors. If a node shares a different key (or set of keys) with each of its neighbors, then it will have to make multiple transmissions of messages, encrypted each time with a different key, in order to broadcast a message to all of its neighbors. In these cases, we believe that transmissions must be kept as low as possible because of their high energy consumption rate.

Finally, a closely related problem to that of broadcasting encrypted messages is the ability to perform aggregation and data fusion processing [Int03]. This however can be done only if intermediate nodes have access to encrypted data to (possibly) discard extraneous messages reported back to the base station. The use of pair-wise shared keys effectively hinders data fusion processing.

It is easy to see that protecting the communication channel between two nodes does not entirely guarantee the security of the sensor network. Different communication paradigms must also be secured in order to withstand attacks initiated by an adversary. For example, network programming protocols have emerged recently for sensor networks that allow someone to disseminate a new program image remotely, over the wireless link to the entire network, in a multi-hop fashion, reprogramming the motes with new software. Currently these protocols are not secured, allowing an attacker to disseminate malicious code and reprogram the motes with her own code. Therefore, an *authentication scheme for network programming* is needed to ensure that the program image originates from the base station.

The most natural solution for authenticated network programming is asymmetric cryptography, where messages are signed with a key known only to the sender. Everybody can verify the authenticity of the messages by using the corresponding public key, but no one can produce legitimate signed messages without the secret key. However, public key schemes should be avoided in sensor networks for multiple reasons: long signatures induce high communication overhead per packet, verification time places a lower bound on the computational abilities of the receiver, and so on, so forth. What makes the problem even more challenging is that we do not deal with simple messages, but streams, i.e., sequences of packets. The size of program images that will be sent over the radio is usually between a few hundreds of kilobytes and a few thousands. So the problem is to provide an efficient authentication mechanism for a finite stream of data.

Encryption and authentication mechanisms provide reasonable defense for mote-class outsider attacks. However, cryptography is inefficient in preventing against laptop-class and insider attacks. It remains an open problem for additional research and development. The presence of insiders significantly lessens the effectiveness of link layer security mechanisms. This is because an insider is allowed to participate in the network and have complete access to any messages routed through the network and is free to modify, suppress, or eavesdrop on the contents.

There are several classical security methodologies so far that focus on trying to prevent these intrusions. However, it is impossible, or even infeasible, to guarantee perfect prevention. Not all types of attacks are known, and new ones

appear constantly. As a result, attackers can always find security holes to exploit in order to gain access in the sensor network. These intrusions will go unnoticed and they will likely lead to failures in the normal operation of the network, as Figure 1.1(a) suggests.

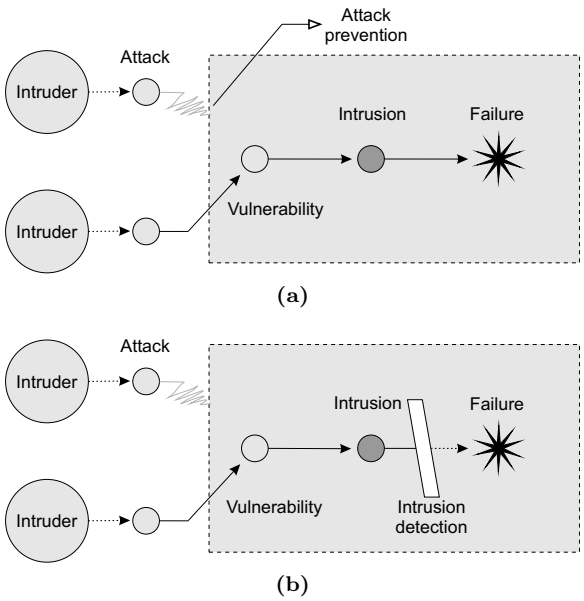

Figure 1.1: *Intrusion sequence. (a) Attackers may exploit a vulnerability and intrude into the network, causing a failure. (b) Intrusion detection functions as a second line of defence.*

The last resort is *intrusion detection*, which can act as a second line of defense: it can *detect* third party break-in attempts, even if this particular attack has not been experienced before. If the intruder is detected soon enough, one can take appropriate measures before any damage is done or any data is compromised (Figure 1.1(b)). An effective intrusion detection system (IDS) can also help us design better prevention mechanisms, by collecting information about intrusion techniques and attack patterns.

The research of IDS in wireless sensor networks has not advanced significantly. There are a few attempts that concentrate on specific attacks, but not a generalized approach that can be both realistic and lightweight enough to run on computationally and memory restricted devices such as the nodes of a sensor network.

1.3 Contributions

Our contributions are the solutions we provide to each of the research problems given in the previous section.

1.3.1 Key Management

We first propose a distributed, deterministic key management protocol designed to satisfy authentication and confidentiality, without the need of a key distribution center. The scheme is scalable since every node only needs to hold a small number of keys independent of the network size, and it is resilient against node capture and replication due to the fact that keys are localized; keys that appear in some part of the network are not used again. Another important property of the protocol is that it is optimized for message broadcast; each node shares one pairwise key with all of its immediate neighbors, so only one transmission is necessary to broadcast a message.

This protocol was later used to design novel techniques for secure in-network processing (i.e., secure data aggregation) for sensor networks [Dim06b]. Data aggregation is possible only if intermediate nodes have access to encrypted data so that they can extract measurement values and apply to them aggregation functions. Therefore, nodes that send data packets toward the base station must encrypt them with keys available to the aggregating nodes, something that this key management scheme provides. It was also used as a base to design new frameworks for link-layer security, like L^3Sec [Sor07; Kro08d], which provides secure services to the higher levels such as confidentiality and authentication in TinyOS.

1.3.2 Secure Network Programming

Next we propose Scatter, a secure code authentication scheme for efficient reprogramming sensor networks. This allows the sensor nodes to efficiently verify that the new code originates from a trusted source, namely the base station. With this security feature added, an attacker can not authenticate herself to the network, and therefore the nodes will reject malicious updates. We followed the approach of constructing a hash chain on the pages of the program image and contribute a novel algorithm to sign the commitment of the chain. The most appropriate method for this task was proved to be an r-times signature scheme. However, there is not such a scheme appropriate for sensor networks. Therefore, we propose a novel r-times signature scheme, which based on Merkle trees manages to reduce the public key size considerably and make it suitable for the sensor nodes.

In this way, Scatter avoids the use of Elliptic Key Cryptography and manages to surpass all previous attempts for secure code dissemination in terms of energy consumption and time efficiency. Besides the design and theoretical analysis of the protocol, we also report the experimental evaluation of Scatter in two different hardware platforms, namely Mica2 and MicaZ, which proves its efficiency in practice.

1.3.3 The Intrusion Detection Problem

Next we introduce the problem of intrusion detection in sensor networks. Based on our experience exploring this problem space, we propose several general guidelines for the design of intrusion detection systems in sensor networks. Specifically, we propose a novel IDS architecture that relies on the cooperation of the nodes. Our approach organizes the IDS agents according to the distributed nature of the events involved in the attacks, and, as a result, an agent needs to send information to other agents only when this information is necessary to detect the attack. The coordination mechanism arranges the message passing between the agents in such a way so that the distributed detection is equivalent to having all events processed in a central place.

Note that distributed intrusion detection in wireless sensor networks is not an easy task since the attacker can also participate in the protocol and try to bring honest nodes to a wrong conclusion. So, first we identified *necessary and sufficient conditions* on the behavior of the local modules such that they contain enough information to cooperatively solve the intrusion detection problem. These conditions also identify scenarios in which cooperative intrusion detection is unsolvable. These conditions are mainly related to the symmetry of views between nodes and show that even the supposedly simple case of one malicious node is surprisingly complex. However we investigated the probability that these symmetry conditions occur in practice using simulations. Overall, they showed that in randomly constructed networks our algorithm can reliably detect the attacker with high probability.

1.3.4 An IDS Architecture

In the proposed IDS, each node hosts an independent intrusion detection agent, capable of detecting intrusions locally based on the data collected by itself and by other neighboring nodes. There are no a priori trusted nodes, or any reputation system. Instead, the system allows the arbitrary behavior of the nodes: a node may behave normally with respect to routing in order to avoid being detected by the IDS, but it can expose a malicious behavior to obstruct the successful detection of another intruder node. The IDS system is based on the power of the majority to protect itself from these misbehaving nodes.

The research of intrusion detection in wireless sensor networks has not advanced significantly so far. The implementation of our IDS is the first to appear in the community. The experimental evaluation show that it is lightweight enough to run on the nodes, requiring only limited computational and memory resources. This proves that studying the problem of IDS in sensor networks is a viable approach and with further research it can provide even more attractive and efficient solutions for securing such networks.

1.4 Thesis Outline

The rest of the dissertation is organized as follows. In Chapter 2, we review the main research directions of sensor network security, we emphasize on the way they are addressed by this thesis and we also outline the important open questions in the area that we believe will draw the attention of the research community in the future.

Chapter 3 presents a distributed, deterministic key management protocol designed to satisfy authentication and confidentiality, without the need of a key distribution center. We discuss how it can be used to secure information propagated toward the base station, but also how to secure aggregation of this information at intermediate nodes. Then we investigate the reverse direction of disseminating information from the base station to the nodes. Finally, we describe a mechanism for evicting compromised nodes as well as adding new nodes. A security analysis is discussed and simulation experiments presented.

Chapter 4 addresses the problem of securing the dissemination of bulk data, like code updates, from the base station to the nodes. We provide a protocol that yields source authentication in the group setting like a public-key signature scheme, only with signature and verification times much closer to those of a MAC. We show how Deluge can be augmented with our solution to give a secure and practical in-network programming system. Our implementation in TinyOS allowed us to test the performance of this system on a real network of motes and prove its efficiency.

Chapter 5 introduces the problem of intrusion detection in wireless sensor networks. We review intrusion detection techniques and architectures from wired and ad-hoc networks and identify which approaches are best for sensor networks. Then we define the problem formally based on a generic system model and we prove a necessary and sufficient condition for successful detection of the attacker. We discuss simulation results showing the probability that this condition does not hold and the probability of identifying the attacker.

Chapter 6 presents a lightweight intrusion detection system that is designed for wireless sensor networks and is based on the theory of Chapter 5. The protocol is based on a distributed architecture, in which nodes overhear their neighboring nodes and collaborate with each other in order to successfully detect an intrusion. We show the overhead imposed by the implementation of such a protocol in TinyOS in terms of memory, communication and computation cost.

Finally, Chapter 7 summarizes the thesis and concludes with a prediction of future technological trends.

1.5 Notation

Throughout the rest of this thesis we shall use the function names E, V, h and MAC respectively for encryption, verification, hash and message authentication code, with optional subscript and superscript to indicate key and algorithm. Table 1.1 captures the most important notations and explains their meanings.

Table 1.1: *Notations and their meanings.*

Notation	Meaning
m	Message
σ	Signature
PK	Public Key
H	Hash function
h	Hash value
F	One-way function
$m_1 \| \| m_2$	Concatenation of messages m_1 and m_2
$E_{K,IV}(m)$	Encryption of message m, with key K and the initialization vector IV, which is used in counter mode (CTR)
$MAC_K(m)$	MAC of message m using key K

For protocols, we shall adopt the classical notation whereby

$$A \rightarrow B : m$$

indicates that principal A sends message m to principal B.

Chapter 2

Security in Sensor Networks

2.1 Introduction

The design of many sensor network applications or protocols for lower layers assume that all nodes are cooperative and trustworthy. This is not the case in most cases of real-world deployments, where the nodes are exposed to many threats that can severely damage the proper network functionality. There are many attacks designed to exploit the unreliable communication channels and the unattended sensor nodes.

Most sensor networks actively monitor their surroundings, and it is often easy to deduce information other than the data monitored. Such information leakage often results in loss of privacy for the people in the environment. Moreover, the wireless communication employed by sensor networks facilitates eavesdropping and packet injection by an adversary. The combination of these factors demands security for sensor networks to ensure operation safety, secrecy of sensitive data and privacy for people in sensor environments.

Nevertheless, sensor networks cannot rely on human intervention to face an adversary's attempt to compromise the network or hinder its proper operation. Neither can they employ existing security mechanisms such as public key infrastructures that are computationally expensive. Instead, an autonomic response of the network that relies on the embedded pre-programmed policies and a coordinated, cooperative behavior is the most effective way to gain maximum advantage against adversaries.

In this chapter we give an overview of the security issues in sensor networks. First we present the limitations of sensor networks that make security for such networks hard, but also their unique characteristics that can be exploited to facilitate the security architect. Then, we formulate the threat model and discuss the requirements that a security protocol has to meet. Following that, we take a first step toward establishing a comprehensive set of security challenges for sensor networks. This overview helps identify research challenges and sets the

scene for the following chapters that present individual research contributions.

2.2 Obstacles to Sensor Network Security

Although wireless sensor networks have an ad-hoc nature, there are several limitations that make security mechanisms proposed for ad-hoc networks not applicable in this setting. In particular, security in sensor networks is complicated by more constrained resources and the need for large-scale deployments. A summary of these limitations follows below:

Table 2.1: *Selection of currently available wireless sensor nodes.*

Platform	MCU	RAM	Program Memory	Radio Chip
BTnode3	ATMega 128	64 KB	128 KB	CC1000/Bluetooth
Cricket	ATMega 128	4 KB	128 KB	CC100
Imote2	Intel PXA271	256 KB	32 MB	CC2420
MICA2	ATMega 128	4 KB	128 KB	CC1000
MICAz	ATMega 128	4 KB	128 KB	CC2420
Tmote Sky	TI MSP 430	10 KB	48 KB	CC2420
Shimmer	TI MSP 430	10 KB	48 KB	CC2420/Bluetooth

2.2.1 Constrained Hardware

A wide range of sensor node platforms has emerged over the past five years. So far, for such devices, the trend has been to increase the lifetime of the nodes by decreasing the resources such as memory, CPU, and radio bandwidth. Therefore, motes have tiny resources, on the order of a few kilobytes of RAM and a few megahertz of processor. For example, Table 2.1 indicates the resources available by some popular mote platforms, like Mica2 developed by UC Berkeley in collaboration with the Crossbow corporation, or the BTnode family from ETH Zurich [Beu03].

Establishing secure communication between sensor nodes becomes a challenging task, given these limited resources, as well as the lack of control of the wireless communication medium. Public-key algorithms, such as RSA [Riv78] or Diffie-Hellman key agreement [Dif76] are undesirable, as they are computationally expensive. Instead, symmetric encryption/decryption algorithms and hash functions are between two to four orders of magnitude faster [Car00], and constitute the basic tools for securing sensor network communications. However, symmetric key cryptography is not as versatile as public key cryptography, which complicates the design of secure applications.

2.2.2 Wireless Communication

Sensor nodes communicate through wireless communication, which is particularly expensive from an energy point of view (one bit transmitted is equivalent to about a thousand CPU operations [Hil00]). Hence one cannot use complicated protocols that involve the exchange of a large number of messages. Additionally, the nature of communication makes it particularly easy to eavesdrop, inject malicious messages into the wireless network or even hinder communications entirely using radio jamming.

2.2.3 Exposure to Physical Attacks

Unlike traditional networks, sensor nodes are often deployed in areas accessible by an attacker, presenting the added risk of physical attacks that can expose their cryptographic material or modify their underlying code. This problem is magnified further by the fact that sensor nodes cannot be made tamper-resistant due to increases in hardware cost. Therefore, sensor nodes are more likely to suffers a physical attack in such an environment compared to typical PCs, which are located in a secure place and mainly face attacks from a network.

2.2.4 Large Scale Deployment

Future sensor networks will have hundreds to thousands of nodes so it is clear that scalability is a prerequisite for any attempt in securing sensor networks. Security algorithms or protocols that were not designed with scalability in mind offer little or no practical value to sensor network security.

2.2.5 Aggregation Processing

An effective technique to extend sensor network lifetime is to limit the amount of data sent back to reporting nodes since this reduces communication overhead [Int03]. However, this cannot be done unless intermediate sensor nodes have access to the exchanged data to perform data fusion processing. End-to-end confidentiality should therefore be avoided as it hinders aggregation by intermediate nodes and complicates the design of energy-aware protocols.

2.3 New Opportunities

Even though the unique characteristics of sensor networks pose some new challenges in security, they also lead to some new opportunities for designing secure protocols and open the door for an entirely new security paradigm. The same properties that allow an attacker to intrude into a sensor network can be used as defense mechanisms, if exploited properly. Below we outline some of these characteristics from the security architect point of view.

2.3.1 Broadcast Communication

As we saw in Section 2.2.2, an attacker can take advantage of the wireless medium and broadcast communication of sensor nodes for intercepting or jamming transmitted packets. In the same way, legitimate nodes can eavesdrop on the traffic passing through their neighborhood. This can constitute a powerful monitoring mechanism for suspicious or abnormal behaviors and lead to the detection of an intruder node.

2.3.2 Massive Redundancy

Sensor nodes are typically low-cost devices allowing sensor networks to pose large scale and massive redundancy. Due to these characteristics the loss or corruption of a sensor node can either be mitigated by redundant sensors or tolerated. Therefore, it is possible to device security protocols that tolerate failures and work correctly even if up to t out of n nodes are compromised by the attacker. Also, in case that the network becomes aware of the intrusion, it can restore its proper operation by using redundant information distributed in other parts of the network.

2.3.3 Sensors as Routers

All sensor nodes act as routers of information toward the base station, in contrast to traditional networks which are based on specific traffic concentration points. Therefore, in sensor networks traffic is distributed for load balancing purposes and it is impossible for an attacker to monitor or control it at certain points. This considerably increases the effort that she has to make, but it also allows the network to reconfigure itself easily in case of node compromises, by setting up alternative paths.

2.4 Threat Models

In sensor networks security, an attacker can perform a wide variety of attacks. Not all of them have the same goal or motivations. So, in order to plan and design better defense systems, we formulate a threat model that distinguishes between two types of attacks: outsider attacks and insider attacks. We now treat each one of these classes in turn.

2.4.1 Outsider Attacks

In an outsider attack (intruder node attack), the attacker node is not an authorized participant of the sensor network. Authentication and encryption techniques prevent such an attacker to gain any special access to the sensor network. The intruder node can only be used to launch passive attacks, like the following:

- *Passive eavesdropping*: The attacker eavesdrops (listens) and records (saves) encrypted messages. The messages may then be analyzed in order to discover secret keys.

- *Denial of service attacks*: In its simplest form, an adversary attempts to disrupt the network's operation by broadcasting high-energy signals. In this way, communication between legitimate nodes could be jammed, or even worse, nodes can be energy depleted.

- *Replay attacks*: The attacker captures messages exchanged between legitimate nodes and replays them in order to change the aggregation results.

2.4.2 Insider Attacks

Perhaps more dangerous from a security point of view is an insider attack, where an adversary by physically capturing a node and reading its memory, can obtain its key material and forge node messages. Having access to legitimate keys, the attacker can launch several kinds of attacks without easily being detected:

- *False data injection (stealthy attack)*: the attacker injects false aggregation results, which are significantly different from the true results determined by the measured values

- *Selective reporting*: the attacker stalls the reports of events that do happen, by dropping legitimate packets that pass through the compromised node.

Of course, an adversary cannot have unlimited capabilities. There is some cost associated with capturing, reverse-engineering and controlling a node. Therefore, we should assume that the adversary can compromise only a limited number of sensor nodes. This fact affects the design of security protocols, as it is easier to offer some protection against a few compromised nodes, but not for the case where a large portion of the network is in control of the attacker.

2.5 Routing Attacks against Sensor Networks

The goal of an attacker, either being insider or outsider, is to manipulate user data directly or trying to affect the underlying routing topology. What makes it even easier for her is the fact that most protocols for sensor networks are not designed having security threats in mind. As a consequence, deployments of sensor networks rarely include security protection and little or no effort is usually required from the side of the attacker to perform the attack.

We mentioned some simple attacks in the previous section. However, there are more sophisticated attacks that exploit specific characteristics of the routing protocols in order effect the topology and gain access to the routed information. These attacks are described analytically by Karlof and Wagner [Kar03].

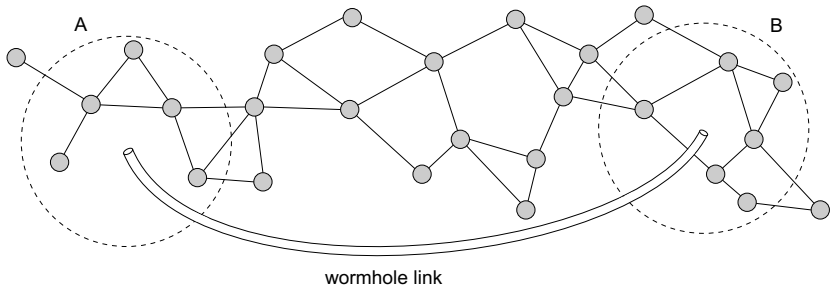

Figure 2.1: *A wormhole attack between two points in the network.*

2.5.1 The Sinkhole Attack

The sinkhole attack is a particularly severe attack that prevents the base station from obtaining complete and correct sensing data, thus forming a serious threat to higher-layer applications. In a sinkhole attack, a compromised node tries to draw all or as much traffic as possible from a particular area, by making itself look attractive to the surrounding nodes with respect to the routing metric. As a result, the adversary manages to attract all traffic that is destined to the base station. By taking part in the routing process, she can then launch more severe attacks, like selective forwarding, modifying or even dropping the packets coming through.

Recently we identified several vulnerabilities of two popular routing protocols of sensor networks, namely the MintRoute and the MultiHopLQI, and showed how they can be exploited by an attacker to launch a sinkhole attack [Kro08a]. It turns out that it is very easy for her to make the compromised node looks attractive to its neighbors or make them look less attractive and eventually make all nodes choose that node as their new parent.

2.5.2 The Wormhole Attack

The wormhole attack is a severe threat against packet routing in sensor networks that is particularly challenging to detect and prevent. To launch such an attack, an adversary establishes a low-latency link, referred as a wormhole link, between two points of the network, as shown in the figure. Once the wormhole link is operational, the adversary eavesdrops messages at one end and tunnels them (possibly selectively) to the other end, where the packets are retransmitted. The low-latency link used in this attack as well as any devices attached at each end of the link belong only to the attacker and are not compromised resources of the network. The link is realized in such a way that packets can travel from one end to the other faster than they would normally do via a multi-hop route in the network. The sensor nodes cannot detect the existence of such a link, as it can be realized with other means, such as a wired connection or an out-of-band wireless transmission.

As shown in the example of Figure 2.1, the net effect of the wormhole attack

is that the nodes within region A think they are neighbors with the nodes within region B and vice versa. If the attacker carefully chooses the place of the wormhole's end-points then it can use it to completely disrupt routing and attract a significant amount of traffic. So, if one end of the wormhole is close to the base station then nodes situated multiple hops away could be convinced that they are only one or two hops away. As a result, these nodes will choose to use the high-quality link for their transmission enabling other kind of attacks such as the sinkhole attack.

2.5.3 The Sybil Attack

A Sybil attack is one in which an attacker uses a malicious device to create a large number of pseudonymous entities, using them to gain a disproportionately large influence. We refer to a malicious device's additional identities as Sybil nodes. Newsome et al. [New04] introduce a taxonomy of the different forms of the Sybil attack in sensor networks. In terms of communication, Sybil nodes can communicate directly or indirectly with legitimate nodes. In the latter case, legitimate nodes are able to communicate with the Sybil nodes through the malicious device, which claims to be able to reach the Sybil nodes. Moreover, the malicious device can fabricate a new identity for a Sybil node, or it can steal an identity from a legitimate node. Finally, in terms of time, the attacker may try to have the Sybil identities participate in the network all at once or present a large number of identities over a period of time, while only acting as a smaller number of identities at any given time.

Sybil attack can be used against many protocols in sensor networks. In multipath routing, seemingly disjoint paths could in fact go through a single malicious node presenting several Sybil identities. If a geographic routing protocol is used [Dim06a], a Sybil node could appear in more than one place at once, instead of having one set of coordinates. In-network processing is also susceptible to Sybil attack. An attacker can affect aggregation results of sensor readings by contributing to the operation many times. I n the same way, she can affect a voting process amongst sensor nodes and make the system come to wrong conclusions. Therefore, Sybil attacks can pose a significant threat to the normal operation of a sensor network.

2.5.4 The HELLO Flood Attack

Many WSN protocols require nodes to broadcast HELLO packets for neighbor discovery purposes. After just a few messages have been exchanged, most nodes have a complete picture of their immediate vicinity and a routing topology logically forms in a self-organizing fashion. However, if a laptop-class attacker broadcasts such packets with large enough transmission power, she could convince every node in the network that the adversary is its neighbor and advertise attractive routing pathways through itself. After convincing portions of the network that it is truly the best routing option, it might choose to ignore incoming messages, effectively disabling large portions or even the entire network.

Unlike the rest of attacks we described so far, the HELLO flood attack does not require an attacking node to create legitimate traffic to be successful. So, for example, even an outsider attacker can capture legitimate "HELLO" messages as they breezed through the air and then forward them with a more powerful antenna. Those messages would reach other nodes well beyond the actual reach of a real sensor node's hardware. It's easy to see that this forwarding and redistribution leads to false network topologies and bogus routing information.

2.6 Typical Security Requirements

Usually in sensor networks there exists one or more base stations operating as data sinks and often as gateways to other networks. In general a base station is considered trustworthy, either because it is physically protected or because it has a tamper-resistant hardware. Concerning the rest of the network, we now discuss the standard security requirements (and eventually behavior) we would like to achieve by making the network secure [Sta02].

- *Confidentiality*: In order to protect sensed data and communication exchanges between sensor nodes it is important to guarantee the secrecy of messages. In the sensor network case this is usually achieved by the use of *symmetric* cryptography as asymmetric or public key cryptography in general is considered too expensive. However, while encryption protects against outside attacks, it does not protect against inside attacks/node compromises, as an attacker can use recovered cryptographic key material [Har05] to successfully eavesdrop, impersonate or participate in the secret communications of the network. Furthermore, while confidentiality guarantees the security of communications inside the network it does not prevent the misuse of information reaching the base station. Hence, confidentiality must also be coupled with the right control policies so that only authorized users can have access to confidential information.

- *Integrity and Authentication*: Integrity and authentication is necessary to enable sensor nodes to detect modified, injected, or replayed packets. While it is clear that safety-critical applications require authentication, it is still wise to use it even for the rest of applications since otherwise the owner of the sensor network may get the wrong picture of the sensed world thus making inappropriate decisions. However, authentication alone does not solve the problem of node takeovers as compromised nodes can still authenticate themselves to the network. Hence authentication mechanisms should be "collective" and aim at securing the entire network.

- *Availability*: In many sensor network deployments (monitoring fires, quality of water in reservoirs, protection against floods, battlefield surveillance, etc.), keeping the network available for its intended use is essential. Thus, attacks like denial-of-service (DoS) that aim at bringing down the network itself may have serious consequences to the health and well being of people.

However, the limited ability of individual sensor nodes to detect between threats and benign failures makes ensuring network availability extremely difficult. Additionally, it is important that the network still operates under such scenarios and that its operation degrades in a predictable and stable way despite the presence of node compromises or failures.

All this discussion suggests that it is necessary to develop networks that exhibit autonomic security capabilities, i.e., be resilient to attacks and have the ability to contain damage after an intrusion.

2.7 Issues in Sensor Network Security Research

A security architecture for sensor networks must integrate a number of security measures and techniques in order to protect the network and satisfy the desirable requirements we have outlined. In what follows we describe a comprehensive set of these components (and the techniques involved) that are currently under research in sensor networks. Some of these research issues are similar to those faced in traditional networks, only with some additional constraints; others are unique to sensor networks.

2.7.1 Key Establishment and Initial Trust Setup

One important component of sensor networks security is programmable and controlled group communication. Members leave and join the group according to some membership rules and follow the same behavior pattern within the group. When setting up a secure sensor network, one must be able to embed trust rules that govern the security level of group communications as well as the self-configuration nature of the network. This includes discovering new nodes and adding them in the group as well as identifying and isolating malicious ones. Eventually this translates in establishing cryptographic keys between the members of the group.

Key establishment protocols used in traditional networks are well studied but cannot be applied here due to the inherent limited capabilities (CPU power, memory, etc.) of sensor nodes. Moreover, key-establishment techniques need to scale to networks with tens of thousands of nodes. Simple solutions such as network-wide keys [Bas01] are not acceptable from a security point of view since compromising a single node leads to compromise of the entire network, leaving no margins for self-healing. On the other hand, having each node sharing a separate key with every other node in the network is not possible due to memory constraints.

Typically, the problem of initial trust setup can be solved by allocating to each sensor node a randomly selected subset from a pre-established set of keys [Esc02; Cha03b; Du03]. Then sensors can communicate securely if they have one or more keys in common. However, these techniques offer only "probabilistic" security as compromising a node may lead to security breaches in other parts of the network.

An alternative approach to pre-established keys is to have the nodes create their keys after the deployment of the network, using preloaded information. The keys are not randomly distributed, but rather they are localized; keys that appear in some part of the network are not used again. So, even if a node is compromised and its keys exposed, an adversary can have access only to a very small portion of the network centered around the compromised node.

In order for sensor nodes to be able to communicate safely using established cryptographic keys, a key refresh mechanism is also needed. In an autonomic scenario, re-keying is equivalent with self-revocation of a key when the network detects an intrusion or the lifetime of the key has expired. In order to keep the desirable security level intact, the network itself has to determine that rekeying is needed and initiate the appropriate mechanisms. Once the system has detected the compromised area, the response is to cut off the intruder and exclude it from any paths forcing the regeneration of new cryptographic keys between the honest nodes.

2.7.2 Resilience to Denial of Service Attacks

Adversaries can limit the value of a wireless sensor network through DoS attacks making it imperative to defend against them. DoS attacks can occur at multiple protocol layers [Woo02], from radio jamming in physical layer to flooding in transport layer, all with the same goal: to prevent the network from performing its expected function. Adversaries can involve malicious transmissions into the network to interfere with sensor network protocols and induce battery exhaustion or physically destroy central network nodes. More disastrous attacks can occur from inside the sensor network if attackers compromise some of the sensors themselves. For example, they could create routing loops that will eventually exhaust all nodes in the loop.

Determining that the network is subject to a DoS attack is a very challenging problem. Especially in large-scale deployments, it is hard to differentiate between failures caused by intentional DoS attacks and nominal node failures. An autonomic sensor network must be able to *monitor* the network traffic and look for suspicious patterns that match some possibly learned rules about what is normal or abnormal behavior [Ami08]. Then it can respond according to the type of the attack.

Potential defenses include techniques such as frequency hopping, spread spectrum communication [Pic82] and proper authentication. What is needed, however, is an autonomic coordinated response to defend against DoS attacks with a minimum latency between the detection and a coordinated response. One example could be the use of unaffected nodes to map the affected region and then route around the jammed portion of the network [Woo03b]. However, the protocols involved must be highly efficient so that they do not themselves become targets for energy depletion attacks.

Figure 2.2: *Connecting JTAG to Tmote Sky. With JTAG access, an adversary is capable of taking complete control over the sensor node.*

2.7.3 Resilience to Node Compromises

Due to the nature of their deployment, sensor nodes are exposed to physical attacks in which an attacker can extract cryptographic secrets or modify their code. Hartung et al. [Har05] demonstrate how to extract cryptographic keys from a sensor node using a JTAG programmer interface in a matter of seconds (see Figure 2.2). Also, Becher et al. [Bec06] evaluate different physical attacks against sensor node hardware and determine the amount of effort an attacker has to undertake to compromise a node. One defense against such attacks would be the use of more expensive tamper resistance hardware; however, this solution would increase the cost per sensor considerably, thus ruling out deployment of sensor networks with thousands of nodes. Moreover, trusting tamper resistant devices can be problematic [And96].

Recent advances in sensor networks research have shown that even without physical access, an attacker can still manage to modify the code running on the nodes, by exploiting memory-related vulnerabilities, e.g., buffer overflow [Goo07b; Goo07a]. This has been demonstrated for TinyOS 2.x on a Tmote Sky wireless sensor node, which uses the Texas Instruments MSP430 microcontroller. The way to inject code into the node is to craft a packet which – when copied over the stack – overwrites the return address with the address of the global copy of itself.

For Atmel's ATmega128, which follows the Harvard architecture and program memory is physically separated from data memory, it would be harder to execute the malicious code. But still, Gu and Noorani [Gu08] in parallel with Yang et al. [Yan08] demonstrated that it is possible to construct a worm which uses existing code in the program memory to replicate itself and propagate to other sensor nodes.

The use of Java on other platforms, like Sun SPOT or Sentilla's Jcreate node seems more secure as it provides built-in protections against code-based at-

tacks that would exploit array boundaries, unchecked cast, pointer arithmetics, etc. Also, the virtual machine checks incoming code compliance with the Java standards before execution [Pla08]. A few schemes have been recently developed to provide memory safety in TinyOS, like Safe TinyOS [Coo07] and Harbor [Kum07], which both work on the source codes of applications. But an attacker can find exploitable routines directly from assembly codes that are not included in applications and hence, malicious packets may still evade these schemes.

A prevention measure would be to use a diversified protection scheme, which diversifies data and code segments by creating different and obfuscated data and code segment for each node in the network [Ala06]. Therefore, the attackers' effort on compromising one node cannot save their efforts on compromising another node. This approach can also be used as a defensive mechanism against worm attacks [Yan08]. Combined with the fact that an adversary would have to capture a large percentage of the sensors in the same time interval, security of the network would be enforced.

On the other hand, a detection measure would be a mechanism that could effectively detect malicious code in sensor nodes and give an assurance that they are running the correct code. This is called *software-based code attestation* [Ses06; Par05; Sha05; Ses04]. For example, SWATT [Ses04] enables an external verifier to verify the code of a running system to detect maliciously inserted or altered code, without the use of any special hardware. This enables new intrusion-detection architectures, where other sensor nodes can play the role of the verifier and alert the rest of the network in case a compromised node is detected. This approach has been followed by a scheme proposed by Yang et al. [Yan07], which does not depend on response time measurement by mobile verifiers as in SWATT. Instead, neighbors of a suspicious node collaborate in the attestation process to make a joint decision.

The problem with the above approach is how the nodes can identify suspicious nodes, before they initiate any code attestation process. A relatively unexplored research area in sensor networks security is intrusion detection, which monitors the behavior of the nodes with respect to the network traffic and identifies suspicious nodes. The challenge is to turn this uncertainty into successful identification with high probability, if the nodes collaborate and exchange their views on the suspicious node, eliminating the need for more complex protocols, like software-based code attestation.

2.7.4 Routing Security

Packet routing is one of the most essential services in sensor networks, as it is used to exchange messages with sensor nodes that are outside of a particular radio range. Researchers have proposed several approaches for efficient routing, but rarely do they consider security as a central design parameter [Par06]. Usually a trusted environment is assumed, where all sensor nodes cooperate and no attacker is present. Securing such protocols is very important, since even a single compromised node could completely paralyze communication in the net-

work. For example, an attacker could inject malicious routing information into the network that results in routing inconsistencies. Protecting communication by authenticating the packets could defend against such attacks, but not against replaying legitimate packets, or other approaches that an attacker may take. An excellent discussion on many of the attacks on routing protocols is presented by Karlof and Wagner [Kar03].

Initially, research on secure routing in sensor networks focused on providing *intrusion-tolerant* security, which seeks to maintain a certain level of availability even in the face of attacks. For example, multipath routing [Gan01; Den04] has been proposed as such a solution. Redundant disjoint paths are used, so even if an intruder compromises a node, information can be routed by alternative paths. An alternative direction is the *prevention* approach, which seeks to harden the routing protocol against attacks by restricting participants' actions. This approach can be efficient and effective, but only for one specific attack at a time, and only for already known attacks. For example, it has been shown how a sensor network can defend against a HELLO flood attack [Ham06], a Sybil attack [New04; Zha05], wormhole attack [Zha05] or a DoS attack [Aga06].

The third and most recent direction for designing secure routing protocols is *detection*, where the nodes monitor the real-time behavior of their neighbors in order to detect malicious behavior. Then recovery and self-healing techniques can be used to eliminate malicious participants and to restore network functionality. Ideally, intrusion detection systems (IDSs) can detect both known security exploits and even novel attacks that are yet to be experienced.

2.7.5 Location Aware Security

Many applications of sensor networks require location information, not only for routing purposes, but also for determining the origin of the sensed information or preventing threats against services [Liu03; Laz03]. Many localization techniques have been proposed, but little research has been done in securing the localization scheme [Sas03; Laz05; Č06; Du06]. Security in this case is twofold: Each node must determine its own location in a secure way (secure localization) and each node must verify the location claim of another node (location verification).

Since providing each node with a GPS receiver increases its cost, many localization services assume the presence of a few such nodes (usually more powerful also), which communicate their coordinates in the network and allow the rest of the nodes to estimate their position. This communication provides malicious attackers with the chance to modify measured distances and make nodes believe that they are at a position which is different from their real one. Furthermore, without location verification mechanisms, a dishonest node can cheat about its own position in order to gain unauthorized access to some services, or avoid being penalized. As more and more protocols and services are based upon location awareness, enabling sensors to determine their location in an untrusted environment becomes essential.

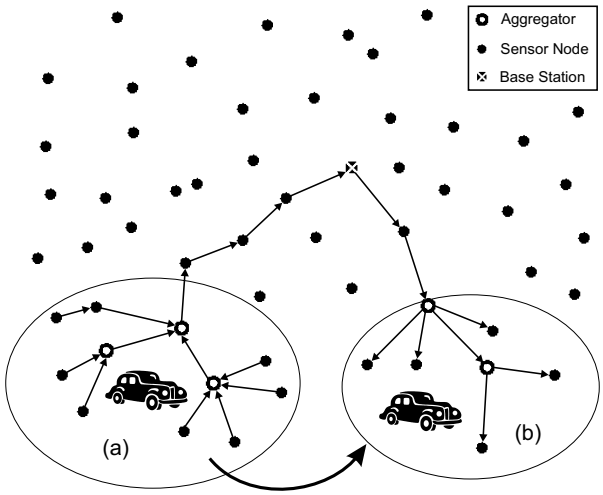

Figure 2.3: *A tracking application using both (a) aggregation and (b) dissemination.*

2.7.6 Secure Aggregation and Dissemination

Since sensor nodes have limited energy which may be exhausted, sensor networks are densely deployed to deal with connectivity and coverage problems. This causes neighboring nodes to have overlapping sensing regions and generate correlated measurements whenever an event occurs in this overlap. Moreover, sensor nodes are usually deployed randomly, which reinforces the effect of overlapping regions. Each node observes its sensing region independent of its neighbors and sends its measurements to the base station. Therefore, communication overhead can be substantially reduced, if raw data sent by nodes can be combined to eliminate redundancy and reduce the number of transmissions.

A technique that has been used to deal with these problems is data aggregation (or data fusion) [Int03; Kri02; Mad02]. The key idea is to combine highly correlated data coming from different sensors into one packet. This happens at intermediate nodes, called *aggregators*, which compute an aggregated value of all the measurements (e.g., an average or maximum temperature), and then forward only a single packet with the resulted value. The reverse process is called *data dissemination*, in which the network hierarchy is used in the reverse direction in order to disseminate control messages from the base station toward the aggregators and eventually toward the sensor nodes. For example, as shown in Figure 2.3, in tracking applications the sensor network may need to be used in both modes: first to aggregate sensed data about the movement of the tracked object and then to disseminate commands to nearby sensors to enable further tracking [Xu03].

In wireless sensor networks, the first step on providing security for data communication is the establishment of shared keys between pairs of nodes so that they can encrypt and authenticate data exchanged between them. However,

caution must be taken so that in-network processing is not hindered by the underlying security protocol. It is well-known that end-to-end data encryption is able to protect private communications between two parties, but it is not a good candidate for private data aggregation. This is because, if end-to-end communications are encrypted, the intermediate nodes cannot easily perform in-network processing to get aggregated results.

In particular, the following requirements must be supported by the key management scheme, in order to facilitate data aggregation and dissemination process:

1. Data aggregation is possible only if intermediate nodes have access to encrypted data so that they can extract measurement values and apply to them aggregation functions. Therefore, nodes that send data packets toward the base station must encrypt them with keys available to the aggregator nodes.

2. Data dissemination implies broadcasting of a message from the aggregator to its group members. If an aggregator shares a different key (or set of keys) with each of the sensor within its group, then it will have to make multiple transmissions, encrypted each time with a different key, in order to broadcast a message to all of the nodes. But transmissions must be kept as low as possible because of their high energy consumption rate.

One way to satisfy the above requirements is by grouping nodes into clusters, where a common key is shared between the nodes of the same cluster. Data are encrypted using that key, so aggregation and dissemination within the cluster is very efficient.

Here we assume, as in other existing secure data aggregation schemes [Prz03; Yan06; Bla06], that an intermediate aggregation node has to decrypt the received data, then aggregate the data according to the corresponding aggregation function, and finally encrypt the aggregated result before forwarding it. An alternative approach exists, where aggregation of encrypted data is possible without decryption in the aggregation nodes. This can be done by using homomorphic encryption ciphers, as proposed by Girao et al. [Wes06] and Castelluccia et al. [Cas05].

2.7.7 Link-layer Security

What makes link-layer security important is that end-to-end security mechanisms are not possible in sensor networks, so more transparent mechanisms provided by the link layer are needed. Protocols used in conventional networks for end-to-end security, such as SSH [Ylo96], SSL [ssl01], or IPSec [Ken98], even though they are feasible in constrained embedded devices [Gup05], they are considered inappropriate since they do not allow in-network processing and data aggregation which play an important role in energy-efficient data retrieval. These operations require the intermediate nodes to access and possibly modify

the contents of packets, which would not be possible if an end-to-end security scheme was used.

In sensor networks it is also important to allow intermediate nodes to check message integrity and authenticity, or else the network would be prone to several denial of service attacks. Using an end-to-end security mechanism, packets would have to be routed all the way to the base station before these checks could be performed, since the intermediate nodes do not have the keys to verify their authenticity and integrity. On the other hand, using a transparent security mechanism at the link layer, malicious packets can be identified and rejected at the first hop.

However, since the usual traffic pattern in WSN is many-to-one, pre-loading one-to-one keys between two sensors and refreshing the keys are practically impossible tasks. Public key cryptography is also considered to be computationally expensive for WSN and therefore, light-weight, yet reasonably secure key management schemes are crucial in order to bring about acceptable security services in WSN. In addition to this, any WSN security protocol has to be flexible and scalable enough to easily allow nodes to join or leave the network.

TinySec [Kar04] is a link-layer security architecture for wireless sensor networks that is part of the official TinyOS release. It generates secure packets by encrypting data packets using a group key shared among sensor nodes and calculating a MAC for the whole packet including the header.

A limitation of TinySec is that messages of less than 8 bytes are not addressed efficiently. This is because TinySec uses a k-byte block cipher to encrypt the message. For longer messages CBC mode is chosen that encrypts the message block by block. But it is not so unusual for a message (i.e., the payload of the TinyOS packet) to be less than 8 bytes, in which case TinySec will cause a ciphertext expansion, because ciphertext stealing requires at least one block of ciphertext. This kind of ciphertext expansion would cause extra communication power cost when sending data with variable length.

TinySec by default relies on a single key manually programmed into the sensor nodes before deployment. This network-wide shared key provides only a baseline level of security. It cannot protect against node capture attacks. If an adversary compromises a single node or learns the secret key, she can gain access on the information anywhere in the network, as well as inject her own packets. This is probably the weakest point in TinySec, since, as we saw in Section 2.7.3, node capture has been proved to be a fairly easy process.

More recent link-layer security protocols have used stronger keying mechanisms to deal with node capture attacks and provide better services to the upper layers [Kro08d]. Examples of such protocols include SenSec [Li05], SecureSense [Xue03], SNEP [Per02], MiniSec [Luk07] and L^3Sec [Sor07]. Some important features that these protocols exhibit include the resilience to node capture attack and scalability, both of which require that the framework is tied to an appropriate key management protocol. Another important feature is flexibility, which allows different types of security services for different types of communications among nodes.

2.7.8 Secure Network Programming

The process of programming sensor nodes typically involves the development of the application in a PC and the loading of the program image to the node through the parallel or the serial port. The same process is repeated for all the nodes of the sensor network before deployment. However, after deployment, there is often the need to change the behavior of the nodes in order to adapt to new application requirements or new environmental conditions. This would require the effort of re-programming each individual node with the updated code and relocate it back to the deployment site. Network programming saves this effort by propagating the new code over the wireless link to the entire network, as soon as that code is loaded to only one node. Then, nodes reprogram themselves and start operating with the updated code.

As network programming simplifies things for legitimate users, it also simplifies things for attackers that want to disrupt the normal operation of the network or operate them for their own advantage. In currently deployed networks the nodes do not authenticate the source of the program; therefore an attacker could easily approach the deployment site and disseminate her own malicious/corrupted code in the network.

This possibility makes sensor networks deployments susceptible to outsider attacks. Besides loosing control of the network or getting back altered measurements, it is even possible that the network is reprogrammed with malicious code that has the same functionality with the legitimate code but also reports data to the adversary. In such a case legitimate users would never know that something is wrong. Hence, it is important that the sensor nodes can efficiently verify that the new code originates from a trusted source, namely the base station.

The most natural solution for authenticated broadcasts is asymmetric cryptography, where messages are signed with a key known only to the sender. Everybody can verify the authenticity of the messages by using the corresponding public key, but no one can produce legitimate signed messages without the secret key. However, public key schemes should be avoided in sensor networks for multiple reasons: long signatures induce high communication overhead of 50 - 1000 bytes per packet, verification time places a lower bound on the computational abilities of the receiver, and so on.

However our goal is not to authenticate just messages, since here we are dealing with *streams*, rather than simple messages. The size of program images that will be sent over the radio is usually between a few hundreds of kilobytes and a few thousands. This fact can allow the use of public key schemes if we manage to reduce the size of the public key and also make signature size to be only a small percentage of the total transmitted stream. Furthermore, if we reduce the verification time down to the order of that of a symmetric scheme, we will have proved that public key cryptography is an attractive solution for such problems.

2.7.9 Efficient Cryptographic Primitives

Because sensor nodes have limited computational and storage capabilities, traditional security solutions are often too expensive for sensor networks. More research in this domain is necessary, especially in exploring the use of efficient asymmetric cryptographic mechanisms for key establishment and digital signatures as a means for leveraging trust in sensor networks and solving some of the problems mentioned above [Gau04; Lop06].

Recently, elliptic curve cryptography (ECC) has emerged as a promising alternative to RSA-based algorithms, as the typical size of ECC keys is much shorter for the same level of security. There have been notable advances in ECC implementation for WSNs in recent years. Uhsadel et al. [Uhs07] propose an efficient implementation of ECC and Liu et al. [Liu08a] developed TinyECC, an ECC library that provides elliptic curve arithmetic over prime fields and uses inline assembly code to speed up critical operations on the ATmega128 processor. Also lately, Szczechowiak et al. [Szc08] presented NanoECC, which is relatively fast compared with other existing ECC implementations, although it requires a heavy amount of ROM and RAM sizes.

Even though elliptic curve cryptography is feasible on sensor nodes, its energy requirements are still orders of magnitude higher compared to that of symmetric cryptosystems. Therefore, elliptic curve cryptography would make more sense to be used only for infrequent but security-critical operations, like key establishment during the initial configuration of the sensor network [Gro06].

Elliptic curve cryptography can also be used as a solution for secure code dissemination protocols in sensor networks, since this problem calls for asymmetric cryptography. When a new code is received by a node, the code should be authenticated by verifying its signature. Since this operation is performed at the mote, both energy efficiency (i.e., verification time) and code size are important parameters. Some proposed protocols, like Sluice [Lan06] and Seluge [Liu08b], use ECC for signature verification of the program images. However, this problem can be solved more efficiently with signing and verification times much closer to those of a MAC, eliminating the need for ECC.

2.8 Conclusions

In this chapter we have presented an overview of current research challenges on sensor networks security. A progress has been made in providing specialized security mechanisms, like key establishment, secure localization, secure aggregation or secure routing. Some of these mechanism are addressed by our research and presented in the following chapters.

According to what discussed in Sections 2.7.1 and 2.7.6, a localized distributed algorithm for key establishment in sensor networks that works well with data aggregation and dissemination is presented in Chapter 3. A scheme is proposed that utilizes the established keys in order to provide secure communication between a source node and the base station, while intermediate nodes

can access data and perform aggregation.

The challenges presented in Section 2.7.7 are addressed by Chapter 4, where we provide an efficient authentication scheme for a finite stream of data. This scheme is based on *symmetric* cryptography primitives while at the same time having the properties of asymmetric cryptography, eliminating the need for ECC, as we mentioned in Section 2.7.9. As we will see, such a scheme can make authenticating code updates (and data streams in general) very efficient for sensor networks.

While addressing the challenges presented in this chapter may protect sensor networks from specific threats, what has been lacking is a *holistic* approach that encompasses autonomic responses over a broad range of attacks. A research challenge therefore, would be the design of an adaptive security architecture that can monitor the sensor network, recognize a security threat and respond by a coordinated self-healing mechanism. Chapters 5 and 6 investigates this approach and describes an intrusion detection system that can offer opportunities for increasing sensor networks security and guaranteeing a robust and survivable solution.

Chapter 3

Key Management

3.1 Introduction

The first step toward protecting sensor data communications is the establishment of encryption keys among sensors so that they can set up secure communication links [Ç07]. As indicated by Hu et al. [Hu05], setting up pairwise keys for secure communications is necessary to secure routing in wireless ad hoc networks. However, given the limited processing power, storage, bandwidth and energy resources, it is widely considered that a sensor device cannot employ sophisticated cryptographic technologies such as public key cryptosystems. Instead, symmetric encryption/decryption algorithms and hashing functions are between two to four orders of magnitude faster [Car00; Cha03a], and constitute the basic tools for securing sensor networks communication.

Setting up pairwise symmetric keys among communicating sensors is an important step for *bootstrapping* [Ç07; Esc02] the mutual trust required for data exchange. However, caution must be taken so that network operations, e.g., in-network processing, are not hindered by the underlying security protocol. In particular, the following requirements must be supported by the key management scheme:

1. Data aggregation is possible only if intermediate nodes have access to encrypted data so that they can extract measurement values and apply to them aggregation functions. Therefore, nodes that send data packets toward the base station must encrypt them with keys available to the aggregator nodes.

2. Data dissemination implies broadcasting of a message from the aggregator to its group members. If an aggregator shares a different key (or set of keys) with each of the sensor within its group, then it will have to make multiple transmissions, encrypted each time with a different key, in order

to broadcast a message to all of the nodes. But transmissions must be kept as low as possible because of their high energy consumption rate.

So, the use of pair-wise shared keys between the nodes and the base station effectively hinders in-network processing. A solution that could satisfy the above requirement could be the use of a key common to all sensor nodes in the network [Bas01]. The problem with this approach is that if a single node is compromised then the security of the whole network is disrupted. Furthermore, refreshing the key becomes too expensive due to communication overhead. Therefore, we need a more sophisticates key management protocol that satisfies the above requirements, while it offers resiliency against node capture and replication.

In this chapter we present such a key management protocol for sensor networks. This protocol is composed by four phases. The first is the key distribution phase, where secret keys are pre-distributed to sensor nodes a priori to the deployment. In the second phase, sensor nodes discover their neighbors and establish common keys with them. This is achieved by using the pre-distributed keys and by exchanging messages directly over their insecure wireless links. In the third phase, each pair of neighboring nodes that do not have common keys, establish one or more keys. The fourth phase is the key update phase which supports the deployment of new sensor nodes and the update of the corresponding secret keys. The proposed scheme utilizes the established keys in order to provide secure communication between a source node and the base station, while intermediate nodes can access data and perform aggregation and dissemination.

In this way we ensure that communication is protected from external parties. In case of an insider attack, where a node is compromised, its secret keys are revealed. Even in this case, our protocol does not allow an adversary to gain control of other parts of the network by using this material. Therefore the compromise of nodes does not result in a breach of security and constrains the damage within a small, localized part of the network. With a proper intrusion detection system, like the one we propose in Chapter 6, the compromised area can be identified and the corresponding keys can be revoked.

3.1.1 Related Work

We have already mentioned Basagni et al.'s pebblenets architecture [Bas01] that uses a global key shared by all nodes. Having network wide keys for encrypting information is very good in terms of storage requirements and energy efficiency as no communication is required among nodes to establish additional keys. It suffers, however, from the obvious security disadvantage that compromise of even a single node will reveal the universal key. Since one cannot have keys that are shared pair-wise between all nodes in the network, a key pre-distribution scheme must be used.

Random key pre-distribution schemes [Esc02; Cha03b; Liu05; Du03] offer a trade-off between the level of security achieved using shared keys among nodes and the memory storage required to keep at each node a set of symmetric

keys that have been randomly chosen from a key pool. Then, according to the model used, nodes use one or more of these keys to communicate securely with each other. A necessary requirement of a good key-distribution scheme is that compromise of nodes should *not* result in a breach of security that is spread in the whole network. Our approach guarantees this since it's not a probabilistic scheme; compromised nodes cannot expose keys in another part of the network. In our work, however, we emphasize on another desirable characteristic: when a node wants to broadcast a message to a subset or all of its neighbors it should *not* have to make multiple transmissions of the same message, encrypted each time with a different key. We now review some other proposals that use security architectures similar to ours.

In LEAP [Zhu03] *every* node creates a cluster key that distributes to its immediate neighbors using pair-wise keys that shares with each one of them. In this case, however, clusters highly overlap so every node has to apply a different cryptographic key before forwarding the message. While this scheme offers deterministic security and broadcast of encrypted messages, it has a more expensive bootstrapping phase and increased storage requirements as each node must set up and store a number of pair-wise *and* cluster keys that is proportional to its actual neighbors.

Slijepcevic et al. [Sli02] propose dividing the network into hexagonal cells, each having a unique key shared between its members. Nodes belonging to the bordering region between neighboring cells store the keys of those cells, so that traffic can pass through. The model works under the assumption that sensor nodes are able to discover their exact location, so that they can organize into cells and produce a location-based key. Moreover, the authors assume that sensor nodes are tamper resistant, otherwise the set of master keys and the pseudo-random generator, pre-loaded to all sensor nodes, can be revealed by compromising a single node and the whole network security collapses. Those assumptions are usually too demanding for sensor networks.

3.1.2 Chapter Contribution

In this work we present a security protocol that has the following properties:

- *Resilience to Node Capture.* Our scheme offers deterministic security as a single compromised node disrupts only a *local* portion of the network while the rest remains fully secured. We designed our protocol without the assumption of tamper resistance. Once an adversary captures a node, key materials can be revealed.

- *Resistance to Node Replication.* Even if a node is compromised and be used to populate the network with its clones, an adversary cannot gain control of the network as key material from one part of the network cannot be used to disrupt communications to some other part of it.

- *Energy efficiency.* We enable secure communication between a node and its neighbors by requiring only *one* transmission per message. Thus mes-

sages do not have to be encrypted multiple times with different keys to reach all neighbors. This saves energy as transmissions are among the most expensive operations a sensor can perform [Per02].

- *Intermediate Node Accessibility of Data.* An effective technique to extend sensor network lifetime is to limit the amount of data sent back to reporting nodes since this reduces communications energy consumption. This can be achieved by some processing of the raw data to discard extraneous reports. However, this cannot be done unless intermediate sensor nodes have access to the protected data to perform *data fusion* processing. Although existing random key pre-distribution schemes provide a secure path between a source and a destination, nearby nodes cannot have access to this information as it is highly unlikely they possess the right key to decrypt data.
- *Scalability.* The number of keys stored in sensor nodes is *independent* from the network size and the security level remains unaffected.
- *Easy Deployment.* Our protocol enables a newly deployed network to establish a secure infrastructure quickly using only local information and total absence of coordination.

3.1.3 Chapter Outline

The organization of the rest of the chapter is as follows: In Section 3.2, we describe our security protocol. We break this discussion into multiple subsections to ease the readability and understanding of the protocol. Furthermore, we prove each of the claims we made in this introduction about the features of our protocol. In Section 3.2.4, we describe how new node addition can be supported and in Sections 3.3 and 3.4, we show how this protocol can be extended to secure the communication between nodes and their aggregators. In Section 3.5, we provide experimental evidence about the scalability of our protocol (in terms of the keys stored in each node) as well as the local resiliency (in terms of nodes in each cluster). In Section 3.6 we show that our protocol is secure against certain types of attacks. Finally we summarize our results in Section 3.7.

3.2 Security Protocol

In this section we first describe a localized algorithm for key management in wireless sensor networks (Sections 3.2.1 and 3.2.2) and then provide a scheme that utilizes the established keys in order to provide secure communication between a source node and the base station (Section 3.2.3). In general, our scheme can provide secure communication between *any* pair of nodes by building transitive keying relationships, but we demonstrate our protocol with respect to the "node to base station" data delivery model, since this is the most commonly used in sensor networks. The protocol is divided into the following phases:

1. Initialization phase that is performed before sensor nodes are deployed.
2. Cluster key setup phase that splits the network into disjoint sets (clusters) and distributes a unique key to each cluster. That key is shared between all the cluster members, as well as the nodes that are one-hop away from the cluster.
3. Secure communication phase that provides confidentiality, data authentication, and freshness for messages relayed between nodes toward the base station.

3.2.1 Initialization

Sensor nodes are assigned a unique ID that identifies them in the network, as well as three symmetric keys. Since wireless transmission of this information is not secure, it is assigned to the nodes during the manufacturing phase, before deployment. In particular the following keys are loaded into sensor nodes:

- K_i: Shared between each node i and the base station. This key will be used to secure information sent from node i to the base station. It is not used in establishing the security infrastructure of the network but only to encrypt the sensed data D that must reach the base station in a secure manner. If we are interested in data fusion processing this key should not be used to encrypt D as otherwise intermediate nodes will not be able to evaluate and possibly discard the data.

- K_c^i: Shared between each node i and the base station. This key will be used only by those nodes that will become clusterheads and it will be the cluster key. These are the keys used to forward information to the base station in a hop-by-hop manner.

- K_m: A master key shared among all nodes, including the base station. This key will be used to secure information exchanged during the cluster key setup phase. Then it is erased from the memory of the sensor nodes.

The base station is then given all the ID numbers and keys used in the network before the deployment phase. Since the base station stores information used to secure the entire network, it is necessary to include it in our trusted computing base.

3.2.2 Cluster Key Setup

We now describe how sensor nodes use the pre-deployed key material in order to form a network where nodes can communicate with each other using a set of trusted keys.

The cluster key setup procedure is divided into two phases: organization into clusters and secure link establishment. During the first phase the sensor nodes are organized into clusters and agree on a common cluster key, while in

the second phase, secure links are established between clusters in order to form a connected graph.

An implicit assumption here is that the time required for the underlying communication graph to become connected (through the establishment of secure links) is smaller than the time needed by an adversary to compromise a sensor node during deployment. As security protocols for sensor networks should *not* be designed with the assumption of tamper resistance [And96], we must assume that an adversary needs more time to compromise a node and discover the master key K_m. In the experimental section we give evidence that this is indeed the case.

Organization into clusters

In this phase, the creation of clusters happens in a probabilistic way that requires the nodes to make at most one broadcast. Each node i waits a random time (according to an exponential distribution) before broadcasting a HELLO message to its neighbors declaring its decision to become a cluster head. This message is encrypted using K_m and contains the ID_i of the node, its key K_c^i and an authentication tag:

$$E_{K_m}(ID_i||K_c^i||MAC_{K_m}(\langle ID_i||K_c^i\rangle)$$

Upon receiving a HELLO message, a node decrypts and authenticates the message. Then it reacts in the following way:

1. If the node has not made any decision about its role yet, it joins the cluster of the node that sent the message and cancels its timer. No transmission is required for that node. The key that it is going to be using to secure traffic is $K_c = K_c^i$.

2. If the node has already decided its role, it rejects the message. This will happen if the node has already received a HELLO message from another node and became a cluster member of the corresponding cluster, or the node has sent a HELLO message being a cluster head itself.

Upon termination of the first phase, the network will have been divided into clusters. All nodes will be either cluster heads or cluster members, depending on whether they sent a HELLO message or received one. We are assuming here that collisions are resolved at a lower level otherwise acknowledgments must be incorporated in this simple protocol. There is, however, the possibility that two neighboring nodes send HELLO before they receive the same message from their neighbor, thus becoming clusterheads of themselves. Furthermore, there is a case for a node to send a HELLO message after all its neighboring nodes have decided their role, and thus become a head of a cluster with no members. Although these possibilities can be minimized by the right exponential distribution of the time delays that nodes send the HELLO messages, they do not affect the proper running of the protocol. In Figure 3.1 we show the distribution of nodes to clusters for densities (average number of neighbors per sensor)

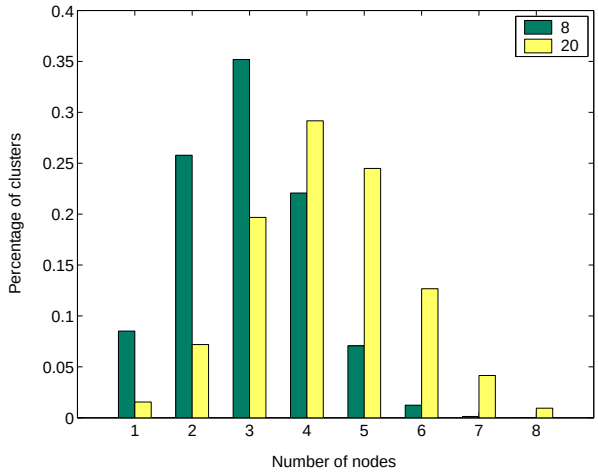

Figure 3.1: *Distribution of nodes to clusters.*

equal to 8 and 20. As it can be seen in the figure, for smaller densities a larger percentage of nodes forms clusters of size one. However, the probability of this event decreases as the density becomes larger.

At the end of this phase each cluster will be given an identifier *CID*, which can be the cluster head's ID. All nodes in a cluster will be sharing the same key, K_c, which is the key K_c^i of the cluster head. From this point on, cluster heads turn to normal members, as there is no more need for a hierarchical structure. This is important since cluster based approaches usually create single points of failure as communications must usually pass through a clusterhead. Figure 3.2 shows an example topology where three clusters have been created with *CID*s 13, 9 and 19.

As it can be seen from Figure 3.2, the maximum distance between two nodes in a cluster is two hops. Since all nodes in a cluster share a common key K_c, we need to keep the size of the clusters as small as possible in order to minimize the damage done by the compromise of a single node. In the experimental section, we give evidence that indeed clusters contain in average a small number of nodes that is independent of the size of the network.

Secure link establishment

In the second phase, all nodes get informed about the keys of their neighboring clusters. We need this phase in order to make the whole network connected since up to this point it is only divided into clusters whose nodes share a common key. We say that a node is *neighbor* of a cluster *CID* when that node has within its communication range at least one member of that cluster. This phase is executed with a simple local broadcast of the cluster key by all nodes. The

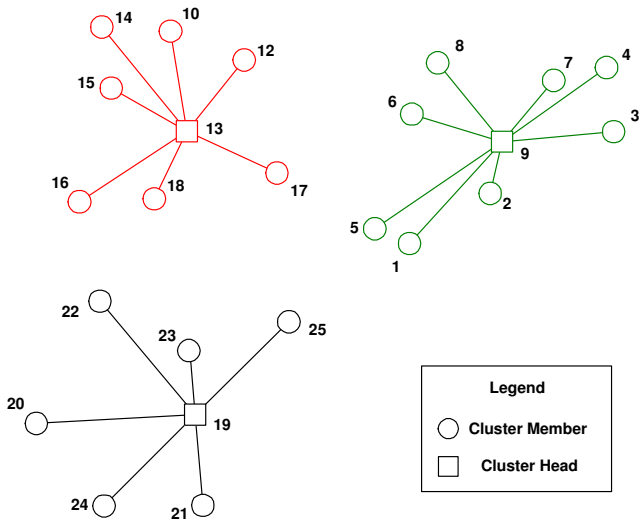

Figure 3.2: *An example topology after organization into clusters*

message sent contains the tag and the *CID*, encrypted using K_m:

$$E_{K_m}(CID_i||K_c||MAC_{K_m}(\langle CID_i||K_c\rangle))$$

Nodes of the same cluster simply ignore the message, while any nodes from neighboring clusters will store the tuple $\langle CID, K_c\rangle$ and use it to decrypt traffic coming from that cluster, as explained in the next section. If the message has been sent from a member of the same cluster, then that message should be ignored.

We must emphasize again that the total time of both steps is too short for an adversary to capture a node and retrieve the key K_m (see also Figure 3.10 in Section 3.5 for a justification of this claim.) Nevertheless an adversary could have monitored the key setup phase and by capturing a node at later time it could retrieve all cluster keys. Therefore after the completion of the key setup phase, all nodes erase key K_m from their memory.

At this point, each node i of the sensor network will have its key K_i and a set $\mathcal{S}$ of cluster keys that includes its own cluster key and the keys of its neighboring clusters. The total number of the keys that a node will have to store depends on the number of its neighboring clusters, thus not all cluster members store the same number of keys. (In the experimental section we give evidence that each node needs to store on average a handful of cluster keys). Most importantly however, the number of keys that each node gets is *independent* of the network size and therefore there is no upper limit on the number of sensor nodes that can be deployed in the network.

We illustrate the operations of the cluster key setup phase with the following example. Consider the sensor network depicted in Figure 3.3. Three clusters

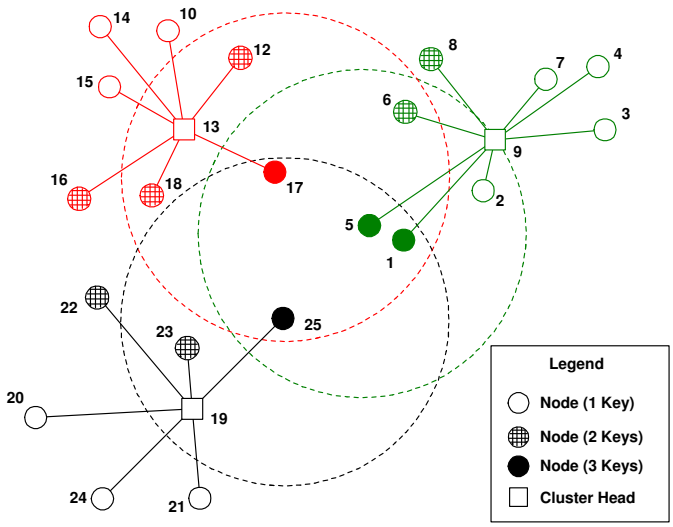

Figure 3.3: *An example topology during the key setup phase.*

with *CID*s 13, 9 and 19 have been formed from the first step. The figure also shows the transmission radius of nodes 25, 17 and 5. As it can be seen, node 25 has two neighboring clusters, since node 17 from cluster 13 and nodes 5 and 1 from cluster 9 are within its communication range. Therefore node 17 will store 3 cluster keys. Likewise, nodes 17, 5 and 1 also have two neighboring clusters and will store 3 cluster keys each. On the other hand, node 6 is within the range of node's 17 but outside the radius of any node from cluster 19, therefore it will only store 2 cluster keys.

3.2.3 Secure Message Forwarding

In this section we describe how information propagating toward a base station can be secured to guarantee confidentiality, data authentication, and freshness. *Confidentiality* guarantees that sensor readings and other sensitive information (e.g. routing information) are protected from disclosure to unauthorized parties. This can be achieved by encrypting data with a secret key. *Data authentication* allows the base station to verify that data was really sent by the claimed sender and not from an adversary that injected it in the network. Even if messages cannot be inferred or forged, an adversary can capture them and replay them at a later time (message replay attack). *Data freshness* ensures that messages are fresh, meaning that they obey a message ordering and have not been reused.

Here we make the assumption that sensor readings must first be encrypted (Step 1 in the description below) and then authenticated in a hop-by-hop manner (Step 2) as data is forwarded to the base station through intermediate nodes. If we are interested in data fusion processing then Step 1 should be omitted. It

is only used when we want to make sure that sensor readings can only be seen by the base station.

It is worth mentioning here that while we describe our protocol for the case that a sensed event must be securely forwarded to a base station, *the same ideas can be used with any routing protocol that requires secure communication between one-hop neighbors.* The key translation character of our protocol is very important in establishing a transitive chain of trust.

Step 1 (Optional)

To achieve the security requirements for the data D that will be exchanged between the source node and the base station, we use a SNEP [Per02] like exchange, as shown in Figure 3.4. A good security practice is to use different keys for different cryptographic operations; this prevents potential interactions between the operations that might introduce weaknesses in a security protocol. Therefore we use independent keys for the encryption and authentication operations, K_{encr} and K_{MAC} respectively, which are derived from the unique key K_i that node shares with the base station. For example we may take $K_{encr} = F_{K_i}(0)$ and $K_{MAC} = F_{K_i}(1)$, where F is some secure pseudo-random function.

$$\begin{array}{lcl} y_1 & \leftarrow & E_{K_{encr},C}(D) \\ t_1 & \leftarrow & MAC_{K_{MAC}}(y_1) \\ c_1 & \leftarrow & y_1||t_1 \end{array}$$

Figure 3.4: *Step 1 for secure communication between source node and base station. This step is applied by the source node alone.*

As it can be seen from Figure 3.4, we use the encrypt-then-authenticate method to construct a "secure channel" between source node and base station. It is shown by Krawczyk [Kra01] that this is the most secure method for symmetric encryption and authentication. Encryption is performed through the use of a counter C that is shared between the source node and the base station. We do this in order to achieve semantic security; an adversary will not be able to obtain partial information about a plaintext, even if it is the same plaintext that is encrypted multiple times. This can also be achieved through randomization but then the random value used in the encryption of the message must also be transmitted. The counter approach results in less transmission overhead as the counter is maintained in both ends. If counter synchronization is a problem (usually the receiver can try a small window of counter values to recover the message) then the counter or the random value used can be sent alongside the message. We leave the choice to the particular deployment scenario as one alternative may be better than the other.

Step 2 (Required)

Since the encrypted data must be forwarded by intermediate nodes in order to reach the base station, we need to further secure the message so that an adversary cannot disrupt the routing procedure. Thus, no matter what routing protocol is followed, *intermediate* nodes need to verify that the message is not tampered with, replayed or revealed to unauthorized parties, before forwarding it.

To secure the communication between one-hop neighbors, we use the protocol described in Figure 3.5. Each node (including the source node) uses its cluster key to produce the encryption key K'_{encr} and the MAC key K'_{MAC}. These keys are used to secure the message produced by Step 1, before it is further forwarded. (As we emphasized previously, if we are only interested in hop-by-hop encryption and authentication, Step 1 should be omitted in which case c_1, in message 2 below, is simply the data D.) Since the nodes that will receive that message do not know the sender and therefore the key that the message was encrypted with, the cluster ID is included in c_2. This way intermediate sensors will use the right key in their set $\mathcal{S}$ to authenticate the message.

$$
\begin{array}{lcl}
\tau & \leftarrow & \text{time}() \\
y_2 & \leftarrow & E_{K'_{encr}}(c_1, \tau, CID) \\
t_2 & \leftarrow & MAC_{K'_{MAC}}(y_2) \\
c_2 & \leftarrow & CID||y_2||t_2
\end{array}
$$

Figure 3.5: *Step 2 for secure communication between source node and base station. This step is applied by all intermediate nodes, besides the source node.*

If authentication is not successful, the message should be dropped since it is not a legitimate one. Otherwise, each node will apply Step 2 with its own cluster key to further forward the message. The fact that this key is shared with all of its neighbors, allows the node to make only *one* transmission per message. Notice that this is the point where our protocol differs from random key predistribution schemes. To broadcast a message in such a scheme the transmitter must encrypt the message multiple times, each time with a key shared with a specific neighbor. And this, of course, is extremely energy consuming.

To continue the example shown on Figure 3.3, assume that node 14 must send a message m toward the base station that lies in the direction of node 4. It first encrypts and tags the message to produce a ciphertext c_1 according to the protocol shown on Figure 3.4 and then wraps this to produce an encrypted block c_2 according to the specifications shown on Figure 3.5. When ready, it broadcasts c_2 to its neighbors. Eventually an encapsulation of c_1 will reach node 12, maybe through node 10. This node will decrypt and authenticate the message since it shares the same cluster key as node 14 and once all the checks are passed, it will re-encrypt c_1 and forward it to its neighbors. One of them is node 8 which is a member of cluster with $CID = 9$. This node will look at

its set of cluster keys $\mathcal{S}$ and use the one which it shares with node 12 (the one corresponding to *CID*= 13). Upon success it will re-encrypt the message with its cluster key and forward it toward its neighbors. Thus nodes that lie at the edge of clusters will be able to "translate" messages that come from neighboring clusters and be able to *authenticate* them in a hop-by-hop manner.

In summary, every node decrypts and authenticates the packets it receives by using the keys it derived from each cluster key. If the node belongs to a different cluster than the transmitter of the message, it will find the right cluster key from those stored in its memory and use it to re-encrypt the message and pass it along to its neighbors. Thus messages get authenticated as they traverse the network. However, while this approach defends against adversaries who do not possess the required cluster keys, it falls prey to insider attacks since an adversary can easily inject spurious messages after it has compromised a node and discovered its cluster key. Unfortunately this kind of attack is not only hard to prevent but is also hard to detect.

To increase security and avoid sending too much traffic under the same keys, cluster keys may be refreshed periodically. To support such functionality, sensor nodes can repeat the key setup phase with a predefined period in order to form new clusters and new cluster keys. Since K_m is no longer available to the nodes, the current cluster key may be used by the nodes instead. The fact that each node can communicate with all of its neighbors using the current cluster key makes it possible to broadcast a HELLO message in a secure way. The message will contain the new cluster key, created by a secure key generation algorithm embedded in each node. Since the key setup phase requires very low communication overhead (as it will be showed in the next section) and takes only a short time to complete, the refreshing period can be as short as needed to keep the network safe. Alternatively, if we do not like the fact that certain nodes are assigned the task of creating new keys (as they may be the compromised ones), we can renew the cluster keys by periodically hashing these keys at fixed time intervals.

3.2.4 Addition of New Nodes

This section address the problem of refreshing the network as sensors usually have limited lifetime and usually die of energy depletion. We assume that new sensors are arbitrary deployed. As they cannot be preassigned to a specific cluster, they must i) associate themselves to an existing cluster, ii) become informed about neighboring clusters and iii) retrieve and store the corresponding cluster keys. Each new node comes equipped with a master key K_{MC} that can be used to generate the relevant cluster keys as it will be explained below.

Every new node transmits a hello message to its neighbors indicating its will to become a member of some existing cluster. The message contains the ID of the new node. Nodes receiving this message will respond with the cluster ID they belong to, authenticated using their cluster key K_c. This is necessary in order to prevent an adversary for realizing the following attack. The adversary may send fake messages containing various cluster IDs. When the new node

makes the association between the cluster ID and the cluster key and store it in its memory, the adversary can later compromise the node thus having acquired the cluster key of any cluster in the network. To prevent this type of impersonation attack the response sent by existing nodes is simply

$$CID, MAC_{K_C}(CID).$$

A new node receiving such a collection of cluster ID's will consider itself a member of the first such cluster while the rest will be the neighboring ones. We need now a way to associate each *CID* with the corresponding cluster key. We assume here that the cluster keys K_c^i of the original nodes were formed using a master key K_{MC} through the application of some pseudorandom function F. The use of a secure one way function F will prevent an adversary who compromised a node and found its cluster key to recover the master key K_{MC} and hence the cluster keys of other nodes. Using F, the cluster key of the i-th node is simply given by

$$K_c^i = F(K_{MC}, i).$$

Each new node can use K_{MC} to generate the various cluster keys and store them in its memory. Then it can participate in encrypting and forwarding messages just like the original nodes. When this phase is over, the master key K_{MC} is deleted from the memory of the nodes.

3.3 Key Establishment for Secure Aggregation

The protocol presented in the previous section uses clustering only for the key establishment phase. After that phase, communication does not use any hierarchical model. In large sensor networks however, where more than one aggregator exist, there is the need to use a hierarchical aggregation model, where aggregation nodes form a tree.

So now let us assume that the network is partitioned into distinct clusters and that each cluster is composed of an aggregator and a set of sensor nodes (distinct from other sets), which gather information and transmit it to the aggregator of their cluster. The aggregator fuses the data from the different sensors, performs mission-related data processing, and sends it to the base station. We do *not* assume that the aggregators are more powerful in terms of energy, memory or computational resources. Aggregators may form multiple levels of aggregation hierarchies encompassing any number of sensor nodes where an aggregator can both aggregate data as well as disseminate commands. Based on this model, we present an efficient mechanisms for establishing trust between the aggregators and the sensor nodes and provide for secure in-network processing.

Each sensor node, S_i, has a unique key denoted K_{S_i} which is computed before deployment as follows:

$$K_{S_i} = F(K_m, S_i),$$

where F is a secure pseudo-random function, K_m is a master key and S_i is the unique identifier (number) for the i-th sensor node. This key will be shared between the sensor node and its aggregator and will be used to exchange data securely. Notice that an adversary upon compromising node S_i cannot recover the master key K_m from the key K_{S_i} because of the one-wayness of F. Hence the rest of the network remains secured.

Each aggregator sensor node, A_j, has K_m stored in memory along with an individual key K_{A_j} which is derived from

$$K_{A_j} = F(K_m, A_j),$$

as described above. We will see later on that K_m is kept by A_j for as long it is necessary to establish secret keys with the nodes belonging in the A_j's group. Then it is *deleted* from the memory of the aggregator.

The base station holds K_m along with the keys of each sensor/aggregator node in the network.

As we said, each sensor node is preloaded with the key K_{S_i} which is the result of the application of a secure pseudo-random function on the master key K_m. In order for a node to communicate with its aggregator, this key must be available in both sides. So, the node must first inform the aggregator about its key in a secure way. This can happen by sending an appropriate HELLO message M_{Hello}:

$$S_i \rightarrow A_j : M_{Hello}, MAC_{K_{S_i}}(M_{Hello})$$

where the M_{Hello} consists of the sensor's id S_i and a nonce N_{S_i} computed by the sensor.

Having received this message, the aggregator A_j is now able to compute K_{S_i} using the master key K_m and authenticate it by checking the MAC. If the MAC verifies, the sensor node is included in the cluster and the aggregator sensor stores all relevant information (such as K_{S_i} and nonce identifier to avoid replay attacks, etc.). Additionally the aggregator may send back a reply to acknowledge the inclusion in the cluster (also authenticated with K_{S_i}).

The same procedure is repeated for every node with its aggregator, in an initial phase after the deployment of the network. This phase is secure as long as the adversary does not know the master key. Therefore, we must assume that the phase is too short in time for an adversary to capture a node and retrieve the master key K_m.

The memory requirements of this phase are determined by the number of keys each aggregator has to store, i.e., the number of nodes in each cluster. Figure 3.6 shows the average number of keys each aggregator has to store as a function of the number of aggregators in the network. As expected, the more the aggregators in the network, the more the clusters that will be formed and therefore, the less the average number of sensor nodes in each cluster.

After the completion of the phase, the master key is *deleted* by the memory of every node. Since the security of our protocol depends on the deletion of the master from the memory of the sensor nodes, we should take care that

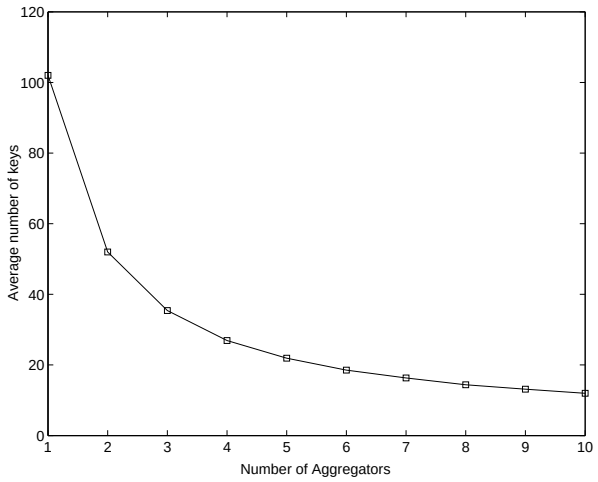

Figure 3.6: *Average number of keys an aggregator has to store in a random network of 1000 nodes.*

the deletion is unrecoverable, for example by overwriting the master key (in practice several times). We have now established a secure channel between the aggregators and the sensor nodes.

3.4 Key Establishment for Secure Dissemination

Having secured the communication of sensor nodes toward their aggregators, we now need to secure the reverse procedure: the dissemination of messages *from* the aggregators *to* the sensor nodes. One simple but inefficient solution would be to send a separate unicast message to each member of the group encrypted and authenticated using the key shared between the aggregator and the specific node. This would result in several transmissions of the same message, wasting energy resources.

The solution to this problem is for the aggregators to construct and propagate a group key to its members during the initial phase after deployment. In this way, dissemination can take place later by a single broadcast of the message encrypted by that key. Each aggregator A_i constructs and sends a group key G_{K_i} to each sensor S_j that belongs to its group:

$$\begin{aligned} A_i: \quad & c = E_{K_{encr}}(\text{“Group key”}, A_i, G_{K_i}), \\ & \sigma = MAC_{K_{MAC}}(c) \\ A_i \rightarrow S_j: \quad & A_i, c, \sigma \end{aligned}$$

The group key is encrypted and authenticated each time using the sensor's private key K_{S_i}, which is available to the aggregator as described in the previous

section. First the aggregator encrypts the group key using the key K_{encr} derived from K_{S_i} and then creates a MAC σ of the resulting ciphertext c.

Once the aggregators settle a group key G_{K_i} with their group members, they can broadcast commands encrypted and authenticated using G_{K_i}. This is sufficient to secure communication from an outsider who does not hold the group key. However, an insider adversary, who has captured a group node and retrieved G_{K_i}, can impersonate the aggregator and send forged messages to nodes in the same group. Therefore we need to secure further the dissemination process.

3.4.1 Defending Against Impersonation Attacks

We enhance the security protocol described above, in order to ward off insider attacks that impersonate the aggregator and try to disseminate messages to the group. The solution is that whenever an aggregator A_i has a new command to disseminate to the nodes, it attaches to it the *next* key, K_l, from a one-way key chain, as follows:

$$\begin{aligned} A_i: \quad & c = E_{G_{K_i}}(\text{“Command”}, A_i, K_l), \\ & \sigma = MAC_{G_{K_i}}(c) \\ A_i \rightarrow Group: \quad & A_i, c, \sigma \end{aligned}$$

So, the l-th command sent by the aggregator to the group nodes contains the l-th commitment of the hash chain and is encrypted and authenticated using keys derived from G_{K_i}.

One-way key chains are a widely-used cryptographic primitive. To generate a chain of length n we randomly pick the last element of the chain K_n. Then, each element of the chain is generated by repeatedly applying a one-way function F, until the K_0 element, which is the commitment to the whole chain. We then reveal the elements of the chain in reverse order,

$$K_0, K_1, \ldots, K_l, \ldots, K_{n-1}, K_n.$$

If we know that K_{l-1} is part of the chain, we can verify that K_l is also part of the chain by checking that $K_{l-1} = F(K_l)$. Therefore, a node receiving a command encrypted with the group key can verify its authenticity by checking whether the new commitment K_l generates the previous one through the application of F. When this is the case, it replaces the old commitment K_{l-1} with the new one in its memory and accepts the command as authentic. Otherwise it rejects it.

One issue with the one-way key chain is that it limits the length of the trust delegated to an aggregator. This is because the length of the chain determines the number of packets that the aggregator can send to its sensor group members. That is, for a chain of length n the aggregator can send at most $n-1$ separate commands at the nodes. An aggregator can renew its key chain as follows: before the aggregator uses the last commitment, it creates a new hash chain $K'_0, K'_1, \ldots, K'_{n-1}, K'_n$ and broadcast a “renew hash chain” command which

contains the new commitment K_0' authenticated with the last unused key of the old chain. This essentially provides the connection between the two chains and the group nodes will be able to authenticate commands as before.

So far we have limited the possibility of an impersonation attack, but we have not eliminated it. There is still a scenario were an adversary can *jam* communications to a sensor S_j so that it misses the last k commands and hence commitments. Then it introduces new commands by "recycling" the unused commitments. It will be impossible for sensor S_j to notice the faked commands as the ordering of commitments is followed.

In order to defend against an attack like this we may assume that sensors are loosely time synchronized and commands are issued only at regular time intervals. That means a packet broadcast from the aggregator at time slot t contains commitment K_t. If a sensor node does not receive anything within the next d slots and the last commitment was K_t, it will expect to see the commitment K_{t+d} that accounts for d missing commands. In this way it cannot be misled and authenticate unused commitments.

Finally, we must note that an implicit defense against impersonation attacks that not only eliminates but also helps detect these types of attacks is to have the sensor nodes respond to the aggregator using the shared key K_{S_i}. In this manner, even if an attacker has issued false commands, it will not be able to use the information sent by the sensor nodes. Additionally, if the aggregator receives responses to commands that it has not issued, it will become aware that an attacker has compromised some nodes in the cluster and eventually take corrective actions.

3.5 Experimental Analysis

We simulated a sensor network to determine some parameter values of our scheme. For this purpose we used the SensorSimII [Ulm00], a simulator framework written in Java. We deployed several thousands of nodes (2500 to 3600) in a random topology of uniform distribution and ran the key setup phase. Of particular interest is the scalability, the communication overhead and memory requirements of our approach.

The storage requirements of our approach are determined by the number of cluster keys stored in each node. We performed experiments with various network sizes and we found that the curves matched exactly (modulo some small statistical deviation). So, we experimented with respect to network *density*. Figure 3.7 shows the average number of cluster keys that each node stores as a function of the average number of neighbors per node (density of network). The number of cluster keys also indicates the number of neighboring clusters that each node has. As is obvious from the figure, the number of stored keys is very small and increases with low rate as the number of neighbors increases, requiring negligible memory resources from the sensor node. The point to be made is that the number of required keys remains *independent* of the actual network size. Thus our protocol behaves the same way in a network with 2000

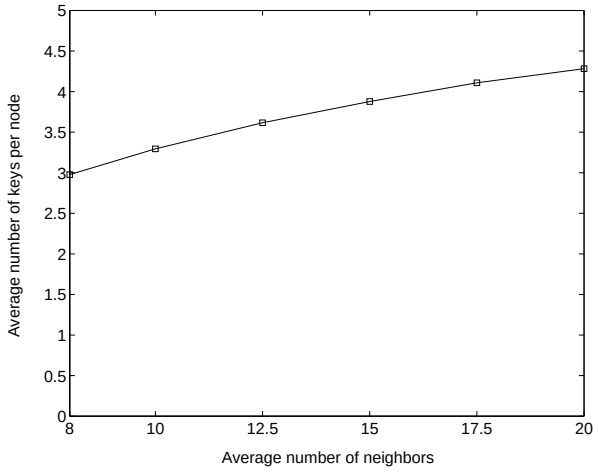

Figure 3.7: *Average number of cluster keys held by sensor nodes as a function of network density.*

or 20000 nodes.

In Figure 3.8 we further show the average number of nodes per cluster for various network densities. Nodes of the same cluster share a common cluster key, and thus an adversary, upon compromising such a node, can also control the communication links of the rest of cluster nodes. Thus, having small clusters, as is indicated in the figure, minimizes the damage inflicted by the compromised node and prevents its spreading to the rest of the network.

The communication traffic required by the key setup phase is partly due to the number of messages sent by the cluster heads to their cluster members during phase one, and partly due to the messages sent by all nodes of the network during the link establishment phase. The former quantity depends on the number of clusterheads and is shown in Figure 3.9. The second quantity, is always constant and equal to n, the number of nodes in the network. Bearing in mind that the key setup phase is executed only once, the total communication overhead due to that phase is kept very low. Further evidence to this fact is given in Figure 3.10, where the average number of messages required *per node* to set up the keys is shown. Thus the overall time needed to establish the keys is a little more than transmission of one message plus the time to decrypt the material sent during this phase.

3.6 Security Analysis

We now discuss one by one some of the general attacks [Kar03] that can be applied to routing protocols in order to take control of a small portion of the network or the entire part of it.

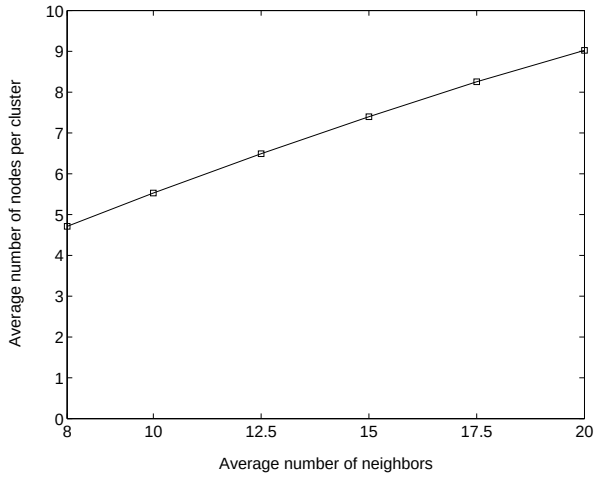

Figure 3.8: *Average number of nodes in clusters as a function of network density.*

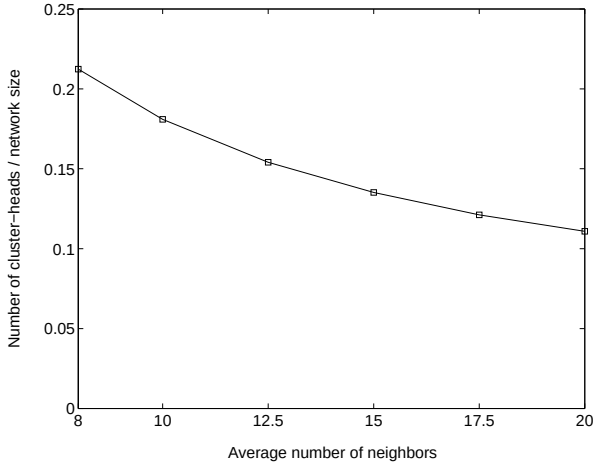

Figure 3.9: *Percentage of cluster heads with respect to total sensor nodes in the network.*

- *Spoofed, altered, or replayed routing information.* As sensor nodes do not exchange routing information, this kind of attack is not an issue.
- *Selective forwarding.* In this kind of attack an adversary selectively forwards certain packets through some compromised node while drops the rest. Although such an attack is always possible when a node is compromised, its consequences are insignificant since nearby nodes can have access to the same information through their cluster keys.

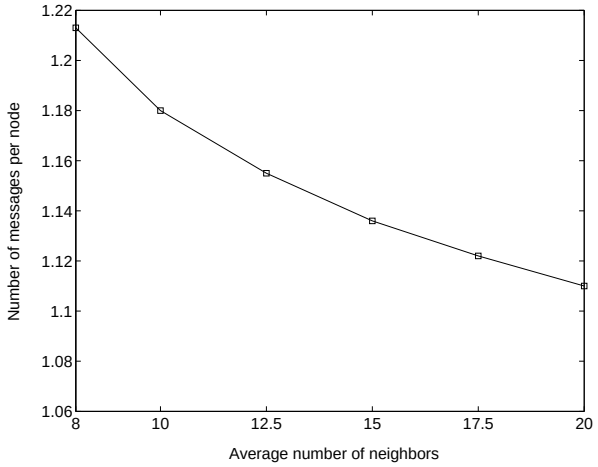

Figure 3.10: *Number of messages exchanged* per node *for organization into clusters and link establishment in a network of 2000 nodes and various densities.*

- *Sinkhole and wormhole attacks.* Since all nodes are considered equal from the key management protocol and there is not a distinction between more powerful and weak nodes, an adversary cannot launch attacks of this kind. Furthermore, in our protocol such an attack can only take place during the key establishment phase. But the authentication that takes place in this phase and its small duration, as we described in the previous section, makes this kind of attack impossible.

- *Sybil attacks.* Since every node shares a unique symmetric key with the trusted base station, a single node cannot present multiple identities. An adversary may create clones of a compromised node and populate them into the same cluster or the node's neighboring clusters but this does not offer any advantages to the adversary with respect to the availability of the information to the base station.

- *Hello flood attacks.* In our protocol, nodes broadcast a HELLO message during the cluster key setup phase in order to announce their decision to become clusterheads and distribute the cluster key. Since, however, messages are authenticated this attack is not possible. (A necessary assumption for all key establishment protocols is of course that the duration of this phase is small so that an adversary cannot compromise a node and obtain the key K_m. In the previous section we presented evidence that this is indeed the case).

  However, this kind of attack is possible during key-refresh. If we assume that a laptop-class attacker has compromised a node and retrieved its cluster keys then she could broadcast such a HELLO message during a key

refresh phase and could attract nodes belonging to neighboring clusters as well and form a new larger cluster with himself as a clusterhead. One way to defend against this is to constraint the key-refresh phase within clusters, i.e., not allow new clusters to be created. Therefore, cluster keys will be refreshed within the same clusters, and an adversary cannot take control of more nodes than she already has, that is the nodes within the same cluster. A better way, however, which makes this kind of attack useless, is to refresh the keys by hashing instead of letting nodes generate new ones.

- *Acknowledgment spoofing.* Since we do not rely on link layer acknowledgements this kind of attack is not possible in our protocol.

The power of our proposal lies exactly in its simplicity. No location information, path reinforcements or routing updates are used. By strengthening security with encryption and authentication, it becomes really hard for an adversary to disrupt the routing procedure.

3.7 Conclusions

We have presented a key establishment protocol that is suitable for sensor network deployment. The protocol provides security against a large number of attacks and guarantees that data securely reaches the base station in an energy efficient manner. Our protocol is based on hop-by-hop encryption, allowing nodes to share keys only with neighboring nodes. The protocol has a number of important characteristics among which are:

- Resiliency against node capture and replication. This is due to the fact that keys are localized. After a deployment phase, nodes share a handful of keys to securely communicate with their neighbors. Thus compromised keys in one part of the network do not allow an adversary to obtain access in some other part of it.

- Efficient broadcasting of *encrypted* messages. When a node wants to broadcast a message to its neighbors it does not have to make multiple transmissions encrypted each time with a different key. We achieve this by encrypting messages with a cluster key which is shared between neighboring nodes. This makes our scheme very energy efficient.

- Intermediate node accessibility of data. When multiple nodes receive the same message, some of them may decide not to forward it. However, this is not possible unless nodes can have access to encrypted data. Using our approach, nodes can "peak" at encrypted information using their cluster key and decide upon forwarding or discarding redundant messages thus enabling data aggregation processing.

- Scalability. Our protocol scales very well as the key establishment phase requires only local information and no global coordination. Furthermore the keys that need to be stored at each node do not depend on the size of the sensor network but only on its *density* (the average number of neighbors per node). Thus our protocol behaves similarly in networks of 2000 or 20000 nodes as long as the density is the same.

The keys established by our protocol can be used to secure messages sent by the sensor nodes toward the base station, or single command messages disseminated by the base station toward the nodes. However, there is another communication paradigm in sensor networks, which cannot be secured through shared keys. This is the dissemination of bulk data, e.g., code updates. In the next chapter we propose a new security scheme with asymmetric cryptography properties, which make it appropriate for authenticating sequence of packets.

Chapter 4

Secure Network Programming

4.1 Introduction

Traditional methods of programming sensor nodes with a new binary require physical access to the nodes themselves: the application is developed in a PC and the resulting program image is loaded to the node through the parallel or the serial port. The same process has to be repeated for all the nodes, which have to re-deployed back to the field. Not only this solution does not scale to large number of geographically distributed nodes, but in many cases it might not even be possible to access the nodes. For these reasons, *network programming protocols* have emerged that disseminate the new code remotely, over the wireless link to the entire network, in a multi-hop fashion. Each sensor node propagates any received updates to its immediate neighbors, which in turn do the same until the new binary reaches all the nodes. In-system programmability allow nodes to reprogram themselves and start operating with the updated code.

A number of network programming protocols suitable for sensor networks have been proposed. These protocols include Deluge [Hui04], MOAP [Sta03], MNP [Kul05], INFUSE [Aru04] and Sprinkler [Nai07]. The protocols focus mainly on the reliability and the latency of the dissemination, but from a security point of view all of them assume that the nodes will behave in a legitimate way, meaning that no node can be compromised and controlled by an adversary. As a result nodes do not care about authenticating the source of the program. This fact allows an attacker to approach the deployment site and disseminate malicious or corrupted code in the network, reprogramming all the nodes at will. Thus, in the same way that a network programming protocol can help the network administrator with the program update procedure, it can also provide an easy way for an attacker to compromise the whole network by installing malicious code.

In this chapter we present Scatter, a protocol that secures network programming and allows sensor nodes to efficiently verify that the new code originates

from a trusted source, namely the base station. With this security feature added, an attacker could not authenticate herself to the network, and therefore the nodes will reject malicious updates. Therefore, having secured the communication of data from the nodes toward the base station in Chapter 3, in this chapter we deal with the reverse problem: the communication of data from the base station toward the network. Even though solutions for authenticating single broadcast messages exist (see for example the work from Benenson et al. [Ben06]), authenticating *sequence* of messages poses different requirements and calls for different kind of solutions.

We start by defining network programming in Section 4.2 and give some details about Deluge, a widely used protocol for this operation. In Section 4.3, we present the requirements for securing network programming protocols, and in Section 4.4 we elaborate on the main approaches that exist and which approach Scatter follows to be more efficient. In Section 4.5, we present a r-time signature scheme appropriate for sensor networks, which is used by Scatter to authenticate program images. Section 4.6 describes the implementation and experimental evaluation of Scatter in different sensor platforms. Finally, Section 4.7 concludes this chapter.

4.2 Network Programming

It is important to give some details about network programming protocols in sensor networks that will help us understand better the process of authenticating this procedure. In general, network programming is achieved by the following steps (see Figure 4.1):

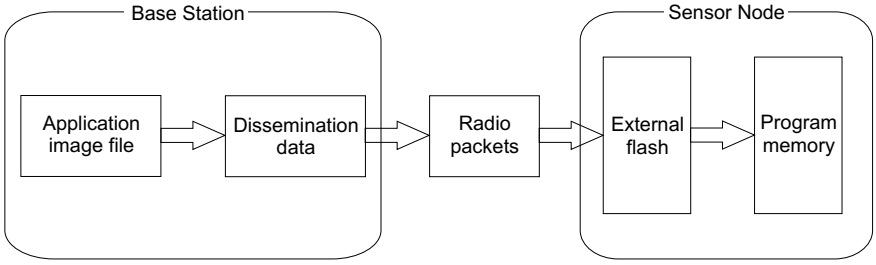

Figure 4.1: *Network programming process.*

1. The base station reads the new application binary code and breaks it into packets to disseminate.

2. The base station sends the packets to the sensor nodes within communication range.

3. The nodes store the packets in the external flash memory after receiving them. They request retransmission of any missing packets.

4. The nodes forward the packets to any of their neighbors that have not received them, until all nodes get the new code.

5. After all packets have been received, the code lies in the external flash memory of the nodes. The nodes verify the program image and call the boot loader to transfer the program code to the program memory. Then the boot loader restarts the system and the new program begins execution.

Since Deluge is one of the most commonly used protocols for network programming in TinyOS, we focus on it in this chapter. However the rest of the protocols use similar principles, so the solutions described here are also applicable to them.

Deluge propagates a program image by dividing it first into fixed-size pages and then using a demand-response protocol to disseminate them in the network. As soon as a node receives a page, it makes it available to any of its neighbors that also need it. At the same time it sends a request to the sender in order to receive subsequent pages. In this way, Deluge supports a sort of *pipelining*: already received pages are forwarded further to the rest of the nodes while the program image is not yet complete and new pages keep coming in.

The integrity of each page is verified by using a 16-bit cyclic redundancy check (CRC) across both packets and pages. So, if a packet gets dropped or corrupted, the node requests from the sender to send it again until the page is completely and correctly received. This means that any authentication protocol added on top of Deluge does not need to be robust on packet loss. It can safely assume that packets always reach their destination. Also, note that since Deluge does not start receiving the next page if the previous one is not completed, an authenticated broadcast protocol does not need to deal with out-of-order delivery of pages (even though packets may do arrive our-of-order).

4.3 Problem Definition

We now define more formally the problem that we study in this paper; we want to provide an efficient source authentication mechanism for broadcasting a program image from the base station to the sensor network. While the authentication mechanism should still allow efficient dissemination procedures, such as pipelining, it should also block malicious updates as early as possible.

The most natural solution for authenticated broadcasts is asymmetric cryptography, where messages are signed with a key known only to the sender. Everybody can verify the authenticity of the messages by using the corresponding public key, but no one can produce legitimate signed messages without the secret key. However, public key schemes like RSA, have been long thought to be impractical for the limited computational, memory and energy resources of sensor nodes. Gura et al. have shown that Elliptic Curve Cryptography (ECC) can provide substantial performance gains over RSA for constrained hardware [Gur04]. For example, TinyECC [Liu08a] provides a freely available

ECC implementation for TinyOS, which runs 12 to 16 seconds to verify a signature on MicaZ motes.

On the other hand, block cipher based hash functions run about hundreds or thousands of times faster than public key schemes. Therefore, our goal is to provide an efficient authentication scheme for a finite stream of data based on *symmetric* cryptography primitives while at the same time having the properties of asymmetric cryptography. In this way, we can avoid the use of ECC or RSA, and reach a much faster and energy efficient solution for authenticating the program image on the motes.

The solution should satisfy the following requirements:

1. **Low computational cost**. Asymmetric cryptography involves high computational cost and is not preferable for use in sensor networks. The solution should impose public-key properties, but at the same time minimize the computational cost for signature verification at the receivers (sensor nodes).

2. **Low verification time**. The rate at which a code segment is transmitted to the receiver should not be reduced significantly by the authentication protocol.

3. **Low communication overhead**. The signature transmitted with data should constitute a small percentage of the total bytes, imposing a low communication overhead.

4. **Low storage requirements**. Any cryptographic material that needs to be stored in the sensor nodes should be as small as possible, given the nodes' limited memory resources.

Moreover, since we are providing an authenticated broadcast protocol we need to assure the following security requirements:

1. **Source authentication**. A mote must be able to verify that a code update originates from a trusted source, i.e., the base station. This means that an attacker should not be able to send malicious code in the network and reprogram the nodes.

2. **Node-compromise resilience**. In case an attacker compromises a node and read its cryptographic material, she must not be able to reprogram any other non-compromised node with malicious code.

Even though we do not address protection against DoS attacks, the solution must provide some resilience against such attacks in the following sense: In case an attacker is trying to transmit malicious code to the network, any receiving node should be able to realize this as soon as possible and stop receiving it or forwarding it to other nodes. This means that nodes should not authenticate the code *after* its reception but rather *during* that process.

4.4 Design Approaches and Related Work

As we said, asymmetric cryptography is the most natural solution to authenticated broadcasts. So, one approach would be to compute a digital signature for the whole program image. However, that would require the nodes to receive and buffer the entire image in order to verify the signature, which is clearly not possible for sensor nodes, given their limited memory resources. Moreover, the receiver needs to be able to "consume" each page it receives, i.e., send it to its own neighbors (pipelining) as well as store it to the external flash memory.

If we view program images as digital streams of data, we can follow the solution by Gennaro and Rohatgi [Gen01] for signing a program image. What they proposed is to divide the stream into blocks and embed some authentication information in each block. In particular, their idea is to embed in each block a hash of the following block. This way the sender needs to sign just the first hash value and then the properties of this signature will propagate to the rest of the stream through the "chaining" technique.

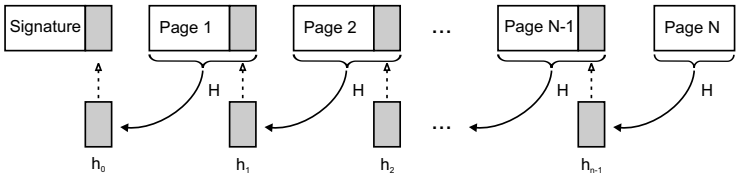

Figure 4.2: *Hash chain construction for the pages of a program image.*

In our case, given a program image divided into N fixed-size pages

$$P_1, P_2, \ldots, P_N$$

and a collision-resistant hash function H, we construct the hash chain

$$h_i = H(P_{i+1}||h_{i+1}), \qquad i = 0 \ldots N-2$$

and we attach each hash value h_i to page P_i (see Figure 4.2), where "||" denotes concatenation. For the last hash value we set

$$h_{N-1} = H(P_N).$$

Now, we can sign only the hash chain commitment, h_0, and the nodes that receive and verify that signature will be able to authenticate all the pages by just computing the same hash values and compare them with those transmitted.

4.4.1 DOS-attack Resilience

One important design choice is whether to construct the one-way hash chain over the pages of an update or the packets that compose the pages. In the former case the drawback is that the nodes need to receive the whole page before they

are able to verify its hash. Since a page is usually composed of hundreds of packets, this gives an advantage to an attacker to launch a DOS-attack, by sending a few malicious packets. After receiving the page, a node will realize that the hash verification fails and it will request the page again, as it will not know which specific packets caused this fail.

The alternative approach of computing the hash values for each packet is followed by both Dutta et al. [Dut06] and Deng et al. [Den06]. Since these protocols authenticate each packet separately, they can stop receiving them as soon as they find the first non authentic packet. Then the node can request that packet again. However, this comes at a big price.

In particular, Deng et al. [Den06] construct signed hash trees (similar to a Merkle trees) based on the hashes of each packet in the program image and they transmit these trees before the actual data. This increases the overhead of packets sent and received by the motes by about 28%. Moreover, due to memory constrains in the motes, these values need to be stored and loaded from the EEPROM each time a packet arrives, which is a very energy consuming operation for the motes [Sta03].

Similarly, Dutta et al. [Dut06], compute the hash of each packet and place it in the previous packet. This increases the overall data that have to be transmitted besides the signature, compared to constructing and transmitting a hash value for each page. It also increases he computations a mote has to perform in order to verify all these hash values, besides the RSA signature verification that this protocol uses.

So, applying a page-level hashing offers better performance but less resilience under a DOS attack, while applying a packet-level hashing imposes high overhead in all cases, in order to offer a better resilience under this attack. In our protocol we have chosen the former and simpler approach of constructing the hash chain on the pages of the program image. This is also followed by other protocols, like Sluice [Lan06]. In this work we concentrate on the signature construction and verification, rather on the hash chain construction. Let us note that applying a different hash granularity is possible in Scatter, without affecting the signature scheme proposed.

4.4.2 Signature Scheme

The most energy consuming operation in the process of secure code dissemination is the authentication of h_0, which is signed and released before the transmission of the pages. Therefore, choosing which security scheme to apply for this operation is important in the overall protocol's performance. All currently proposed solutions [Lan06; Dut06; Den06; Liu08b; Sha07] use a public key scheme like RSA or Elliptic Curve Cryptography (ECC) for signing the hash commitment. In the next section we are going to show that there is a much more efficient way that can be applied for the case of network programming protocols.

We will base our analysis on the fact that real world software updates in sensor networks do not constitute an everyday operation but rather they are performed occasionally. Therefore, we do not need to authenticate an unlimited

number of broadcasts. We only need to be able to do so for a sufficiently large number of times. This fact allows us to use *r-time signature schemes*, which exhibit fast verification times.

4.5 An r-time Signature Scheme

An r-time signature scheme is similar to a public-key scheme in that it can be used to sign messages that can be verified using publicly known information. These r-time signatures decrease dramatically the signing and verification time compared to public-key signatures, however, one can only sign up to r messages with a given key pair. After that, the security level drops below acceptable limits and a new key pair must be generated. But with regards to network programming, if we can efficiently sign and verify, for example, $r = 32$ new program images before we redistribute a new public key, we have a tradeoff that makes this an attractive solution.

Recently, some signature schemes were proposed that seem attractive for sensor networks. For example, Reyzin and Reyzin [Rey02] introduced HORS, an r-time signature scheme with efficient signature and verification times. This scheme was further improved by Pieprzyk et al. [Pie03]. Both of these r-times signature schemes can sign several messages with the same key with reasonable security before they can get compromised. However there are some drawbacks that prevent us from applying those schemes to sensor networks. The main one is the size of the public/secret key pair and the size of the generated signatures. For example, the HORS scheme uses a public key size of 20 KBytes for $r = 4$, which is not suitable for use in sensor networks. It also grows unacceptably high if we want to sign more messages (bigger r) and keep security at an acceptable level.

To overcome these drawbacks, we propose a novel r-time signature scheme, which is optimized for use in sensor networks. This scheme manages to drop the signature and public key sizes to values that fit in the memory of sensor nodes, and also reduce the signature verification time to that of a few hash operations.

Let F be an l-bit one-way function. First we need to produce the secret and public key pair. To do this the signer applies the following steps:

Secret Key Generate t random l-bit quantities for the secret key:

$$SK = (s_1, \ldots, s_t).$$

Public key Compute the public key as follows: Generate t hash values

$$(u_1, \ldots, u_t),$$

where

$$u_1 = F(s_1), \ldots, u_t = F(s_t).$$

Separate these values into d groups, each with t/d values. Use these values as leaves to construct d Merkle trees, as shown in Figure 4.3. The roots of the trees constitute the public key PK of our scheme.

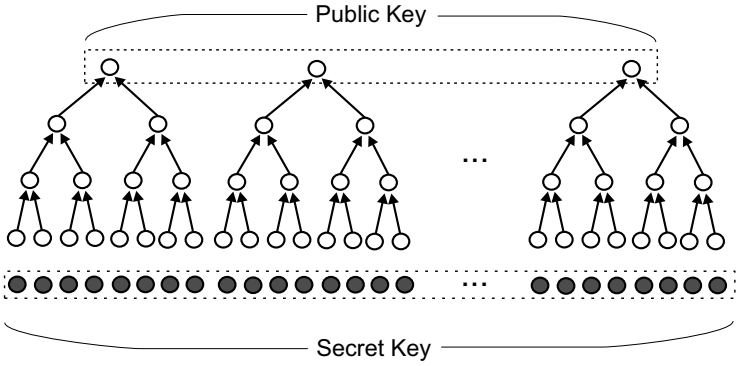

Figure 4.3: *Public key construction using Merkle trees.*

A Merkle hash tree is a complete binary tree where each node is associated with a value, such that the value of each parent node is the hash function on the values of its children:

$$v(parent) = H(v(left)||v(right))$$

where the function v here stands for the value of a node and H for a hash function.

Using the Merkle trees we have achieved a public key size of a few hash values, i.e., the roots of the Merkle trees. These values need to be passed to all sensor nodes in an authenticated way. This can be done for example during initialization of sensor nodes.

Next we show how any message m can be signed using this secret and public key pair. The procedure for this is described in the following steps (see also Figure 4.4):

1. Use a cryptographic hash function H to convert the message to a fixed length output. Split the output into k substrings of length $\log_2 t$ each.

2. Interpret each substring as integer in the range $[1 \dots t]$. Use these integers $i_1, i_2, \dots, i_k$ to select a subset σ of k values out of the set of secret values $SK = (s_1, \dots, s_t)$.

3. The signature of the message m is made up by the selected secret values along with their corresponding authentication paths.

By *authentication path* of a secret value we mean the values of all the nodes that are siblings of nodes on the path between the leaf that represents the secret value and the root of the corresponding Merkle tree, as shown in Figure 4.5. Note that for each message that we sign, a part of the secret key is leaked out. That's why this process cannot be repeated *ad infinitum*.

A node that has received the message m and wants to verify its signature, recomputes the hash value of the message, reproduces the same indices and

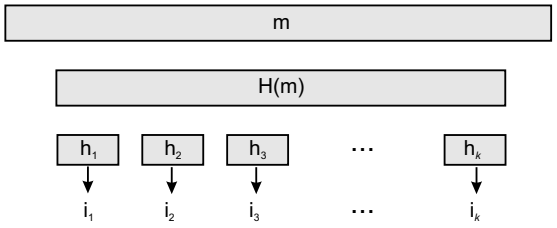

Figure 4.4: *Producing k indices of secret values from a message m.*

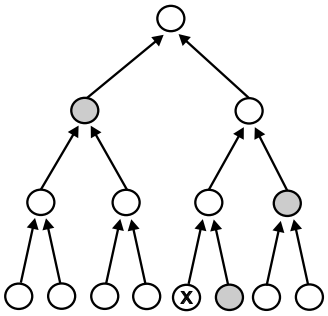

Figure 4.5: *The nodes shaded gray constitute the authentication path of the marked leaf.*

picks the corresponding values of the set PK. Remember that the node has the public values PK (roots of the Merkle trees), but not the Merkle trees. Then it evaluates each authentication path of the signature to reproduce the root of the Merkle tree and compare it with the corresponding member of the public key PK that it has in its memory. The signature is accepted if this is true for all k values. The detailed description of the algorithm is shown in Figure 4.6.

Let us emphasize that the verification of the signature at the sensor nodes requires only hash and comparison operations. Both of these operations can be performed very fast and efficiently in the nodes. For example, the time to hash one block using MD5 in a sensor node is 2.58 ms, as we will see in Section 4.6.3. The number of hash operations that will be needed depends on the number and the size of the authentication paths (or else, the size of the signature), which also determines the security level of the scheme.

The r-time signatures scheme we described can be tuned by setting various parameters, like the number of secret values t, the number of Merkle trees T, or the number of k parts that we split the hash of the message m (i.e, number of secret values that we release in the signature). The values that we choose for these parameters will determine the *signature size* and the *public key size*, two quantities that are important for the efficiency of the scheme, but also its *security level*. We study how these quantities are affected by our protocols parameters in the following sections.

Key Generation

Input: Parameters l,k,t

Generate t random l-bit strings $s_1, s_2, \ldots, s_t$.

Let $u_i = F(s_i)$ for $1 \leq i \leq t$.

Group t hash values $u_1, u_2, \ldots, u_t$ into d groups of t/d values.

Place each group at the leaves of a Merkle tree, constructing d Merkle trees.

Let $w_1, w_2, \ldots, w_d$ be the roots of the Merkle trees.

Output: $PK = (k, w_1, w_2, \ldots, w_d)$ and $SK = (k, s_1, s_2, \ldots, s_t)$

Signing

Input: Message m and secret key $SK = (k, s_1, s_2, \ldots, s_t)$

Let $h = H(m)$.

Split h into k substrings $h_1, h_2, \ldots, h_k$, of length $\log_2 t$ bits each.

Interpret each h_j as an integer i_j for $1 \leq j \leq k$.

Let $\mu_{i_j} = (s_{i_j}, AP(s_{i_j}))$, i.e., the secret value s_{i_j} along with its authentication path $AP(s_{i_j})$.

Output: $\sigma = (\mu_{i_1}, \mu_{i_2}, \ldots, \mu_{i_k})$

Verifying

Input: Message m, signature $\sigma = (\mu'_1, \ldots, \mu'_k)$ and public key $PK = (k, w_1, \ldots, w_t)$

Let $h = H(m)$.

Split into k substrings $h_1, h_2, \ldots, h_k$, of length $\log_2 t$ bits each.

Interpret each h_j as an integer i_j for $1 \leq j \leq k$.

Compute which Merkle tree corresponds to i_j: $M_j = i_j/(t/d)$ for $1 \leq j \leq k$.

Hash the values in each μ'_k to produce the root w'_{M_j}.

Output: "accept" if for each $j, 1 \leq j \leq k, w'_{M_j} = w_{M_j}$; otherwise "reject"

Figure 4.6: *The r-time signature scheme. F is a one-way function and H is a hash function.*

4.5.1 Public Key and Signature Size

There is a trade-off between the public key size and the signature size. The public key stored in each sensor node is given by the hash values residing at the roots of the trees. The more the number of the trees, the bigger the public key becomes but the smaller the signature size becomes. To see why, notice that

the signature size depends on the length of the authentication paths, which are ultimately related to the height of the Merkle trees. More trees means less secret values per tree and hence smaller height.

Ideally, we would like both of these sizes to be as small as possible. The signature is transmitted over the radio to be received and verified by the motes, so the smaller it is the less energy and time will be needed for these operations. Similarly, the public key is stored in the memory (RAM) of the motes, to be used for the signature verification procedure. So, there is a limit on how large it can be. To calculate the formulas that give these two quantities, let T denote the number of trees. Hence the public key size is simply

$$|PK| = |h|T, \tag{4.1}$$

since every root contains a hash value of its children. We use the notation $|h|$ for the output of the hash function h in bits. For example, $|h|$ can be equal to 128 bits in the case of MD5 or 160 bits in the case of SHA–1.

If we have T Merkle trees and t secret values, there can be at most t/T values stored at the leaves of each tree. Thus the height of each tree (and the length of each authentication path) is simply $\log_2(t/T)$. The signature S consists of k such authentication paths, where each path is a sequence of hash values of $|h|$ bits. Thus the signature size is given by

$$|S| = |h|(k \log_2 \frac{t}{T}). \tag{4.2}$$

This equation can be simplified further if we recall how the k secret values are selected. The message m to be authenticated is first hashed to obtain $H(m)$, a value that is $|h|$ bits long. Then these $|h|$ bits are broken into k parts, where each part references one of the secret values. Thus the number of secret values t must be equal to $2^{|h|/k}$, or equivalently

$$|h| = k \log_2 t. \tag{4.3}$$

Combining Equations (4.3) and (4.2), we find that the signature size is given by

$$|S| = |h|(|h| - k \log_2 T). \tag{4.4}$$

4.5.2 Security Level

Next we calculate the security level of the scheme, since it is also affected by the values we choose for the parameters of the scheme. Let r be equal to the number of messages that we allow to be signed with the current instance of the secret key. For an analysis (see also the work by Leonid and Natan Reyzin [Rey02]) we assume that the hash function H behaves like a random oracle and that an adversary has obtained the signatures of r messages using the same setting of secret/public key. Then the probability that an adversary can forge a message is simply the probability that after rk values of the secret key have been released, k elements are chosen at random that form a subset of

the rk values. The probability of this happening is $(rk/t)^k$. If we denote by Σ the attainable security level in bits, by equating the previous probability to $2^{-\Sigma}$, we see that Σ is given by

$$\Sigma = k(\log_2 t - \log_2 k - \log_2 r) \tag{4.5}$$

and by using Eq. (4.3), we get

$$\Sigma = k(|h|/k - log_2 k - log_2 r). \tag{4.6}$$

4.5.3 Signature Verification Time

To be able to decide on particular values for r, k and T, we would also like to have an estimation of how these parameter affect the verification time of the signature. We implemented the r-time signature scheme in TinyOS and measured the verification times for various values of T on the Mica2 platform. Figure 4.7 shows the results.

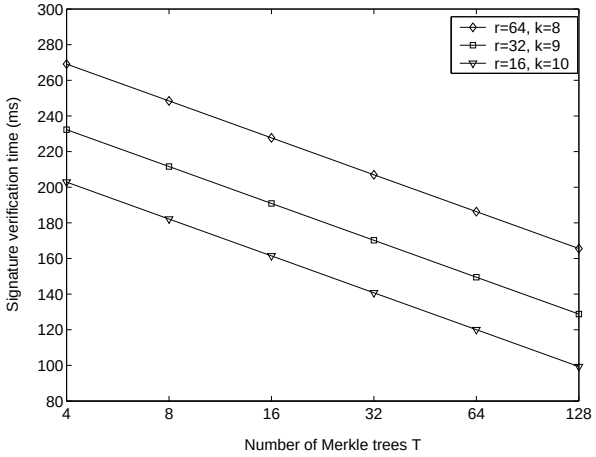

Figure 4.7: *Signature verification time as a function of T.*

For the signature verification in Figure 4.7, the time needed is directly dependent on the signature size. The parameter determining the signature size is the number of Merkle trees T. As we build more trees on the secret values, their height gets smaller, and therefore the signature size is reduced. Consequently, the verification time at the mote's side is also reduced.

Figure 4.8 is a graphical representation of Eq. (4.1), using MD5 to produce the hash values, which means $|h| = 128$. Let us keep the public key size equal to approximately 1 KByte, so that it fits well in the memory of a typical Mica2 node (approximately 4 KB of RAM). So, we set $T = 64$. Figure 4.7 shows the verification time of the signature as a function of the number of Merkle trees T, for different values of r (number of images that can be signed using the same keys) and k (number of substrings that we split the hash value of the message

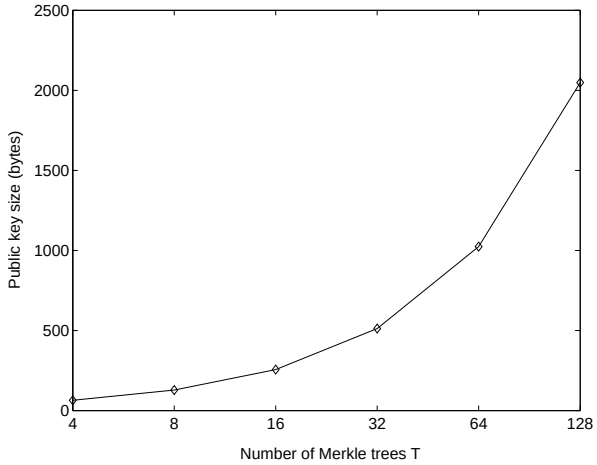

Figure 4.8: *Public key size as a function of T.*

into). Let us say we want to sign $r = 64$ messages before we need to update the keys. What would be a good value for k to choose? Figures 4.9 and 4.10, show a graphical representation of Equations (4.4) and (4.6) respectively, for $T = 64$ and $|h| = 128$.

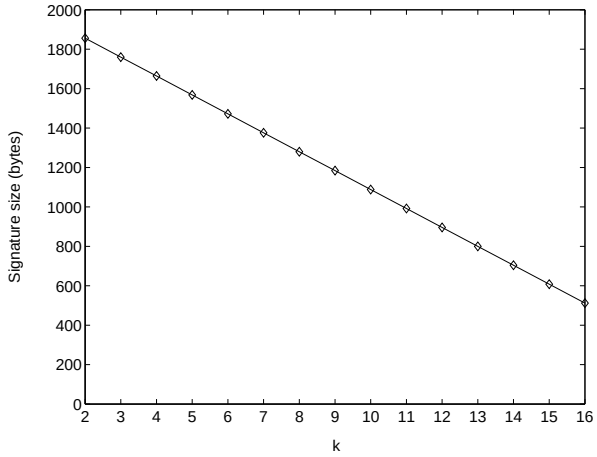

Figure 4.9: *Signature size as a function of k.*

Observing Figures 4.9 and 4.10, a good value for k could be $k = 8$. Then, for $r = 64$ we would get a security level of around 60 bits against passive adversaries, and a signature size of 1280 bits. The verification time of this signature, from Figure 4.7 would be equal to 186.3 ms. Notice that we used standard implementations of hash functions, so we believe that this signature verification time can be improved even further using optimized code for the

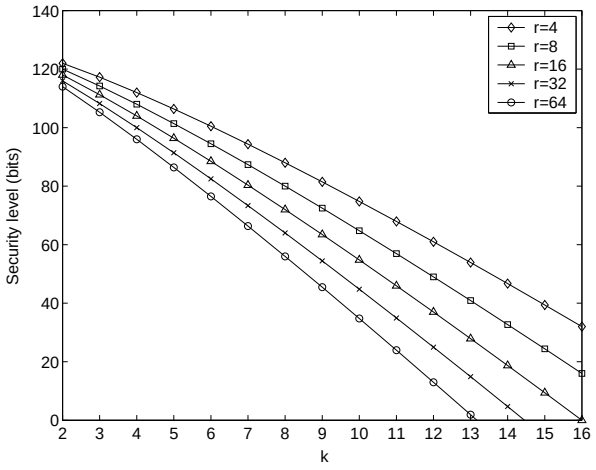

Figure 4.10: *Security level as a function of k.*

particular hardware of the motes.

We can see now the advantage of using this r-time signature scheme to authenticate a message in sensor networks. If we had chosen to use Elliptic Curve Cryptography, the signature verification would be much larger. For example, TinyECC [Liu08a] provides ECDSA [Ame98] verification functionality on the sensor nodes that takes between 12 and 16 seconds, which is highly inefficient compared to the verification times of the order of milliseconds, given in Figure 4.7.

To get a complete picture of the performance of a programming protocol like Deluge that uses this r-time signature scheme to authenticate the broadcasted program images, we need first to show how the integration of the two schemes is done, and then measure the overall time to download and authenticate a complete image to a sensor node. We do this in the next sections, presenting also some implementation details.

4.6 Implementation and Experiments

4.6.1 Implementation

We implement Scatter as an extension to Deluge in TinyOS distribution. Our implementation has two parts, one for the base station side and one for the sensor side programs. The former extends the Deluge Java tools to construct and inject new code dissemination packets into the sensor network. The latter is written in nesC [Gay03] and runs on regular sensor nodes.

We add two main functionalities in the Java tools on the base station side: Computation of the hash values of the image pages, and signing of h_0, the public commitment of the hash chain we have built in order to authenticate

the sequence of pages. In the previous section we described how we can create a signature for a message m using an r-time signature scheme. Now, we will apply this scheme to sign h_0, the public commitment of the hash chain we have built in order to authenticate the sequence of pages.

Figure 4.11 shows in detail the process of preparing a secure program image at the PC side for dissemination in the network. Deluge first partitions the image into P pages, given as a parameter the page size $|Page|$. Each page is further partitioned into N packets, which are transmitted to the sensors. For completeness of the figure, we show what constitutes a program image that Deluge transmits to the nodes: some metadata associated with it, information about the length of the image in bytes, and the image itself. To the pages that the image is partitioned to (1081 bytes in this example), Deluge also appends a CRC that is calculated for each page.

According to the authentication scheme, after we partition the image into pages, we compute a hash chain in reverse order from the last page to the first. We need to append these hash values at the end of the corresponding pages. If we use MD5, a hash value is 16 bytes, so it fits in a packet of 23 bytes (the default value for TinyOS packets). So, each value of the hash chain padded with $0s$ (to give 23 bytes) is attached at the end of the corresponding page.

The final step includes the addition of one or more pages at the beginning of the image that stores the first value of the hash chain along with the signature we produced for it using the r-time signature scheme. The number of the extra pages is determined by the size of the signature (normally it should not be more than 2 pages). If $|S|$ denotes the size of the signature, then the number of the extra pages is $\lceil |S|/|Page| \rceil$. If necessary, we add some padding $0s$ at the end of the signature to fill up the page.

Figure 4.11 illustrates an example program image, where pages 0 and 1 are reserved for the signature. The original image is stored in pages $2 \ldots P + 2$. Besides these two pages, the only extra information transmitted with the program image is the last packet of each page, containing the corresponding hash chain values.

We now move to the side of the motes and describe the verification process for the signature and the authentication of each page. In particular, we are interested in the overhead posed by this security protocol in terms of memory and time.

4.6.2 Memory Requirements

The dissemination and authentication of the code is done in a per-page basis. As soon as the last packet of a page is received, the mote checks to see if it is complete and issues a request back to the sender for any missing packets, like in original Deluge. When the page is complete, a CRC check is done to verify its integrity. Then the mote needs to verify that the hash value of the page it just received is the same as the corresponding value that came with the previous page.

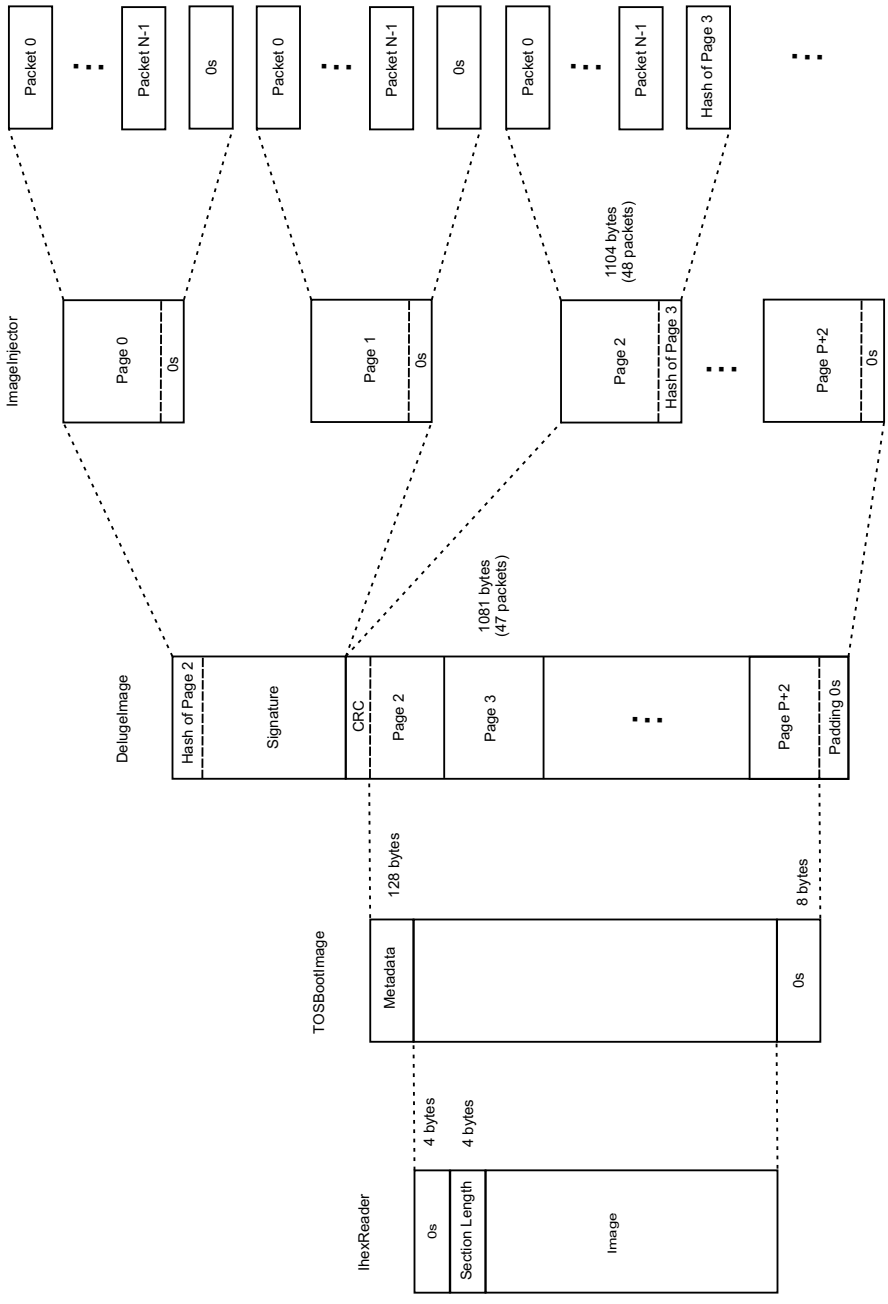

Figure 4.11: *Partitioning of the program image into pages and packets before the dissemination into the network.*

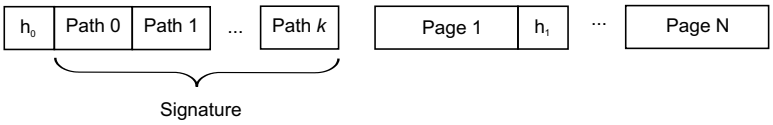

Figure 4.12: *The order at which a mote receives the signature, the pages and their hash values. Verification of the signature is possible by storing only one path at the time.*

To be able to hash the page, the mote needs to buffer it in RAM. This requires a buffer equal to the page size. The default value of Deluge for that parameter is 1104 bytes, so it perfectly fits in a sensor node's memory (being 4 KB for Mica2, or 10 KB for Tmote). Of course, the page size is a parameter of Deluge that can easily be changed to any smaller value if the final memory requirements exceed the available resources.

An exception for what is described above is the first two pages that the mote receives, which contain the signature of the hash chain commitment. They also need to be stored in EEPROM as the rest of the pages. However, the mote does not need to keep the whole page in RAM, but rather just each authentication path. This is because the signature is made up by authentication paths and the authentication of each path can be done independently by the others.

Referring to Figure 4.12, the mote first receives the hash value of Page 1. This will provide the indices to the public values. Then, the first authentication path of the signature will be received. The verification of that path evolves only a few hashing operations and a comparison of the result with the corresponding public value. This can be done fast enough by the mote, so that the path has been verified before the next path starts coming in. So, only a temporary storing is needed, equal to the size of a path (dependent on the height of the Merkle trees at the base station).

4.6.3 Experimental Evaluation

In this subsection, we report the experimental evaluation of Scatter in two different hardware platforms, namely MicaZ and Mica2. For comparison purposes, we perform the same set of experiments with Deluge, as well as Seluge. We choose to compare Scatter with Seluge, because according to their experiments [Liu08b], their protocol outperforms the approach of Deng et al. [Den06] and Dutta et al. [Dut06].

The main overhead in execution time that Scatter imposes on Deluge is due to the main two security operations evolved, i.e., hashing of the pages and verifying the signature. We measured that overhead for each operation separately, as well as the total overhead compared to plain Deluge over a complete image transfer, using our implementation in Mica2 motes.

We first examined the execution time needed to hash a page of default size (1104 bytes). This is shown in Table 4.1 for two different hash functions, SHA-1 and MD5. It is obvious from the results that the time performance of MD5 is

Table 4.1: *Time performance of hash functions in Mica2 platform.*

Function	Time to hash one block	Time to hash 1104 bytes
SHA–1	7.56 ms	131.7 ms
MD5	2.58 ms	49.6 ms

considerably better than that of SHA-1. For both functions we used publicly available code and no optimization was made for the specific hardware. So, these values can be further improved.

We have already shown the signature verification times for Mica2 in Figure 4.7. So, now we show the overall time that it takes to download and authenticate a new program image on one mote from the PC and compared it with the time that it takes for plain Deluge. To investigate and compare the impact of dissemination code size on performance, we use six different code image sizes, ranging from 5 to 30 pages of 1104 bytes each. These correspond to images from 4.8 KB to 31 KB. Figure 4.13 shows our results. Note that the time for a mote to receive a new image is subject to packet losses so we perform the same experiment 20 times and take an average over them. For this specific experiment we used $l = 80$, $k = 8$, $t = 65536$, $r = 32$, and $T = 32$. This setting gave a signature size of 1408 bytes and a public key size equal to 512 bytes.

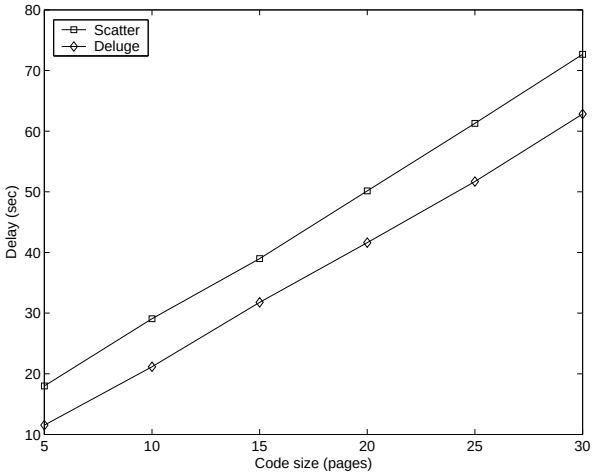

Figure 4.13: *Time taken to transfer an image to a Mica2 node.*

An important observation that we can make from Figure 4.13 is that the overhead imposed on Deluge by Scatter is almost steady as the code size increases. This is because for applications that are larger by p pages, the time overhead will increase by p times the time for the sensor node to hash a page and

compare it with the hash value included in the next page. This is insignificant compared to the overhead due to signature transfer and verification, which is independent on the program image size.

To compare Scatter with Seluge we run another set of experiments using MicaZ motes. This is because Seluge's implementation is based on CC2420 radio component on MicaZ to reduce its overhead. It uses the hardware cryptographic support available by that component for symmetric cryptographic operations and it also uses the larger packet payload sizes supported by IEEE 802.15.4, the standard implemented by CC2420. Seluge needs large packets to accommodate its hash values, but in this way it also decreases the total number of packets required for a given program image and therefore decreases the propagation delay.

Figure 4.14 shows the total delay to download different program images from the PC to a MicaZ mote, using Deluge, Scatter and Seluge. For Seluge we used two variants, one with packet size equal to 62 bytes and one with packet of 102 bytes. For Deluge and Scatter we used the default TinyOS packet size, which is 23. By using larger packet sizes for them also, it would only decrease the transfer time further.

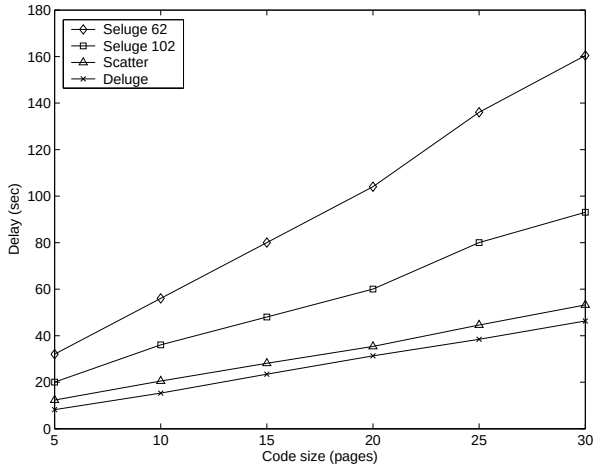

Figure 4.14: *Time taken to transfer an image to a MicaZ node.*

Observing Figure 4.14, we can see the benefits from using an authentication scheme that is based only to hash operations, like Scatter, as opposed to a scheme that uses ECC based public key operations, like Seluge. Seluge uses TinyECC [Liu08a] on the motes, which provides an ECC implementation for TinyOS that includes an ECDSA (Elliptic Curve Digital Signature Algorithm) module. As a result, Seluge-62 introduces an average overhead of 255% compared to the completion time of Deluge and Seluge-102 brings it down to 114%. On the other hand, for Scatter we measure an average overhead of 24% compared to Deluge.

Let us also note that the lower transmission times observed in Figure 4.14 regarding Deluge and Scatter, compared to the corresponding measurements obtained for Mica2 in Figure 4.13, are due to the faster data rate of the CC2420 radio available in MicaZ. Mica2 uses the ChipCon CC1000 RF transceiver running at 38.9 kbps, while MicaZ features a much faster 250 kbps radio.

4.6.4 Updating the Public Key

After using the key pair we have produced for authenticating new program images r times (r being for example 32 or 64), we can no longer use them, because the security level has dropped below acceptable levels. So we need to update them, and we use the same authentication scheme to distribute the new public key to the motes. To send this new public key to the motes we sign it using the current secret key, and send it to the motes just like if it was a new image (only much smaller). We embed a small bit pattern at the beginning to allow the motes realize that it is the new public key. In this way the motes will verify the new public key and start using it for the next images.

For $l = 80, k = 8, t = 65536, r = 32$, and $T = 32$ we calculated the time needed to send a new public key in one mote in an authenticated way. The new public key (512 bytes) fits in one page, resulting in 3 pages to be transmitted in total (including 2 pages for the signature). Performing this experiment on Mica2, the mote was able to receive and verify the new public key in 7.01 seconds, which is a low price to pay, given that we need to perform this operation after $r = 32$ code disseminations.

4.7 Conclusions

In this chapter we presented Scatter, an efficient and practical scheme for authenticated network programming in sensor networks. The solution imposes asymmetric cryptography properties using *symmetric* cryptography primitives, outperforming other proposals. It minimizes the public key and signature sizes to values that are appropriate for sensor networks. The verification procedure at the motes is also time and computationally efficient, since it involves only hashing and comparison operations.

We showed how easily an implementation of such a scheme can be adapted to an existing in-network programming protocol, namely Deluge. We tested our secure Deluge version and measured the verification time of the signature at the mote's side. This verification time constitutes the main computational overhead imposed by our scheme and is at the order of one to two hundreds milliseconds, which is very efficient for applications running on sensor nodes.

Chapter 5

The Intrusion Detection Problem

5.1 Introduction

The techniques introduced by the previous chapters can effectively protect a sensor networks against certain attacks. However, these security protocols are based on a particular attacker assumption α. If the attacker is "weaker" than α, the protocol will achieve its security goal, i.e., it will *prevent* an intruder from breaking into a sensor network and hinder its proper operation. If the attacker is "stronger" (i.e., behaves more maliciously) than specified by α, there is a non-negligible probability that the adversary will break in. Because of their resource constraints, sensor nodes usually cannot deal with very strong adversaries. So what is needed is a second line of defense: An Intrusion Detection System (IDS) that can *detect* a third party's attempts of exploiting the insecurities of the network, even if such attacks have not been experienced before.

Intrusion detection systems provide a necessary layer of in-depth protection for wired networks. However, little research has been performed about intrusion detection in the areas of wireless sensor networks. The reason that this kind of research has not advanced yet may be that the concept of "intrusion detection" is not clear in the context of such networks. In this chapter therefore we concentrate on the definition of the problem and identify scenarios in which it can be solved, assuming one attacking node. In our opinion it is essential to set the theoretical foundation of this new research area first, before trying to design and implement an Intrusion Detection System (IDS) specifically for sensor networks.

In Section 5.2, we briefly survey the existing intrusion detection techniques from wired and ad-hoc networks and indicate important approaches that are appropriate for wireless sensor networks. In Section 5.3, we take a more detailed look into one of these approaches, namely the watchdog approach. In Section 5.4, we outline the requirements that an IDS for sensor networks should satisfy and in Section 5.5, we review the existing bibliography. Then, in Sec-

tion 5.6, we define our system model and assumptions, based on which we formulate the problem of intrusion detection, in Section 5.7. In Section 5.8 we prove necessary and sufficient conditions under which an IDS can successfully detect an attacker and we describe scenarios in which cooperative intrusion detection is unsolvable. This sets the necessary theoretical foundation for Chapter 6, where we propose an IDS specifically designed for sensor networks. Finally, Section 5.9 concludes this chapter.

5.2 Designing an IDS for Sensor Networks

In intrusion detection, we wish to provide an automated mechanism that identifies the source of an attack and generates an alarm to notify the network or the administrator, so that appropriate preventive actions can take place. As an attack we consider any set of actions that target the computing or networking resources of our system. Attackers may be using an external system without authorization or have legitimate access to our system but are abusing their privileges (i.e., an insider attack). It is important to realize here that the IDS comes into the picture *after* an intrusion attempt has occurred. It does not try to prevent these attempts in the first place.

5.2.1 Intrusion Detection Techniques

In order to detect an intruder, we need to use a model of intrusion detection. We need to know what an IDS should look out for. In particular, an IDS must be able to distinguish between normal and abnormal activities in order to discover malicious attempts in time. However this can be difficult, since many behavior patterns can be unpredictable and unclear. There are three main techniques that an intrusion detection system can use to classify actions [Axe00]:

- *Misuse detection.* In misuse detection or signature-based detection systems [Ilg95; Lin99], the observed behavior is compared with known attack patterns (signatures). So, action patterns that may pose a security threat must be defined and given to the system. The misuse detection system tries to recognize any "bad" behavior according to these patterns. Any action that is not clearly prohibited is allowed. The main disadvantage of such systems is that they cannot detect novel attacks. Someone must continuously update the attack signature database. Another difficulty is that signatures must be written in a way to encompass all possible variations of the pertinent attack, and yet avoid flagging non-intrusive activity as an intrusive one.

- *Anomaly detection.* Anomaly detection [Jav94] overcomes the limitations of misuse detection by focusing on normal behaviors, rather than attack behaviors. This technique first describes what constitutes a "normal" behavior (usually established by automated training) and then flags as intrusion attempts any activities varying from this behavior by a statistically

significant amount. In this way there is a considerable possibility to detect novel attacks as intrusions. There are two problems associated with this approach: First, a system can exhibit legitimate but previously unseen behavior. This would lead to a substantial false alarm rate, where anomalous activities that are not intrusive are flagged as intrusive. Second, and even worse, an intrusion that does not exhibit anomalous behavior may not be detected, resulting in false negatives.

- *Specification-based detection*. Specification-based detection [Ko01; Ko97] tries to combine the strengths of misuse and anomaly detection. It is based on deviations from normal behavior. However, in this case, the normal behavior is not defined by machine learning techniques and training. It is based on manually defined specifications that describe what is a correct operation and monitors any behavior with respect to these constraints. In this way, legitimate but previously unseen behaviors will not cause a high false alarm rate, as in the anomaly detection approach. Also, since it is based on deviations from legitimate behaviors, it can still detect previously unknown attacks. On the other side, the development of detailed specifications by humans can be time-consuming and bare the inherent risk that certain attacks may pass undetected.

Caution must be taken when applying the anomaly detection technique in sensor networks. It is not easy to define what is a "normal behavior" in such networks, as they usually adapt to variations in their environment or according to other parameters, such as the remaining battery level. So, these legitimate changes of behavior may easily be mistaken from the IDS as intrusion attempts. Moreover, sensor networks cannot bear the overhead of automatic training, due to their low energy resources. Specification-based detection seems the most appropriate approach in this case, if one can design appropriate rules that cover as broad range of attacks as possible.

5.2.2 Intrusion Detection Architectures

Traditionally, intrusion detection systems for fixed networks were divided into two categories: *host-based* and *network-based*. The host-based architecture was the first architecture to be explored in intrusion detection. A host-based intrusion detection system (HIDS) is designed to monitor, detect, and respond to system activity and attacks on a given host (node). Any decision made is based on information collected at that host by reviewing audit logs for suspicious activity. This contradicts the distributed nature of sensor networks and makes it impossible to detect network attacks. A network-based architecture is clearly more appropriate here.

Network-based intrusion detection systems (NIDS) use raw network packets as the data source. A network-based IDS typically listens on the network, and captures and examines individual packets in real time. It can analyze the entire packet, not just the header. In wired networks, active scanning of packets from

a network-based intrusion detection system is usually done at specific traffic concentration points, such as switches, routers or gateways. On the other hand, wireless sensor networks do not have such "bottlenecks". Any node can act as a router and traffic is usually distributed for load balancing purposes. So, it is impossible to monitor the traffic at certain points.

So, when designing an IDS for sensor networks, we must be careful of where to locate the detection agents, due to the distributed nature of the network and traffic routed within. One possible solution is to have an identical agent inside every node. That would be a realistic solution, if the agents were designed to be lightweight and cooperative through a distributed algorithm. Another solution would be to have a hierarchical model, where some more computationally intensive agents were placed on certain nodes, while other agents with restrictive tasks were placed on the rest of the nodes. We review systems using both solutions in Section 5.5.

5.2.3 Decision Making Techniques

Intrusion detection systems can be further classified according to the decision making techniques that they use in order to detect and initiate a response to an intrusion attempt. This decision can be made either collaboratively or independently by the nodes.

Since the nature of sensor networks is distributed and most of the services provided require cooperation of other nodes, it is only natural that intrusion detection should also be done in a *cooperative* manner. In this case, every node participates in intrusion detection and response by having an IDS client installed on them. Each node is responsible for detecting attempts of intrusion locally. If an anomaly is detected by a node with weak evidence, or if the evidence is inconclusive, then a cooperative mechanism is initiated with the neighboring nodes in order to take a global intrusion detection action. Such a mechanism is described by Zhang et al. [Zha03] for ad-hoc networks, where nodes use a majority-based distributed intrusion detection procedure. More sophisticated cooperative decision-making schemes may use mobile agents [Alb02; Kac03] or fuzzy logic [Sir01] to better support the decision process.

When designing a cooperative decision making mechanism for intrusion detection in sensor networks, one should consider the fact that a node can be compromised and hence, send falsified data to its neighbors trying to affect the decision. So, one must be skeptical as to which nodes to trust. The fact that it is difficult for an adversary to compromise the majority of the nodes in a specific neighborhood can play an important role here. Moreover, a cooperative mechanism has to consider the bandwidth and energy resources of the nodes. The nodes cannot exchange security data and intrusion alerts without considering the energy that has to be spent for sending, receiving and processing these messages.

In an *independent* decision-making system, there are certain nodes that have the task to perform the decision-making functionality. They collect intrusion and anomalous activity evidences from other nodes and based on them they

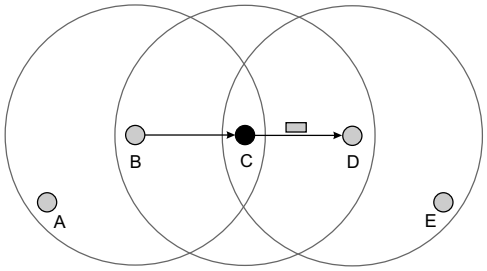

Figure 5.1: *Node B is selectively forwarding packets to node C. Node A promiscuously listens to node B's transmissions.*

can make decision about network-level intrusions. The rest of the nodes do not participate in this decision. In such architectures, the decision-making nodes can attract the interest of an attacker, since their elimination would leave the network undefended. Furthermore, the information that they process is limited, since it originates from specific nodes. Another disadvantage of such approaches is that they restrict computation-intensive analysis of overall network security state to a few key nodes. Their special mission of processing the information from other nodes and deciding on intrusion attempts results in an extra processing overhead, which may quickly lead to their energy exhaustion, unless different nodes are dynamically elected periodically.

5.3 The Watchdog Approach

As we saw in Section 5.2.2, in order to apply a network-based intrusion detection system in sensor networks, packet monitoring should take place in several nodes of the network, due to its distributed nature. In this section we look in more detail at a technique that can be used for packet monitoring, called the *watchdog* approach [Mar00].

The watchdog approach relies on the broadcast nature of the wireless communications and the fact that sensors are usually densely deployed. Each packet transmitted in the network is not only received by the sender and the receiver, but also from a set of neighboring nodes within the sender's radio range. Normally these nodes would discard the packet, since they are not the intended receivers, but for intrusion detection this can be used as a valuable audit source. Hence, a node can activate its IDS agent and monitor the packets sent by its neighbors, by overhearing them. However, this is not always adequate to draw safe conclusions on the behavior of the monitored node.

There are certain concerns that arise in this case which will be highlighted by way of an example. In the setting shown in Figure 5.1, suppose that a packet should follow the path $A \rightarrow B \rightarrow C \rightarrow D$. Now, suppose that C is compromised and exhibits a malicious behavior, selectively dropping packets. There are three cases, arising from the wireless nature of communications, where having a node

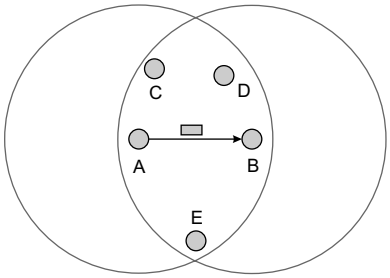

Figure 5.2: *Nodes A, C, D and E can be watchdogs of the link $A \rightarrow B$.*

B monitoring node C cannot result in a successful detection of node C:

1. Node C forwards its packet and node A sends a packet to B at the same time. Then a collision occurs at B. Node B cannot be certain which packets caused this collision, so it cannot conclude on C's behavior.

2. Node C forwards its packet to node D at the same time that node E makes a transmission. Then a collision occurs at D, which cannot be detected by B. Node B thinks that C has successfully forwarded its packet and therefore, C can skip retransmitting the packet, without being detected.

3. Node C forwards its packet to node D at the same time that D makes a transmission. Then a collision occurs at D. Again, node B thinks that C has successfully forwarded its packet, even though it never reached node D.

From the above cases we can conclude that only one watchdog is not always enough to detect an attack, so this approach should involve information from more nodes. Then these nodes could cooperate and exchange their partial views in order to draw their final conclusions. In Chapter 6, we will describe an IDS based on this observation.

Furthermore, to detect certain attacks, it's not enough to monitor just one node, but rather a *link*, meaning the packets transmitted by the nodes at both of its ends. For example, to detect selective forwarding, a watchdog should be able to overhear packets arriving at a node and transmitted by that node. So, if we want to see whether a node B forwards packets sent by node A, we must activate a watchdog that resides within the intersection of A's and B's radio range. For example, in Figure 5.2, the nodes A, C, D and E can be watchdogs for the communication between A and B.

Finally, one could argue that the watchdog approach increases the energy consumption of the nodes, since they have to overhear packets not destined for them. However, let us note that in most radio stacks of today's sensor platforms each node receives packets sent by neighboring nodes anyway. They cannot know if a packet is addressed to them unless they receive it and check the destination field. So, the only overhead imposed to the nodes is any further processing of the packet.

5.4 Requirements of IDS for WSN

In order to elaborate on the requirements that an IDS system for sensor networks should satisfy, one has to look at the specific characteristics of these networks. Each sensor node has limited communication and computational resources and a short radio range. Furthermore, each node is a weak unit that can be easily compromised by an adversary [Bec06], who can then load malicious software to launch an insider attack.

In this context and taking under consideration the discussion in Section 5.2, a distributed architecture, based on node *cooperation* is a desirable solution. In particular, we require that an IDS for sensor networks must satisfy the following properties:

1. *Localize auditing.* An IDS for sensor networks must work with localized and partial audit data. In such networks there are no centralized points (apart from the base station) that can collect audit data for the entire network, so this approach fits the sensor networks paradigm. Dealing with partial data means that the IDS should also address the problem of high false alarm rate.

2. *Minimize resources.* An IDS for sensor networks should utilize a small amount of resources. The wireless network does not have stable connections, and physical resources of network and devices, such as bandwidth and power, are limited. Disconnection can happen at any time. In addition, the communication between nodes for intrusion detection purposes should not take too much of the available bandwidth.

3. *Trust no single node.* In a collaborative IDS, the nodes cannot assume that other participant nodes can be trusted. Unlike wired networks, sensor nodes can be easily compromised. These nodes may behave normally with respect to the routing of the information in order to avoid being detected by the IDS. However, they can expose a malicious behavior to obstruct the successful detection of another intruder node. Therefore, in cooperative algorithms, the IDS must assume that *no* single node can be fully trusted.

4. *Be truly distributed.* The process of data collection and analysis should be performed on a number of locations, in order to distribute the load of the intrusion detection. The distributed approach also applies to execution of the detection algorithm and alert correlation.

5. *Support addition of new nodes.* In practice it is likely that a sensor network will be populated with more nodes after its deployment. An IDS should be able to support this operation and distinguish it from an attack (e.g. wormhole attack) that has the same effect.

6. *Be secure.* An IDS should be able to withstand a hostile attack against itself. Compromising a monitoring node and controlling the behavior of the embedded IDS agent should not enable an adversary to revoke a legitimate node from the network, or keep another intruder node undetected.

5.5 Existing Approaches

Several proposed architectures of intrusion detection systems already exist for Ad Hoc networks. The first scheme to be proposed was introduced by Zhang et al. [Zha03], which is a distributed and cooperative IDS model, where every node in the network participates in the detection process. Another architecture, called LIDS [Alb02] utilizes mobile agents on each of the nodes. These agents are used to collect and process data on remote hosts and transfer the results back to their home nodes, or migrate to another node for further investigation. Also based on mobile agents is the IDS proposed by Kachirski and Guha [Kac02]. The agents are categorized as monitoring, decision-making and action agents. All nodes accommodate host-based monitoring agents but only a few nodes chosen by a distributed algorithm host agents with network monitoring and decision capabilities.

These IDS architectures for Ad Hoc networks cannot be applied directly to sensor networks. The differences in the nature of the two kinds of networks impose different requirements, which forces us to design new solutions. A first attempt to apply anomaly detection in sensor networks is presented by da Silva et al. [Sil05]. According to the author's proposed algorithm, there are some monitor nodes in the network, which are responsible for monitoring their neighbors looking for intruders. These nodes listen to messages in their radio range and store certain message fields that might be useful to the rule application phase. The rules concern simple observations, such as:

- the message sending rate must be within some limits,
- the payload of a forwarded message should not be altered,
- the retransmission of a message must occur before a defined timeout, and
- the same message can only be retransmitted a limited number of times.

Then they try to detect some attacks, like message delay, repetition, data alteration, blackhole and selective forwarding. It is concluded from the paper that the buffer size to store the monitored messages is an important factor that greatly affects the false positives number. Given the restricted memory available in motes, it turns out that the detection effectiveness is kept to lower levels.

A similar approach is followed by Onat and Miri [Ona05], where each node has a fixed-size buffer to store the packets received from neighbors and their corresponding arrival time and received power. If its power is not within certain limits, the packet is characterized anomalous. An intrusion alert is raised if the rate at which anomalous packets are detected over the overall rate at which packets are received is above a given threshold. In this way the authors claim that it is possible for a node to effectively identify an intruder impersonating a legitimate neighbor.

Loo et al. [Loo05] and Bhuse and Gupta [Bhu06] describe two more IDSs, emphasizing on routing attacks in sensor networks. Both papers assume that

routing protocols for ad hoc networks can also be applied to WSNs: Loo et al. [Loo05] assume the AODV (Ad hoc On-Demand Distance Vector) protocol while Bhuse and Gupta [Bhu06] use the DSDV and DSR protocols. Then, specific characteristics of these protocols are used like "number of route requests received" to detect intruders. However, to the best of our knowledge, these routing protocols are not attractive for sensor networks and they have not been applied to any implementation that we are aware of.

Roman et al. [Rom06] propose an IDS architecture where all nodes are loaded with an IDS agent. This agent is divided into two parts: local agents and global agents. Local agents are active in every node and are responsible for monitoring and analyzing only local sources of information. Global agents are active at only a subset of nodes. They are in charge of analyzing packets flowing in their immediate neighborhood. In order for the whole communication in the network to be covered by global agents, the global agents must be activated at the right nodes. For example, if clusters are used, the global agents will be activated at the cluster-heads. In case of a flat architecture, the authors propose another solution (called spontaneous watchdogs) that tries to activate only one global agent for a packet circulating in the network.

A completely different approach is presented by Anjum et al. [Anj04], where the authors assume signature-based intrusion detection. This is the only work that takes a position against promiscuous monitoring and argues that detection should be based only on the analysis of packets that pass through a node. The problem then is to determine at which nodes should the IDS modules be placed, such that all the packets are inspected at least once. The proposed solution is based on the concepts of dominating set and minimum cut set and on the requirement that the nodes running the IDS module should be tamper resistant.

5.6 System Model

5.6.1 Sensor Nodes and Communication

We present a strong system model used for proofs of necessary and sufficient conditions for intrusion detection in the sequel. It is useful, because if some problem is impossible to solve in our model, it is also impossible to solve in weaker models which are closer to the reality.

In our model, a wireless sensor network consists of a set $S = \{s_1, s_s, \ldots, s_n\}$ of n sensor nodes. Sensors communicate by sending messages over a wireless broadcast medium, meaning that if some sensor s sends a message, many other sensors can receive the message simultaneously. We assume that communication is instantaneous and that possible collisions on the wireless medium can only delay the receipt of a message for a relatively short time.

For any sensor node s, the set of nodes with which it can directly communicate is denoted by $N(s)$. For simplicity, we assume a static and symmetric neighborhood relation, i.e., if $s \in N(s')$ then $s' \in N(s)$. We assume that every node knows its 2-hop-neighborhood.

Although the above assumptions are strong, considering the unreliable asymmetric wireless links and frequent neighborhood changes in real sensor networks, we use them especially for the proofs presented in this chapter. In Chapter 6 we discuss how the system model can be weakened, and how these changes will affect our proofs and protocols. We then present an IDS for realistic sensor networks, which does not make these strong assumptions.

We make no assumptions about the communication topology defined by all neighborhood sets apart from that all nodes that behave according to the protocol (honest nodes, see below) are connected via a path consisting only of the honest nodes. We expect that in typical sensor networks the density of the nodes is rather high so that this condition will be satisfied with high probability.

We assume a synchronous system model, i.e., sensor nodes are able to measure time reliably using e.g., hardware clocks that run within a linear envelope of real time. Time synchronization in sensor networks received close attention in the last years, efficient protocols for secure time synchronization were developed [Gan05; Sun06], such that this assumption seems reasonable.

5.6.2 Attacker Model

We assume that an attacker can capture at most t nodes to launch an attack on the sensor network. We model this by allowing at most t nodes to behave in an arbitrary manner (Byzantine failure [Gär03]). However, we do not propose a Byzantine Agreement protocol, but focus on the Intrusion Detection Problem (Section 5.7). The relationship between Intrusion Detection and Byzantine Agreement is discussed later in Section 5.8.3.

The predicate $faulty(s)$ on S is true if and only if (iff) s is captured by the attacker. We define $honest(s) \equiv \neg faulty(s)$.

The attacker can follow the protocol for a certain period of time and therefore behave in a way which cannot be detected. However, at some point in time the attacker must deviate from the protocol in some faulty node to launch an attack. At this point in time, we say that the attacker *attacks*.

In this chapter, we concentrate on the case where $t = 1$, as it is already rather complex. In this case, we call the faulty node the *source* of the attack, or the attacker, and use the predicate $source(s)$ which is true iff s is the attacker.

5.6.3 Alert Module

In intrusion detection, we wish to identify the attacker. Attacks are locally detected by a local intrusion detection mechanism. We abstract such mechanisms of a sensor node into an *alert module*.

Whenever the alert module at node s notices something wrong in its neighborhood, the alert module simply outputs some set $D(s)$ of *suspected sensors*, called the *suspected set*. The size of $D(s)$ depends on the quality of the alert module. If $|D(s)| = 1$, then the sensor has identified the attacker. Most often however, $D(s)$ will contain a larger set of neighbors or may even be equal to $N(s)$.

For example, in the detection of the selective forwarding attack [Kro07b], the nodes are able to identify the one node that drops the messages by monitoring the transmissions in their neighborhood. In this case we have that $|D(s)| = 1$. In another example, the protocol for detecting the sinkhole attack [Kro07c], nodes do not have a clue of who the attacker might be, except that it has to be one of their neighbors. Therefore, $D(s) = N(s)$ in this case.

Formally, the alert module satisfies the following properties:

- If the alert module at a honest node s outputs $D(s)$, then the source is in that set, i.e., $\exists s' \in D(s) : source(s')$.
- If the attacker attacks, then within some time delay δ the alert module at some sensor s outputs a set $D(s)$.
- Only neighbors are suspected by honest nodes, i.e., $\forall s \in S : honest(s) \implies D(s) \subseteq N(s)$.

If the alert module at some node s outputs some set, we call s an *alerted node*. The predicate A on S denotes the set of alerted nodes, i.e., $A(s)$ holds iff s is an alerted node. The set of alerted nodes is called the *alerted set*. Note that faulty nodes may or may not belong to the alerted set, depending on the strategy that the attacker chooses to follow.

Not all neighbors of the attacker necessarily belong to the alerted set. For example, for the detection of selective forwarding only the alert module of the common neighbors of nodes s and s' is triggered [Kro07b]. This is because only these nodes can hear both the transmissions of node A and node B. However, for the Sinkhole attack, the alerted set includes all neighbors [Kro07c].

5.7 The Intrusion Detection Problem

Intrusion detection not only means to detect that some node has been attacked, it also includes identifying the source of an attack. In our case, the cooperative intrusion detection process is triggered by an attack and the subsequent alerts by the local alert modules of the neighboring sensors. The process ends by having the participating sensors jointly *expose* the source.

More formally, the predicate $expose_s(s')$ is true if node s exposes node s'. The intrusion detection problem can now be defined as follows:

Definition (Intrusion Detection Problem (IDP))**.** *Find an algorithm that satisfies the following properties:*

- *If an honest node s exposes a node s', then s is in the alerted set and s' is the source, i.e., $\forall s \in S : honest(s) \wedge expose_s(s') \implies A(s) \wedge source(s')$.*
- *If the attacker attacks, then at most after some time τ all honest nodes in the alerted set expose some node.*

Note that both parts of the IDP definition are necessary. The first property refers to the aspect of *partial correctness*: If the algorithm does something, then it satisfies some input/output relation. It basically restricts the behavior of honest nodes to output something meaningful (if they output anything). Disregarding the first property would allow implementations that output information which is not useful.

The second property refers to the aspect of *termination*: The algorithm actually has to do something. Disregarding the second property would allow implementations that output nothing. Separating the two properties therefore separates concerns and makes reasoning about the problem simpler.

5.8 Conditions for Solving Intrusion Detection

The idea of cooperative intrusion detection is to exchange the outputs of local alert modules, thereby narrowing down the set of possible nodes that could be the attacker. In the following we assume that nodes have no other way to learn anything about the attacker than using their alert modules.

As an initial example, consider the case depicted in Figure 5.3(a). Node p suspects the source q, i.e., $D(p) = \{q\}$. Being Byzantine, q can claim to output $D(q) = \{p\}$. Since p implicitly knows that it is honest, it will ignore the information provided by q and expose q, solving IDP.

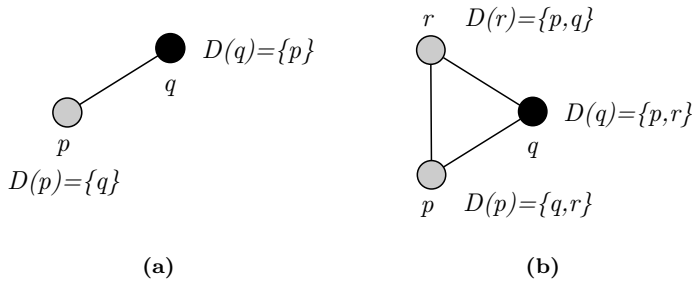

Figure 5.3: *Different types of alerted neighborhoods. Sources of attacks are marked black. In case (a) the IDP is solvable, while in case (b) the IDP is not solvable.*

Now consider a slightly updated example (see Figure 5.3(b)). There three nodes p, q, and r all suspect each other (node q is the source). Every node occurs in exactly two suspect sets, p cannot distinguish node r from node q, if it only may use the suspect sets. We conclude that it is impossible to solve IDP in this case.

Generalizing these two examples, the question arises about general conditions for the solvability of the intrusion detection problem (IDP). In general, two types of conditions are interesting: Necessary conditions and sufficient conditions. A condition is *sufficient* for IDP iff the truth of the condition implies

that there exists an algorithm solving the IDP. A condition is *necessary* iff the existence of an algorithm to solve IDP implies that the condition is true.

In the following, we give necessary (Section 5.8.2) and sufficient (Section 5.8.1) conditions for solvability of IDP using a deterministic algorithm for $t = 1$ in our system model. We show the relationship between IDP and Byzantine Agreement in Section 5.8.3.

5.8.1 Sufficient Conditions for Solving IDP

The Intrusion Detection Condition (IDC)

We now give a sufficient condition for IDP solvability for $t = 1$ and deterministic algorithms. The intuition behind the condition is a generalization of the observation made in Figure 5.3(b): If the suspect sets about some node s are structurally equivalent to those of the source, then the problem is in general not solvable.

Formally, for a node s we define the set $AN(s)$ to be the set of *alerted neighbors* of s, i.e.:

$$AN(s) = \{t | A(t) \wedge t \in N(s)\}$$

Furthermore, we define the set of alerted neighbors of p with respect to q $\tilde{AN}(p, q)$ to be the set of alerted neighbors of p without q, i.e.:

$$\tilde{AN}(p, q) = AN(p) \setminus \{q\}$$

As an example, consider Figure 5.3(b). Here, all three nodes are in alert mode and $AN(s) = D(s)$. The value of $\tilde{AN}(b, a)$ is the information content of $AN(b)$ that is valuable to a. Since a itself knows that it is honest, it will exclude itself from the set $AN(b)$, yielding $\tilde{AN}(b, a) = \{c\}$.

Definition (Intrusion Detection Condition (IDC))**.** *The intrusion detection condition (IDC) is defined as:*

$$\forall p, q \in S : source(q) \implies \tilde{AN}(p, q) \neq \tilde{AN}(q, p)$$

Roughly speaking, IDC means that no other node has the same alerted neighborhood as the attacker. If p and q are neighbors, both are in each other's neighborhood and so they are always different. To exclude this case, we defined $\tilde{AN}$. Note that if p and q are not neighbors, then IDC simplifies to:

$$\forall p, q \in S : source(q) \implies AN(p) \neq AN(q)$$

Theorem 1 (Sufficiency of IDC)**.** *If $t = 1$, IDC is sufficient for ID, i.e., if IDC holds then IDP can be solved.*

Proof. Let all alerted nodes exchange their suspected sets. This is possible in our system model because each pair of honest nodes is connected by a path consisting of honest nodes, and communication is reliable.

Note that the attacker can also go into alert mode. Moreover, it can send different suspected sets to different nodes. However, as we assume that all nodes know their 2-hop-neighborhood, the suspected set of the attacker may only contain its neighbors. Otherwise, the attacker's messages would be discarded.

Consider the suspected sets received by an honest node p. As the attacker is included in the suspected set of every honest node, and no honest node has the same alerted neighborhood as the attacker, no honest node can be suspected by more nodes than the attacker. Thus, if some node is suspected by more nodes than all other nodes, this node can be immediately identified as the attacker.

A more complicated case arises when there are two or more nodes which are suspected by the same number of nodes. This situation can arise, e.g., if the attacker also goes into the alert mode and accuses some of its neighbors.

We denote the attacker as q. Assume that there is a node $p \neq q$ which is suspected by the same number of nodes as q. How can a node r distinguish between q and r?

(1) If $p = r$, then r knows that it is honest, and exposes q.

(2) Consider $p \neq r$. If all honest nodes suspect p, then the IDC does not hold. Thus, for some honest node s holds: $p \notin D(s)$ and $q \in D(s)$. It follows that q is alerted and $p \in D(q)$, as the number of nodes which suspect p is the same as the number of nodes which suspect q.

Node r must now decide which of nodes s and q lies about their suspicion. We now show that there is an alerted node v which is not neighbor of s. Indeed, if all alerted nodes were neighbors of s, than s and q would have the same alerted neighborhood with respect to each other, which contradicts the IDC. Thus, node r has to find out which of the nodes s and q is not a neighbor of some alerted node. This is possible as all nodes know their 2-hop neighborhood. This node has to be honest, and the remaining node is identified as the attacker. □

As an example, consider Figure 5.4. Nodes s and r are honest nodes and alerted. Node p is also honest, but not alerted. The attacker is node q, which is alerted. In this example, nodes p and q are both suspected by two nodes. How can node r distinguish the attacker?

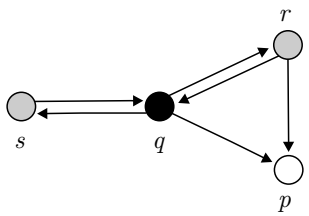

Figure 5.4: *Node q is the attacker, nodes s, r and q are alerted, while p is not alerted and it is marked white. $x \to y$ means that node x suspects node y. $D(r) = \{q, p\}$, $D(q) = \{p, r, s\}$, and $D(s) = \{q\}$.*

IDC holds here:

- $\tilde{AN}(p, q) = \emptyset$, $\tilde{AN}(q, p) = \{s\}$

- $\tilde{AN}(r,q) = \{p\},\ \tilde{AN}(q,r) = \{p,s\}$
- $\tilde{AN}(s,q) = \emptyset,\ \tilde{AN}(q,s) = \{p\}$

Nevertheless, node p collects two suspicions for each of q and r. Thus, either q or s is lying about their suspicions. However, nodes r and s are not neighbors, and therefore, s cannot be the attacker. (In this example, the node v from the proof is equal to r.)

The Neighborhood Conditions (NC)

What happens if IDC is not satisfied? Can IDP still be solved, or is IDC also a necessary condition for solving IDP?

In the following we show that IDC is not a necessary condition. We give another sufficient conditions for IDP solvability which can be valid in the network independently of the validity of the IDC.

Definition (Neighborhood Conditions)**.** *The Neighborhood Conditions (NC) consist of two conditions:*

- NC_1. *All neighbors of the attacker are alerted.*
- NC_2. *If two or more nodes are suspected by the majority of nodes, then all honest nodes suspected by the majority have non-alerted neighbors.*

Theorem 2 (Sufficiency of NC)**.** *If the NC holds, i.e., NC_1 and NC_2 hold, then the IDP can be solved.*

Proof. We give an informal reasoning here. The details of the algorithm for this case are given in Section 6.4.5.

Let all alerted nodes exchange their suspected sets. If only one node is suspected by the majority of nodes, then this node is the attacker, as all neighbors of the attacker are alerted (NC_1). If there are two or more nodes which are suspected by the majority, the nodes in the alerted set have to find out which of these nodes have non-alerted neighbors. According to NC_2, only the attacker does not have non-alerted neighbors. □

5.8.2 Necessary and Sufficient Conditions for Solving IDP

We now show that for the solvability of IDP either the IDC or the NC (i.e., NC_1 and NC_2) should be satisfied in the sensor network.

Theorem 3. *IDP can be solved using a deterministic algorithm if and only if the IDC or NC holds.*

Proof. As shown in Theorems 1 and 2, if IDC holds or if NC holds, then the intrusion detection problem can be solved. We now show that the IDC or the NC is necessary for the solvability of the IDP. It suffices to show the following:

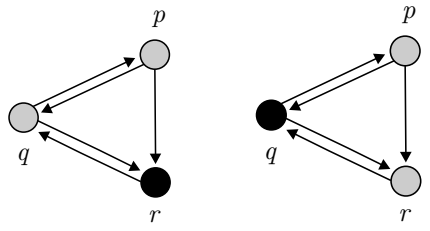

Figure 5.5: *Case (a): Node p suspects q and r, node q suspects p and r, node r is the attacker and suspects q. IDC and* NC_2 *are not satisfied. Case (b): The suspicions remain as in case (a), but node q is the attacker. No algorithm for solving the IDP can distinguish between (a) and (b). Therefore, it is impossible to expose the attacker.*

if the IDC does not hold and the NC does not hold, then the IDP cannot be solved.

Assume that the above claim is not true. That is, there exists a deterministic algorithm $\mathcal{A}$ that always exposes the attacker in case both the IDC and the NC do not hold. Consider Figure 5.5(a). The IDC does not hold there because $\tilde{AN}(p,r) = \tilde{AN}(r,p) = \{q\}$. Also NC does not hold, because NC_2 does not hold: The attacker r and the honest node q are suspected by two nodes, but q does not have any non-alerted neighbors. In this case, the algorithm $\mathcal{A}$ should expose r. However, the situation in Figure 5.5(b) is exactly the same as in (a) from $\mathcal{A}$'s point of view. The suspicions remain the same, the topology also does not change. Thus, there is no additional information to help $\mathcal{A}$ to distinguish between situations (a) and (b). However, $\mathcal{A}$ should be able to distinguish between (a) and (b) and to expose r or q accordingly. It follows that $\mathcal{A}$ does not exist. □

5.8.3 Byzantine Agreement vs. Intrusion Detection

In IDP, the honest nodes have to jointly expose the attacker. That is, they have to reach agreement on the attacker's identity. At first glance, this looks similar to Byzantine Agreement [Pea80], where the nodes have to reach agreement on their inputs. Nevertheless, we show that these two problems cannot be reduced to each other. In some cases, Byzantine Agreement can be solved, whereas Intrusion Detection is not solvable. On the other hand, sometimes Intrusion Detection is solvable, whereas Byzantine Agreement is not.

Consider Figure 5.6(a). Here, the three nodes p, q and r are connected, q is the attacker. It participates in the protocol and suspects both p and r. The honest nodes, on the other hand, both suspect q. In this case, Intrusion Detection is trivially solvable. However, Byzantine Agreement for three participants with $t = 1$ cannot be solved [Pea80].

In Figure 5.6(b), all nodes suspect each other. IDC does not hold, as nodes

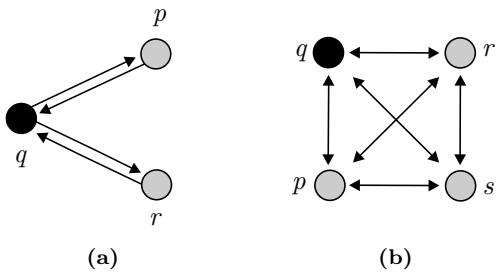

Figure 5.6: *Byzantine Agreement and Intrusion Detection cannot be reduced to each other. Case (a): Honest nodes p and r both suspect only the attacker q, thus Intrusion Detection can be solved, but Byzantine Agreement is not solvable. Case (b): Intrusion detection cannot be solved, Byzantine Agreement is solvable.*

s and q have the same alerted neighborhood with respect to each other. NC also does not hold, as no node has non-alerted neighbors. Thus, Intrusion Detection is not solvable. However, Byzantine Agreement for $t = 1$ can be solved here [Pea80].

5.9 Conclusions

In this chapter we argued that an IDS for sensor networks should locate its detection agents inside all nodes. This will provide partial views of the attack, which can be combined through a cooperative mechanism and provide the nodes with strong evidence of the attack. We then made a first attempt to formalize the problem of intrusion detection in sensor networks, and showed the benefits and theoretical limitations of the cooperative approach to intrusion detection. We presented necessary and sufficient conditions for successfully exposing the attacker under a general threat model. For the needs of the proofs, we used a strict theoretical model, which can be weakened to reflect the conditions in realistic sensor networks. We discuss how to do this in the following chapter.

Chapter 6

A Cooperative Intrusion Detection Algorithm

6.1 Introduction

In the previous chapter we generalized the problem of intrusion detection for sensor networks and defined an abstract framework under which we were able to prove necessary and sufficient conditions for the successful detection of the attacker, assuming that there is only one such node ($t = 1$). As we saw, if there is no other node that has the same alerted neighborhood as the attacker (IDC), then we can detect the attacking node. If this condition does not hold, we can still detect it: it is sufficient that all neighbors of the attacker are alerted and that any honest node that is equally suspected with the attacker has non-alerted neighbors (NC). More interestingly, we showed that if none the above conditions hold, then there is no algorithm to detect the attacker. But if a condition does hold, it remains to define specific algorithms for the detection. This is the subject of this chapter. We present an intrusion detection algorithm, which is based on the cooperation of the honest nodes and reveals the attacker.

The algorithm is implemented by an IDS agent installed in all sensor nodes. The agent runs independently from the application and monitors communication activity within the radio range of the host node. When an attack is launched, the corresponding agents dynamically become activated around the attacking node and collaborate with each other in order to share their partial views, agree on the identity of the source and expose it. As a result, the nodes collectively form an IDS system to defend the sensor network.

The general functionality of the IDS agent can be described as follows:

- Network Monitoring: Every agent performs packet monitoring in their immediate neighborhood collecting audit data.
- Decision Making: Using this audit data, every agent decides on the intru-

sion threat level on a host-based basis. Then they publish their findings to their neighbors and make the final collective decision.

- Action: Every agent has a response mechanism that allow it to respond to an intrusion situation.

The communication amongst the clients allows us to use a distributed algorithm and show that with collaborative processing an IDS can become lightweight enough to be realistic for sensor networks.

The rest of the chapter is organized as follows. In Section 6.2, we define our system model and discuss how it is related to the model we defined in Chapter 5. In Section 6.3 we state our threat model and in Section 6.4 we describe our algorithm in detail, sectioned in phases. Then, Section 6.5 presents a system perspective of the IDS, describes its modules and how they are interconnected. Section 6.6 elaborates on the probability that our IDC and NC conditions hold in practice, through simulation results. Finally in Section 6.7, we present experimental results of the IDS implementation on the motes and investigate its performance in real scenarios. Section 6.8 concludes this chapter.

6.2 System Model

For the proofs in Chapter 5 we required reliable and timely communication as part of our assumptions. In principle, we can also use weaker system models as long as they allow some protocols for reliable and timely exchange of suspected sets with high probability. For example, in this chapter we use an advertise-request protocol to guarantee this. So, our system model can be weakened and consider unreliable links and unpredictable delays for the wireless communication. When a node transmits a packet, it does not know which nodes successfully received the message, since the MAC layer of the receivers does not send any acknowledgments or requests for retransmissions. A node may miss to receive a message, either because a collision occurs or because its radio is not available at the time of the transmission.

In the previous chapter, we also assumed a static and symmetric neighborhood relation. This assumption can also be weakened. All we need is that the nodes have secure information on their 2-hop neighborhood which do not change during a particular protocol run. In this chapter, we let the nodes find out their neighborhood in the secure initialization phase, where the attacker is absent. The neighborhood tables which are used for intrusion detection are then fixed. In case that a neighbor crashes, it looks in the protocol as if the node was not alerted. This can be tolerated as long as the IDC holds (the NC does not hold in this case). On the other hand, if some new neighbors arrive, there must be a secure protocol for their addition and the update of the neighborhood tables, but we have left this as a future work. So, for this chapter we do not support new nodes addition.

6.3 Threat Model

We follow the same threat model as in the previous chapter, where the attacker can capture a number of t nodes to launch an attack and we distinguish among these t nodes one single node which is the *source* of the attack, i.e., this node is the first to behave in a faulty way. All non-source faulty nodes are called *collaborators*. We assume that collaborators are neighbors of the source of an attack. Thus an attacker can have $t-1$ collaborators. To be consistent with Chapter 5, we will assume that $t = 1$. In this way the necessary and sufficient conditions will be valid, and the algorithm presented here is nicely connected with the theorems presented in the previous chapter.

6.4 The Algorithm

We partition our algorithm into the following phases: The initialization phase, where nodes construct and share some secret values, the voting phase, where nodes exchange their lists of suspected nodes and the key verification phase, where nodes publish their keys and verify the authenticity of the signed votes. Then, if IDC holds, as shown in Figure 6.1, we can move to the final phase of exposing the attacker. If it does not hold, but NC holds, we add one extra phase, the external ring reinforcement phase. In the following sections we describe each phase in detail.

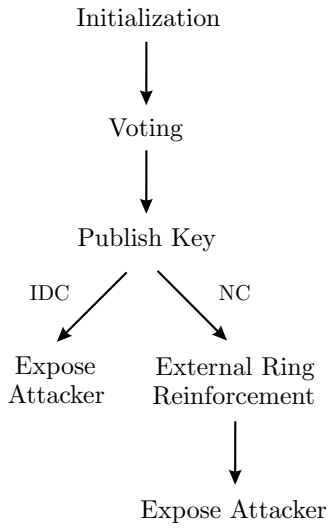

Figure 6.1: *The phases of the IDS algorithm. If IDC does not hold, but NC holds, we add one extra phase, the external ring reinforcement.*

6.4.1 Initialization Phase

The initialization phase takes place right after the network deployment. The duration of this phase is short enough so that we can safely assume the absence of an attacker in the network's vicinity. We also require that all nodes have discovered their immediate neighbors, which is a standard procedure after deployment in almost all routing protocols.

During this phase all nodes discover their 2-hop neighborhood by broadcasting their IDs with a packet that has a TTL field equal to 2, meaning that each packet will be forwarded only once by the sender's 1-hop neighbors. The discovered neighborhood information is stored in a table, which we call the 2-hops neighborhood table.

Next, each node generates a one-way key chain of length n,

$$(K_0, K_1, \ldots, K_{n-1}, K_n),$$

using a pre-assigned unique secret key K_n and a one-way hash function F. As the last step in the initialization phase, each node announces the resulted has chain commitment K_0 to all of its 1-hop and 2-hop neighbors following the same procedure described above for the 2-hop neighborhood discovery.

6.4.2 Voting Phase

During the voting phase each alerted node sends its vote to all the other members and respectively collects their votes (see Algorithm 1). Let us denote the message that bears the vote from node s as $m_v(s)$. Each vote consists of the nodes suspected by the sender, so for node s,

$$m_v(s) = id||D(s).$$

Node s "signs" its vote calculating the MAC with the next key K_j from its one-way key chain, and broadcasts

$$m_v(s), MAC_{K_j}(m_v(s)).$$

Following that, it sets a timer T_v to expire after time τ_v. During that time it waits to receive the votes of the rest of the alerted nodes and buffers them, as it has to wait for the key publishing phase in order to authenticate them.

The vote of each alerted node needs to reach all other alerted nodes, which means that we need a forwarding mechanism. Since the messages are signed with a key known only to the sender, the attacker cannot change the votes. However, we make no assumptions about the behavior of the attacking node. This means that the attacker may refuse to forward votes from its neighbors. For example, in Figure 6.2, the attacker may choose to drop the votes it receives, so they must be forwarded through other paths, bypassing the attacker. Note that these paths can consist of more than two hops.

To ensure that the votes propagate to all alerted nodes, we follow a broadcast message-suppression protocol, similar to SPIN [Kul02]. When an alerted

Algorithm 1: The *Voting* algorithm

Data: IDs of alerted nodes
Result: Vector of collected votes
begin
  Create $m_v(s) = id||D(s)$;
  Calculate $MAC_{K_j}(m_v(s))$ using K_j;
  Broadcast $m_v(s), MAC_{K_j}(m_v(s))$;
  Set timer $T_v = \tau_v$;
  while ![T_v(*expired*)] **do**
    if *receive* $m_v(q)$ **then**
      Store $m_v(q)$ and corresponding MAC;
      Broadcast $m_{adv}(q)$; // ADV message
    end
    if [*receive* $m_{req}(q)$] **then**
      Broadcast $m_v(q), MAC_{K_j}(m_v(q))$;
    end
    if [*receive* $m_{adv}(q)$]&&[*don't have* $m_v(q)$] **then**
      Start timer T_{req};
    end
    while ![T_{req}(*expired*)] **do**
      Register overheard $m_{req}(q)$;
    end
    if [T_{req}(*expired*)] **then**
      if !*overheard* $m_{req}(q)$ **then**
        Broadcast $m_{req}(q)$; // REQ message
      end
    end
  end
end

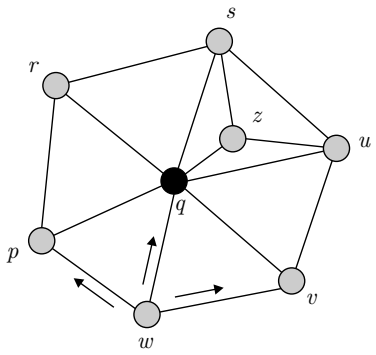

Figure 6.2: *Set of alerted nodes around the attacker q. Node w broadcasts its vote, which needs up to three hops to reach all other alerted nodes, in case that q does not participate in the protocol.*

node receives a vote, it advertises it, by broadcasting an ADV message. Upon receiving an ADV, each neighboring node checks to see whether it already has received or requested the advertised vote. If not, it sets a random timer T_{req} to expire, uniformly chosen from a predetermined interval. When the timer expires, the node sends a REQ message requesting the specific vote, unless it has overheard a similar REQ from another node. In the latter case, it cancels its own request, as it is redundant.

6.4.3 Publish Key Phase

In the Publish Key phase each node broadcasts the next key of its hash chain, K_j, which was used to sign the vote. When a node receives the disclosed key, it can easily verify the correctness of the key by checking whether K_j generates the previous one through the application of F. If the key is correct, it replaces the old commitment K_{j-1} with the new one in its memory. The node can now use the key to verify the signature of the corresponding vote stored in its buffer from the previous phase. If this process is successful, it accepts the vote as authentic.

We allow sufficient time for the nodes to exchange their keys by setting a timer T_p. This timer is initialized just after a node publishes its own key and it is set to expire at time τ_p. During this time period, the nodes follow the same ADV-REQ scheme that we described for the exchange of votes. That is, when an alerted node acquires a key, it advertises it to its neighbors and they request it sending the corresponding REQ message. In this way, the keys can propagate to all the alerted nodes, even if the attacker does not participate in the process.

When the timer expires, the nodes move to the final step of processing the votes and exposing the attacker. In the case where a key has been missed, the corresponding vote is discarded. The code for this phase is given in Algorithm 2.

Since nodes are not time synchronized, and some nodes may start publishing

Algorithm 2: The *Publish Key* algorithm

```
Data: Buffer of received votes
Result: Attacker's ID
begin
    Broadcast key K_j;
    Set timer T_p = τ_p;
    while ![T_p(expired)] do
        if receive K_i then
            if Verify(K_i) && Authenticate(m_v^i) then
                Store K_i;
                Broadcast m_adv(K_i);
            end
            else
                Discard m_v;
            end
        end
        if [receive m_req(K_i) ] then
            Forward the requested key K_i;
        end
        if [receive m_adv(K_i) ]&&[ don't have K_i] then
            Start timer T_req;
        end
        while ![T_req(expired)] do
            Register overheard m_req(K_i);
        end
        if [T_req(expired)] then
            if !overheard m_req(K_i) then
                Broadcast m_req(K_i);
            end
        end
    end
end
```

their keys while others are still in the voting phase, we need to consider "man in the middle" attacks. When a node sends its vote, an attacker may withhold it until that node publishes its key. Then it can change the vote, sign it again with the new key, and forward it to the next alerted node. Following that, the attacker also forwards the key, and the receiver will be able to verify the signature and accept the fake vote as authentic.

An explicit defense against the previous attack would be to require the nodes to be loosely synchronized as in μTESLA [Per02]. Here, however, we have decided to keep things simple and deal with this problem implicitly by relying on *residual paths* amongst the nodes (although we plan to investigate the synchronization approach and consider its possible benefits). As votes are forwarded by all nodes, even if an attacker refuses to forward a vote, it will arrive to the intended recipients via other paths. We also take some additional measures in our algorithm having a node accepting a vote only while it has not publish its own key and it has not received the key from the node that sends the vote.

Another issue that we must discuss at this point is the key chain, which is a core element of our algorithm. We must stress that its length is finite and at some point all the available keys will have been used. Older keys cannot be reused, since they have been revealed by the node. Therefore, the nodes should be able to regenerate the key chain in a possibly compromised environment. To do that we follow the same method that we described in Chapter 3: before the node uses the last commitment, it creates a new hash chain $K'_0, K'_1, \ldots, K'_{n-1}, K'_n$ and broadcasts the new commitment K'_0 authenticated with the last unused key of the old chain. This essentially provides the connection between the two chains and the alerted nodes will be able to authenticate the votes as before.

6.4.4 Exposing the Attacker

When each alerted node $s_1, s_2, \ldots, s_n$ has collected and authenticated the votes from all the other alerted nodes, it will have knowledge of all the corresponding suspect lists, $D(s_1), D(s_2), \ldots, D(s_n)$, itself included. Then it applies a local operator on these lists which will produce the final intrusion detection result, i.e., the attacker's ID. In particular it applies a count operator which counts the number of times δ_i each node i appears in the suspect lists, or else the number of votes it collects. All alerted nodes will reach the same result, since they all apply the same operator on the same sets. Here we distinguish three cases:

- IDC holds. In Section 5.8.1, we proved that IDC is a sufficient condition for the intrusion detection problem. Then, we know that if a node collects more nodes than all other nodes, this node is the attacker. There is also the case where two or more nodes collect the same number of votes. In this case, we proved that still the nodes can distinguish the attacker, by inspecting their 2-hop neighborhood table.
- IDC does not hold and NC holds. In this case, if there is one node holding the majority of the nodes, we again know that this is the attacker.

If not, then the honest nodes which also collect the majority have non-alerted neighbors. So, the nodes move to the external ring reinforcement phase, where these neighbors are called to vote and support their honest neighbors. We will see in Section 6.4.5 that in this case the attacker is revealed.

- IDC does not hold and NC does not hold. Then no safe conclusions can be drawn. As we proved in Section 5.8.2, IDC or NC needs to hold in order to find the attacker. In different case there is no algorithm to solve the intrusion detection problem.

6.4.5 External Ring Reinforcement Phase

As we said, when there are other nodes that have the same set of alerted neighbors $\tilde{AN}$ with respect to the attacker, i.e., IDC does not hold, the voting process may be inconclusive, if these nodes collect the same number of votes. In this section we present an algorithm where, if NC holds, the alerted region can distinguish amongst the prevailing candidates and find the actual one. So, for what follows we assume that NC holds, meaning that *all* neighbors of the attacker are alerted and that honest nodes collecting the majority of the votes have non-alerted neighbors.

Let us assume the set $\mathcal{P} = \{p_1, p_2, \ldots, p_k\}$ of the nodes collecting the same number of votes as the attacker, including the attacker itself. According to NC, the nodes in $\mathcal{P}$ do not have exactly the same neighborhood. Honest ones will also have other neighbors, which are not in alerted state and are going to play an important role in this phase. Therefore, we make the following definition:

Definition (External Ring)**.** *The external ring is defined as the set of nodes which are not alerted, but they are directly reachable by an alerted node, i.e., any of them is a direct neighbor of at least one node in the alert region.*

Figure 6.3 shows an example where nodes 96 (the attacker) and 76 have the same alerted neighborhood, and therefore collected the same number of votes during the voting phase, i.e., $\mathcal{P} = \{96, 76\}$. The circle in the figure shows the neighborhood of the attacker. The nodes in the external ring are marked as white, while the alerted nodes as gray.

The neighborhood of node 76 also includes the nodes 81 and 79, which are not alerted. These two nodes know that their neighbor 76 is not the attacker, since they are not alerted. If they share this information with the nodes in the alerted region, they can help them distinguish the attacker. Therefore, they somehow come to support their neighbor. This can reinforce the intrusion detection process because there will be no node to support the attacker. If such a node existed it would be the attacker's neighbor and consequently, member of the alerted region.

The external ring reinforcement phase is initiated by the nodes in the alerted region. They broadcast a request to their non-alerted neighbors, including the

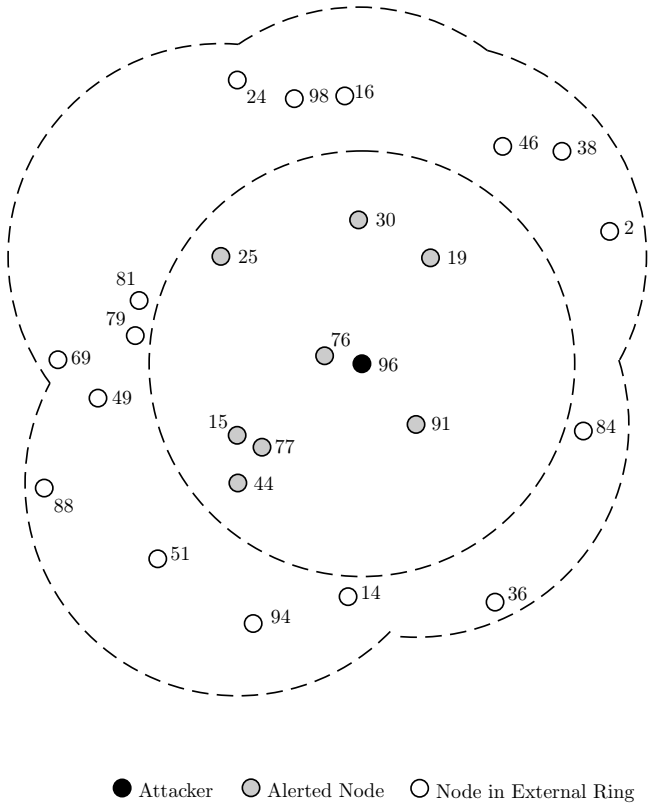

Figure 6.3: *The attacker's external ring is defined by the nodes which are 2-hops away from the attacker.*

set $\mathcal{P}$ in the message. The intended receivers check to see if any nodes in $\mathcal{P}$ are their neighbors and broadcast a message voting *in favor* of them.

The response message sent by any node of the external ring is forwarded by alerted nodes as in the voting and publish key phases, so that it can reach all nodes in the alerted region. It is also signed using the next key in the key chain of the sender. The key is released after some fixed period of time and used by the receivers to authenticate the message.

For the example shown in Figure 6.3, nodes 81 and 79 will vote in favor of node 76. Since there are no other nodes who have a neighbor from $\mathcal{P}$, there will be no more votes. Hence, node 76 will collect two votes and node 96 none. Now the alerted nodes can conclude on the attacker successfully.

6.4.6 Responding to the Attack

Once the network is aware that an intrusion has taken place and have detected the attacker, appropriate actions must be taken as a response. The first action

is to cut off the intruder as much as possible and isolate the compromised nodes. After that, proper operation of the network must be restored. This may include changes in the routing paths, updates of the cryptographic material (keys, etc.) or restoring part of the system using redundant information distributed in other parts of the network. Depending on the confidence and the type of the attack, we categorize the response to two types:

- *Direct response*: Excluding the suspect node from any paths and forcing regeneration of new cryptographic keys with the rest of the neighbors.
- *Indirect response*: Notifying the base station about the intruder or reducing the quality estimation for the link to that node, so that it will gradually loose its path reliability.

Intrusion detection systems in other types of networks always report an intrusion alert to a human, who takes the final action. This approach is usually neglected in relevant literature. Sensor networks should (and they actually are) able to demonstrate an autonomic behavior, taking advantage of their inherent redundancy and distributed nature. Autonomic behavior means that any response to an intrusion attempt is performed without human intervention and within finite time.

6.5 The IDS Architecture

Having covered the algorithmic part of our IDS, we now emphasize on the conceptual modules that implement this algorithm. They basically define the architecture of the IDS agents, who reside in all nodes of the network. They can broadcast messages and communicate with the agents in the neighboring nodes, allowing the realization of our distributed algorithm. Figure 6.4 shows analytically the architecture. The functionality of each module is outlined below.

- The `LocalPacketMonitoring` module gathers audit data to be provided to the `LocalDetection` module. As we said, audit data in a sensor networks IDS can be the communication activities within the node's radio range.
- The `NbPerimeter` module is responsible for maintaining consistent information about 1-hop and 2-hop neighbors of the nodes. Information about 2-hop neighbors is needed because the detection process involves the communication of the nodes which are neighbors of the (yet unknown) attacker, but they might be 2-hops away from each other.
- After the deployment of the sensor network, the `KeyManagement` module of the node generates a one-way key chain of length n, using a pre-assigned unique secret key K_n. Then it announces the hash chain commitment K_0 to the rest of the alerted nodes. The `KeyManagement` module also stores

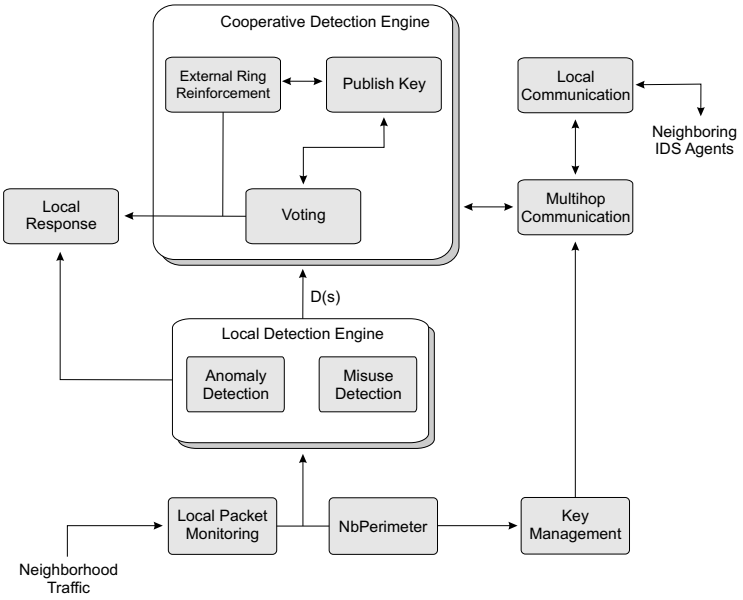

Figure 6.4: *The IDS Architecture.*

the corresponding information from them, i.e., the node IDs and their keys. This information needs to remain consistent and up-to-date during the lifetime of the network. So, the `KeyManagement` module updates the corresponding key every time a node publishes a new one from its key chain.

- The `LocalDetectionEngine` collects the audit data and analyzes it according to some given rules. A set of rules is provided for each attack, and whenever one or more rules are satisfied, a local alert is produced by the module. When we say that the `LocalDetection` engine of a node s produces an alert, what we mean is that it simply outputs some set $D(s)$ of the suspected nodes. The size of $D(s)$ depends on the quality of the alert module and the nature of the attack. If $|D(s)| = 1$, then the sensor has identified the source of the attack and the IDS protocol moves to local response. In different case, $D(s)$ is passed on to the `CooperativeDetection` engine.

- The `Voting` module is the first part of the `CooperativeDetection` engine. It implements the voting algorithm of Section 6.4.2. The module sends out the suspect list $D(s)$ and collects the corresponding lists from the rest of the alerted nodes.

- The `PublishKey` module takes over after the voting and runs the algorithm described in Section 6.4.3, where the node publishes the next key of its hash chain and collects the keys from the rest of the alerted nodes.

Then it authenticates the received votes and extracts the suspect lists. Aggregation of these lists may or may not give a majority to a node. In the former case the result is given directly to the `LocalResponse` module. In the latter case, and if NC holds, the `ExternalRingReinforcement` module initiates the next phase.

- The `ExternalRingReinforcement` module follows the protocol that is described in Section 6.4.5. Given that NC holds, the identity of the attacker will be revealed. Then it is passed to the `LocalResponse` module.
- The `LocalResponse` module takes appropriate actions against the attacking node, as discussed in Section 6.4.6.

This architecture results in a very lightweight IDS agent that can perfectly fit into the memory of a mote and requires a small amount of its computational and communication resources. More evidence on the resources consumption and performance of this architecture are given in the following sections.

6.6 Simulation Results

We have simulated a sensor network of 100 nodes placed uniformly at random in order to test our proposed intrusion detection algorithm. For each run of the simulation, we chose at random one node to be the attacker. This way we could have the neighbors of that node being in alert mode and apply the intrusion detection algorithm, assuming of course that they did not know the attacker. In what follows in this section, we assumed that all neighbors of the attacker are alerted.

Figure 6.5 shows the probability that there exist other alerted nodes having the same alerted neighborhood as the attacker, i.e., the probability that IDC does not hold. We presented an example of this case in Figure 6.3, where nodes 96 (the attacker) and 76 have the same $\tilde{AN}$. Then, as we saw, the rest of the nodes in the alert region are unable to conclude on the attacker. As expected, this probability drops as the average number of neighbors increases, but still remains high (close to 30% for a very dense network). That's why the external ring reinforcement phase is necessary to complete the protocol.

During the external ring reinforcement phase, nodes not being alerted are allowed to vote, in order to distinguish the attacker between the nodes in the set $\mathcal{P}$, given that all nodes of the attacker are in alert mode. This can only be successful when the rest of the nodes in $\mathcal{P}$ do not have exactly the same neighborhood as the attacker. Otherwise, this phase and therefore the intrusion detection will fail. Figure 6.5 shows this probability, i.e., the probability that there exists a node $s \in \mathcal{P}$ with the same neighborhood as the attacker. This is equivalent to the probability that NC does not hold. Of course, as the network becomes more dense, this probability drops, and for more than 7 neighbors in average it becomes less than 10%.

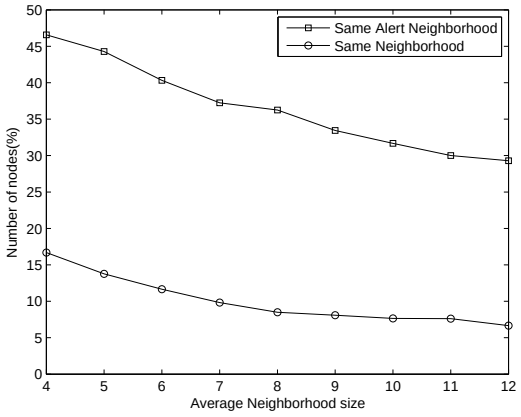

Figure 6.5: *Probability that there exists a node with the same alert neighborhood as the attacker. This is much higher than the probability that a node has exactly the same neighborhood as the attacker. The former reflects the failure of the IDC, while the latter the failure of NC and therefore the impossibility of the intrusion detection problem.*

In Figure 6.6 we have also calculated the probability that the IDS successfully identifies the attacker. To do this we have run the protocol in $10,000$ different topologies, choosing each time a random attacker. If the voting phase was conclusive the protocol ended, in different case the external ring reinforcement phase was activated. As expected, the protocol always succeeded, except for the cases where another node with exactly the same neighborhood as the attacker existed.

6.7 Implementation

In this section, we present experimental results from our implementation of the IDS described in this chapter. The goal is to exhibit a framework that actually works on the motes, and can be used as a reference point. Moreover, it will become clear to the reader that such a system for sensor networks is lightweight enough to be a viable and realistic solution from implementation and real deployment perspective.

The current development of the IDS algorithm builds on Moteiv Telos motes, a popular architecture in the sensor network research community. It features the 8 MHz TI MSP430 micro-controller and a 16-bit RISC processor that is well known for its low energy consumption. Yet, even though the implementation is tested on the Telos motes, all the components are designed with adequate generality, such that porting them to different sensor platforms should yield similar performance results.

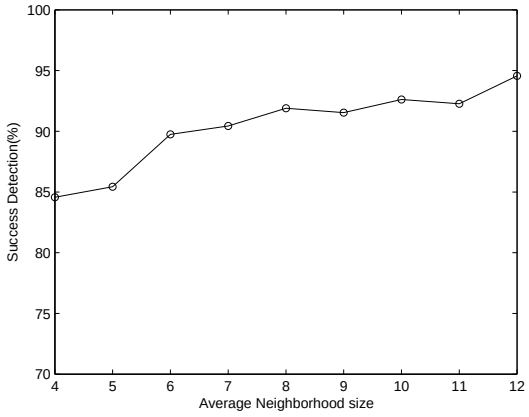

Figure 6.6: *The overall success rate of the protocol.*

6.7.1 Memory Requirements

The memory footprint of the our implementation is an important measure of its feasibility and usefulness on memory constrained sensor nodes. The total memory footprint is composed of the memory footprint of the compiled code, which is present in ROM, and the memory footprint of the data memory required to run the code. An IDS for constrained devices must be compact in terms of both code size and RAM usage, in order to leave room for applications running on top of the system. Table 6.1 lists the memory footprint of the modules, compiled for the MSP430 microcontroller.

Table 6.1: *Size of the compiled code, in bytes.*

Module	RAM usage	Code Size
Neighborhood Discovery	136	968
Exchange of Keys	216	4060
Reliability (ADV-REQ)	104	32
Voting	159	4844
Total	615	9904

The largest module in terms of RAM footprint in Table 6.1 is the Key Management module. This is because the Key Management module contains statically allocated tables for the neighbors and their keys. In terms of ROM, the largest module is the Voting module, since it has the most lines of code. In total, the IDS consumes 615 bytes of RAM and 9,904 bytes of code memory. This leaves enough space in the mote's memory for user applications. For example, the total RAM available in Telos motes is 10 KB.

6.7.2 Experiments

To evaluate the performance of the implementation of the IDS, we tested it in a real environment. In particular we deployed several nodes in random topologies on the floor of an office building. We set a node to be the "attacker" and we gradually incremented the number of its neighbors to form larger alert regions. For each alert region size, we repeated the experiment for 20 different random topologies. Let us note that for these experiments we took care that IDC always held, so that we could always find the attacker. We have covered the scenario that this is not the case in the previous section.

The experiments were performed by having the motes running a typical monitoring application. In particular we loaded the Delta application, where the motes report environmental measurements to the base station every 5 seconds. We also deployed the MultihopLQI protocol at the routing layer, which is an updated version of the MintRoute protocol [Woo03a] for the Chipcon CC2420 radio. We tuned it to send control packets every 5 seconds. Our goal is to demonstrate how well the IDS will function, even under the presence of traffic on other layers. Then we simulated an attack to trigger the IDS protocol.

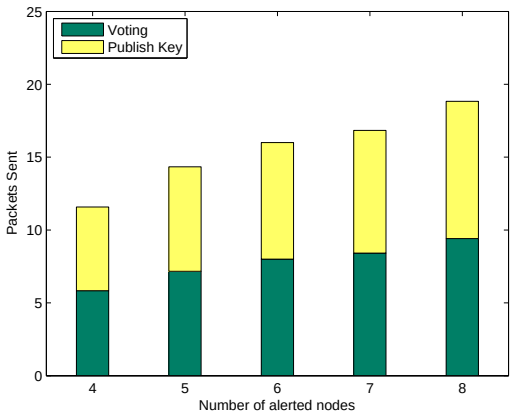

Figure 6.7: *Measured communication cost for different number of alerted nodes.*

Figure 6.7 depicts the communication cost of the protocol measured in packets sent by a node. In particular, we broke it down to the packets exchanged for the voting phase and the publish key phase (as a total of exchanging the votes, ADV, REQ and keys). As it is expected, the number of packets exchanged in the two phases are the same, since the protocol does not change; only the content of the packets does. For small alert region sizes the cost is only about 12 packets, while for more dense regions the cost still remains low (19 packets). This is the total communication cost per attack and involves only the alerted nodes. It is also measured as a mean time averaged on different random topologies. The number of packets depends on the topology and the number of the alerted

nodes, as these parameters determine the number of votes and keys circulated amongst them.

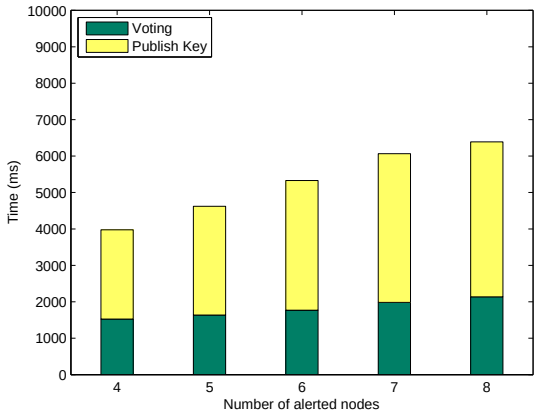

Figure 6.8: *Detection time for different number of alerted nodes.*

Next we measured the time that each phase of the IDS protocol required, i.e., the voting phase and the publish key phase. Figure 6.8 shows the measured mean times for each of the above phases, for different number of alerted nodes (i.e., attacker's neighborhood). What we can infer from Figure 6.8 is that the voting phase has smaller deviation than the publish key phase, as the number of alerted nodes increases and contributes the smallest overhead in the total delay. The most time-consuming phase is the publish key phase, where nodes exchange their keys and verify the votes.

To get a better insight of this, we measured the time needed for the computational operations within this phase. In particular, the time a node needs to authenticate each received key (i.e., to check if the hash of the new key matches with the previous one) is approximately $15\,ms$. The validation of the signature of the vote takes about $25\,ms$ and the aggregation of the received vote with the rest in order to produce the final result takes $150\,ms$. For the construction of its own vote, a nodes needs $60\,ms$ and signing it with its key takes $25\,ms$.

Figure 6.9 expresses the percentage of costs for computation and communication for the publish key phase. We can conclude that most of the overhead arises from the transmission of data rather than from any computational costs. This overhead for the communication is due to the inherent inability of TinyOS to receive the next packet before finishing the processing of the current one. In our implementation, upon receiving a key, the node has to verify it is a valid one before accepting it. To save memory space, we do not buffer the key for later processing, but rather we authenticate it on the fly. Meanwhile, TinyOS cannot receive the next key. That's why we had to include a random delay so that nodes publish their keys in different time instances. This delay, although experimentally minimized, contributes significantly to the results of Figure 6.9.

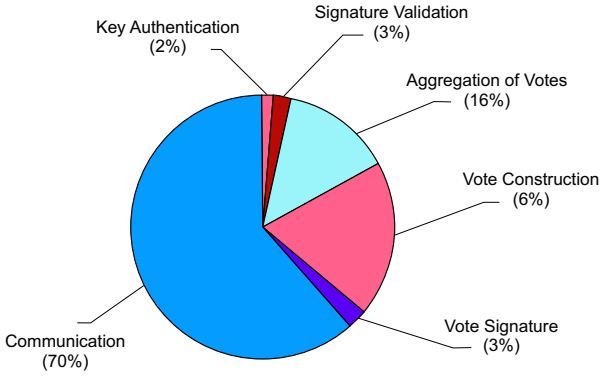

Figure 6.9: *Costs of computation and communication in terms of time for the Publish Key phase.*

It is also important to see how the behavior of the IDS is affected by the traffic introduced by the application layer. That is, if the application needs to increase the data rate of the information routed in the network, the bandwidth that remains for the communication of the IDS agents decreases. We performed experiments to see how this affects the detection delay. In particular, we gradually increased the data rate of the Delta application. In Delta, each sensor node reports measurements periodically sending a packet to the base station. Delta is based on the MultihopLQI at the routing layer, which broadcasts route update packets periodically to maintain the routing tree. We tuned MultihopLQI to send a route update packet every 5 seconds and experimented with different data rates for Delta. For the MAC layer we used the default in TinyOS, i.e., the CSMA protocol.

Figure 6.10 shows the detection delay of the IDS as the aggregative data rate of packets at the routing and application layers increases. This increase actually corresponds to different packet rates of Delta (1 packet every 1, 3, 5 and 10 seconds), as the rate of the route update packets was fixed. The number of alerted nodes was set to 6 throughout the experiments. As we see from the figure, for an increase of 300% in the data rate (from 34.8 bps to 139.2 bps) the detection delay is increased only by 1.6 seconds. As more and more packets are sent and received from the nodes, a delay to exchange the necessary packets for the intrusion detection is unavoidable, due to the CSMA back-off waiting time. We believe that a better MAC layer protocol would improve this delay further.

6.8 Conclusions

In this chapter, we discussed the design of an intrusion detection system for sensor networks that uses a large number of autonomous, but *localized*, cooperating agents in order to detect an attacker. The nodes use coordinated surveillance

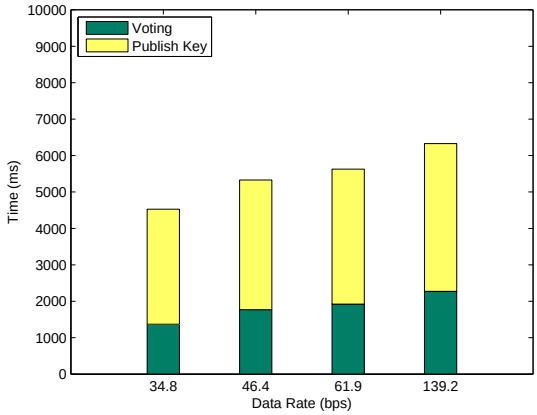

Figure 6.10: *Behavior of the detection process under the presence of traffic on other layers.*

by incorporating inter-agent communication and distributed computing in decision making to collaboratively infer the identity of the attacker from a set of suspicious nodes.

The IDS we discussed is novel and realistic, considering the current state of the art in wireless sensor networks. The demonstrated implementation details show that it is lightweight enough to run on sensor nodes, in terms of communication, energy, and memory requirements. This shows that studying the problem of intrusion detection in sensor networks is a viable research direction and with further investigation it can provide even more attractive solutions for securing such types of networks.

Chapter 7

Conclusions

The difficulties of security in sensor networks mainly stem from the constraints imposed by the simplicity of sensor devices: limited power, limited communication bandwidth and processing capabilities, and small storage capacity. In this thesis, we concentrated on the study and design of distributed security algorithms for sensor networks that prevent the attacker from accessing the information routed within or injecting malicious packets. We also studied the problem of detecting the attacker when the prevention measures cannot succeed and she intrudes the network. For each of the proposed solutions we emphasized on minimizing the overhead imposed by its implementation on widely used sensor platforms, so that it becomes realistic and attractive to the developer. We now summarize the key results of our work and discuss open issues.

7.1 Summary of Main Results

First we focused on the establishment of trust relationship among wireless sensor nodes, and presented a key management protocol for sensor networks. The protocol includes support for establishing four types of keys per sensor node: individual keys shared with the base station, pairwise keys shared with individual neighboring nodes, cluster keys shared with a set of neighbors, and a group key shared with all the nodes in the network. We showed how the keys can be distributed so that the protocol can support in-network processing and efficient dissemination, while restricting the security impact of a node compromise to the immediate network neighborhood of the compromised node. Applying the protocol makes it really hard for an adversary to disrupt the normal operation of the network.

Shared key cryptography can not be used to secure other operations in sensor networks, like network programming, where bulk data have to be disseminated from the base station to the sensor nodes. We therefore presented a method for verifying the integrity and authenticity of such data. The two cryptographic

primitives used in our security protocol are digital signatures and cryptographic hashes. Instead of using RSA or Elliptic Curve Cryptography like proposed by other researchers, we constructed our own r-time signature scheme for efficiently constructing and verifying the digital signatures. This allowed us to use a protocol that imposes asymmetric cryptography properties using solely symmetric cryptographic primitives and drop drastically the energy and time requirements. Our method allows sensor-network developers to configure the security-performance trade-off to suit their particular needs.

We have argued that despite the security offered by the above protocols, a sensor network will have always vulnerabilities that an adversary could exploit. Intrusion detection can complement the intrusion prevention techniques to secure the network. However, new techniques must be developed to make intrusion detection work efficiently for sensor networks. We have argued that such techniques should be distributed and cooperative. If such a scheme is followed, there are some interesting findings. In particular, we proved that if two nodes are not suspected by the same set of neighbors, it is sufficient to distinguish the attacker. Moreover, if an attacker is suspected by all its neighbors, the above condition is also necessary.

Then we provided a cooperative IDS, in which the nodes use coordinated surveillance by incorporating inter-agent communication and distributed computing in decision making to collaboratively infer the identity of the attacker from a set of suspicious nodes. The factor that determined the design of the IDS is that the attacker, as well as any other compromised nodes that collaborate with her, can interfere with the protocol and affect the result. The countermeasures consist of authenticating the exchanged packets and sending them through multiple paths. The communication cost indeed constitutes the highest percentage of the overhead, but still, it is as low as about 15 packets per node. The overall detection time is a few seconds and it successfully concludes on the attacker with probability up to 95%.

7.2 Discussion and Future Research

Some interesting research directions arise from the theorems of Chapter 5. While we provided sufficient and necessary conditions for detecting the attacker in the case of $t = 1$, it remains an open question whether such conditions exist in the case where the attacker can capture more nodes than the source ($t > 1$). In this case, the nodes controlled by the attacker can act as "collaborators" of the source of the attack, trying to prevent the intrusion detection protocol from successfully revealing the attacker. For example, they can join the rest of the alerted nodes, such that they can affect the voting result. In case of $t > 1$, it is also possible that the alert modules of some honest nodes are triggered by different attackers. Clearly, in this case we need to come up with stronger necessary and sufficient conditions for the intrusion detection problem.

In Chapter 6, we presented an algorithm assuming that the attacker can control only one node ($t = 1$), in order to be consistent with theorems of Chap-

ter 5. It is not hard to see however that it can be also applied in the case that $t > 1$. The cryptographic tools that we use can offer the required security for the messages exchanged by honest nodes in our protocol. Since these messages are signed with keys only known to their initial senders, they cannot be altered by intermediate malicious nodes. What a collaborating node can do however, is to include itself in the voting process and try to affect the final result by casting votes against legitimate nodes. The best strategy for them would be to vote against a specific honest node, hoping that it will collect more votes than the attacker. This clearly depends on the number of the collaborators and sets a limit on how many of them our protocol can tolerate. If there is a set of at least $t + 1$ honest alerted nodes, the majority vote will still point to the attacker.

The second thing that a collaborator can do is to drop packets sent by honest nodes. According to the protocol, alerted nodes may be more than one hop from each other, so in order to communicate with each other they are based on intermediate nodes that forward their packets. Being in that path, a collaborator can selectively drop the votes send by some nodes, affecting the final result. Our protocol is based on the massive redundancy of sensor networks and the existence of multiple paths between two nodes to deliver the packets to all participating nodes. As long as this assumption holds, the proposed protocol can be used to identify the attacker, even in the presence of collaborators.

We also plan to look into dynamic neighborhood changes for our IDS system and in particular, into secure node addition and removal in sensor networks. Security in this case is necessary, or else the attacker would be able to introduce her own nodes (i.e., collaborators) and avoid detection. Of course, secure node addition is an independent problem, and it was addressed in Chapter 3, but it is interesting to see how it can be adapted to work smoothly with our intrusion detection system.

Another important topic opens after the solutions provided by Chapter 6. Having designed an IDS architecture that is appropriate for sensor networks, it makes sense to start looking into specific attacks and define rules for the local detection module. That is, rules that based either on misuse or anomaly detection can produce suspect lists. The best way to do that is to look into how specific attacks can be realized in practice and study the methods from the attacker's point of view. This will give a better insight of the attacks and will lead to powerful detection rules. Then, another question comes in the scene: Could we define more general rules that can be applied for detecting a broader class of attacks? Definitely the research area of intrusion detection in wireless sensor networks promises to be an interesting research field in the future.

Bibliography

[Aga06] A. Agah, M. Asadi, and S. K. Das. Prevention of DoS attack in sensor networks using repeated game theory. In *ICWN '06: Proceedings of the 2006 International Conference on Wireless Networks*, pp. 29–36. 2006.

[Ala06] A. Alarifi and W. Du. Diversify sensor nodes to improve resilience against node compromise. In *SASN '06: Proceedings of the fourth ACM workshop on Security of ad hoc and sensor networks*, pp. 101–112. ACM, New York, NY, USA, 2006.

[Alb02] P. Albers, O. Camp, J.-M. Percher, B. Jouga, L. Mé, and R. Puttini. Security in ad hoc networks: A general intrusion detection architecture enhancing trust based approaches. In *Proceedings of the First International Workshop on Wireless Information Systems (WIS-2002)*, pp. 1–12. April 2002.

[Ame98] American National Standards Institute. ANSI X9.62-1998, public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ECDSA), 1998.

[Ami08] S. O. Amin, M. S. Siddiqui, and C. S. Hong. Detecting jamming attacks in ubiquitous sensor networks. In *SAS '08: Proceedings of the IEEE Sensors Applications Symposium*, pp. 40–45. 2008.

[And96] R. Anderson and M. Kuhn. Tamper resistance: a cautionary note. In *WOEC'96: Proceedings of the 2nd conference on Proceedings of the Second USENIX Workshop on Electronic Commerce*, pp. 1–1. USENIX Association, Berkeley, CA, USA, 1996.

[Anj04] F. Anjum, D. Subhadrabandhu, S. Sarkar, and R. Shetty. On optimal placement of intrusion detection modules in sensor networks. In *BROADNETS '04: Proceedings of the First International Conference on Broadband Networks (BROADNETS'04)*, pp. 690–699. 2004.

[Aru04] M. Arumugam. Infuse: a TDMA based reprogramming service for sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys '04)*, pp. 281–282. 2004.

[Axe00] S. Axelsson. Intrusion detection systems: A survey and taxonomy. Tech. Rep. 99-15, Department of Computer Engineering, Chalmers University of Technology, March 2000.

[Bas01] S. Basagni, K. Herrin, D. Bruschi, and E. Rosti. Secure pebblenet. In *Proceedings of the 2001 ACM Iternational Symposium on Mobile Ad Hoc Networking & Computing, MobiHoc 2001*, pp. 156–163. Long Beach, CA, October 4–5 2001.

[Bec06] A. Becher, Z. Benenson, and M. Dornseif. Tampering with motes: Real-world physical attacks on wireless sensor networks. In *SPC '06: Proceeding of the 3rd International Conference on Security in Pervasive Computing*, vol. 3934 of *Lecture Notes in Computer Science*, pp. 104–118. Springer, 2006.

[Ben06] Z. Benenson, L. Pimenidis, E. Hammerschmidt, F. C. Freiling, and S. Lucks. Authenticated query flooding in sensor networks. In *Proceedings of the 21st IFIP International Information Security Conference (SEC 2006)*. May 2006.

[Beu03] J. Beutel, O. Kasten, and M. Ringwald. BTnodes – a distributed platform for sensor nodes. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pp. 292–293. ACM, New York, NY, USA, 2003.

[Bhu06] V. Bhuse and A. Gupta. Anomaly intrusion detection in wireless sensor networks. *Journal of High Speed Networks*, vol. 15(1):pp. 33–51, 2006.

[Bla06] E.-O. Blaß, J. Wilke, and M. Zitterbart. A security–energy trade-off for authentic aggregation in sensor networks. In *Proceedings of the Second IEEE Workshop on Wireless Mesh Networks (WiMesh 2006)*, pp. 135–137. IEEE Press, Washington D.C., USA, September 2006.

[Car00] D. W. Carman, P. S. Kruus, and B. J. Matt. Constraints and approaches for distributed sensor network security. Technical Report 00-010, NAI Labs, 2000.

[Cas05] C. Castelluccia, E. Mykletun, and G. Tsudik. Efficient aggregation of encrypted data in wireless sensor networks. In *MOBIQUITOUS '05: Proceedings of the The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pp. 109–117. IEEE Computer Society, Washington, DC, USA, 2005.

[Ç07] S. A. Çamtepe and B. Yener. Key distribution mechanisms for wireless sensor networks: a survey. Technical Report TR-05-07, Rensselaer Polytechnic Institute, 2007.

[Cha03a] H. Chan and A. Perrig. Security and privacy in sensor networks. *IEEE Computer*, vol. 36(10):pp. 103–105, 2003.

[Cha03b] H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pp. 197–213. IEEE Computer Society, Washington, DC, USA, 2003.

[Coo07] N. Cooprider, W. Archer, E. Eide, D. Gay, and J. Regehr. Efficient memory safety for TinyOS. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pp. 205–218. ACM, New York, NY, USA, 2007.

[Den04] J. Deng, R. Han, and S. Mishra. Intrusion tolerance and anti-traffic analysis strategies for wireless sensor networks. In *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks*, p. 637. IEEE Computer Society, Washington, DC, USA, 2004.

[Den06] J. Deng, R. Han, and S. Mishra. Secure code distribution in dynamically programmable wireless sensor networks. In *Proceedings of the fifth international conference on Information processing in sensor networks (IPSN '06)*, pp. 292–300. 2006.

[Dif76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, vol. IT-22(6):pp. 644–654, 1976.

[Dim04] T. Dimitriou, I. Krontiris, F. Nikakis, and P. G. Spirakis. SPEED: Scalable protocols for efficient event delivery in sensor networks. In N. Mitrou, K. P. Kontovasilis, G. N. Rouskas, I. Iliadis, and L. F. Merakos, editors, *NETWORKING*, vol. 3042 of *Lecture Notes in Computer Science*, pp. 1300–1305. Springer, 2004.

[Dim05a] T. Dimitriou and I. Krontiris. Autonomic communication security in sensor networks. In I. Stavrakakis and M. Smirnov, editors, *Autonomic Communication – WAC*, vol. 3854 of *Lecture Notes in Computer Science*, pp. 141–152. Springer, 2005.

[Dim05b] T. Dimitriou and I. Krontiris. A localized, distributed protocol for secure information exchange in sensor networks. In *Proceedings of the 5th IEEE International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN '05)*. April 2005.

[Dim06a] T. Dimitriou and I. Krontiris. GRAViTy: Geographic routing around voids. *International Journal of Pervasive Computing and Communications, Special Issue on Key Technologies and Applications of Wireless Sensor and Body-area Networks*, vol. 2(4):pp. 351–361, 2006.

[Dim06b] T. Dimitriou and I. Krontiris. Secure in-network processing in sensor networks. In Y. Xiao, editor, *Security in Sensor Networks*, chap. 12, pp. 275–290. CRC Press, 2006.

[Dim06c] T. Dimitriou, I. Krontiris, and F. Nikakis. Fast and scalable key establishment in sensor networks. In S. Phoha, T. F. L. Porta, and C. Griffin, editors, *Sensor Network Operations*, pp. 557–570. Wiley-IEEE Press, 2006.

[Du03] W. Du, J. Deng, Y. S. Han, and P. K. Varshney. A pairwise key predistribution scheme for wireless sensor networks. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pp. 42–51. ACM, New York, NY, USA, 2003.

[Du06] W. Du, L. Fang, and N. Peng. LAD: localization anomaly detection for wireless sensor networks. *J Parallel Distrib Comput*, vol. 66(7):pp. 874–886, 2006.

[Dut06] P. Dutta, J. Hui, D. Chu, and D. Culler. Securing the Deluge network programming system. In *Proceeding of the 5th International Conference on Information Processing in Sensor Networks (IPSN 2006)*, pp. 326–333. April 2006.

[Esc02] L. Eschenauer and V. D. Gligor. A key-management scheme for distributed sensor networks. In *CCS '02: Proceedings of the 9th ACM Conference on Computer and Communications Security*, pp. 41–47. ACM, New York, NY, USA, 2002.

[Gan01] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. In *MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pp. 251–254. ACM, New York, NY, USA, 2001.

[Gan05] S. Ganeriwal, S. Čapkun, C.-C. Han, and M. B. Srivastava. Secure time synchronization service for sensor networks. In *WiSe '05: Proceedings of the 4th ACM workshop on Wireless Security*, pp. 97–106. ACM, New York, NY, USA, 2005.

[Gär03] F. C. Gärtner. Byzantine failures and security: Arbitrary is not (always) random. Technical Report IC/2003/20, Swiss Federal Institute of Technology (EPFL), 2003.

[Gau04] G. Gaubatz, J.-P. Kaps, and B. Sunar. Public key cryptography in sensor networks—revisited. In *1st European Workshop on Security in Ad-Hoc and Sensor Networks (ESAS 2004)*, vol. 3313 of *Lecture Notes in Computer Science (LNCS)*, pp. 2–18. Springer, Heidelberg, August 2004.

[Gay03] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation (PLDI '03)*, pp. 1–11. ACM, New York, NY, USA, 2003.

[Gen01] R. Gennaro and P. Rohatgi. How to sign digital streams. *Information and Computation*, vol. 165(1):pp. 100–116, 2001.

[Gia08] T. Giannetsos, I. Krontiris, T. Dimitriou, and F. C. Freiling. Intrusion detection in wireless sensor networks. In Y. Zhang and P. Kitsos, editors, *Security in RFID and Sensor Networks*. CRC Press, Taylor&Francis Group, 2008.

[Goo07a] T. Goodspeed. Memory-constrained code injection. `http://travisgoodspeed.blogspot.com/`, September 2007.

[Goo07b] T. Goodspeed. Msp430 buffer overflow exploit for wireless sensor nodes. `http://travisgoodspeed.blogspot.com/`, August 2007.

[Gro06] J. Großschädl. TinySA: A security architecture for wireless sensor networks (extended abstract). In *Proceedings of the 2nd International Conference on Emerging Networking Experiments and Technologies (CoNEXT 2006)*. ACM Press, 2006.

[Gu08] Q. Gu and R. Noorani. Towards self-propagate mal-packets in sensor networks. In *WiSec '08: Proceedings of the first ACM conference on Wireless network security*, pp. 172–182. ACM, New York, NY, USA, 2008.

[Gup05] V. Gupta, M. Millard, S. Fung, Y. Zhu, N. Gura, H. Eberle, and S. C. Shantz. Sizzle: A standards-based end-to-end security architecture for the embedded internet. In *PERCOM '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, pp. 247–256. 2005.

[Gur04] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In *Cryptographic Hardware and Embedded Systems (CHES '04)*, pp. 119–132. August 2004.

[Ham06] A. Hamid, Mamun-Or-Rashid, and C. S. Hong. Routing security in sensor network: HELLO flood attack and defense. In *ICNEWS '06: Proceedings of the First International Conference on Next-Generation Wireless Systems*, pp. 77–81. 2006.

[Har05] C. Hartung, J. Balasalle, and R. Han. Node compromise in sensor networks: The need for secure systems. Technical Report CU-CS-990-05, Department of Computer Science, University of Colorado, 2005.

[Hil00] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. *ACM SIGPLAN Notices*, vol. 35(11):pp. 93–104, 2000.

[Hu05] Y.-C. Hu, A. Perrig, and D. B. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. *Wireless Networks*, vol. 11(1-2):pp. 21–38, 2005.

[Hui04] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pp. 81–94. 2004.

[Ilg95] K. Ilgun, R. A. Kemmerer, and P. A. Porras. State transition analysis: A rule-based intrusion detection approach. *Software Engineering*, vol. 21(3):pp. 181–199, 1995.

[Int03] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking*, vol. 11(1):pp. 2–16, 2003.

[Jav94] H. S. Javitz and A. Valdes. The NIDES statistical component: Description and justification. Annual report, Computer Science Laboratory, SRI International, Menlo Park, CA, March 1994.

[Kac02] O. Kachirski and R. Guha. Intrusion detection using mobile agents in wireless ad hoc networks. In *KMN '02: Proceedings of the IEEE Workshop on Knowledge Media Networking*, p. 153. 2002.

[Kac03] O. Kachirski and R. Guha. Effective intrusion detection using multiple sensors in wireless ad hoc networks. In *Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS '03)*, p. 57. January 2003.

[Kar03] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: Attacks and countermeasures. *AdHoc Networks Journal*, vol. 1(2–3):pp. 293–315, September 2003.

[Kar04] C. Karlof, N. Sastry, and D. Wagner. TinySec: A link layer security architecture for wireless sensor networks. In *Second ACM Conference on Embedded Networked Sensor Systems (SenSys 2004)*, pp. 162–175. November 2004.

[Ken98] S. Kent and R. Atkinson. Security architecture for the internet protocol. Internet Engineering Task Force RFC 2401, November 1998.

[Ko97] C. Ko, M. Ruschitzka, and K. Levitt. Execution monitoring of security-critical programs in distributed systems: A specification-based approach. In *SP '97: Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pp. 175–187. 1997.

[Ko01] C. Ko, P. Brutch, J. Rowe, G. Tsafnat, and K. N. Levitt. System health and intrusion monitoring using a hierarchy of constraints. In *RAID '00: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, pp. 190–204. 2001.

[Kra01] H. Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pp. 310–331. Springer-Verlag, London, UK, 2001.

[Kri02] B. Krishnamachari, D. Estrin, and S. B. Wicker. The impact of data aggregation in wireless sensor networks. In *ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*, pp. 575–578. IEEE Computer Society, Washington, DC, USA, 2002.

[Kro06a] I. Krontiris and T. Dimitriou. Authenticated in-network programming for wireless sensor networks. In T. Kunz and S. S. Ravi, editors, *AD-HOC Networks & Wireless – ADHOC-NOW*, vol. 4104 of *Lecture Notes in Computer Science*, pp. 390–403. Springer, 2006.

[Kro06b] I. Krontiris and T. Dimitriou. A practical authentication scheme for in-network programming in wireless sensor networks. In *Proceedings of the 2nd ACM Workshop on Real-World Wireless Sensor Networks (REALWSN '06)*, pp. 13–17. Uppsala, Sweden, June 2006.

[Kro07a] I. Krontiris and T. Dimitriou. Secure network programming in wireless sensor networks. In Y. Xiao and Y. Pan, editors, *Security in Distributed and Networking Systems*, chap. 12, pp. 289–310. World Scientific Publishing Co., 2007.

[Kro07b] I. Krontiris, T. Dimitriou, and F. C. Freiling. Towards intrusion detection in wireless sensor networks. In *Proceedings of the 13th European Wireless Conference*. Paris, France, April 2007.

[Kro07c] I. Krontiris, T. Dimitriou, T. Giannetsos, and M. Mpasoukos. Intrusion detection of sinkhole attacks in wireless sensor networks. In M. Kutylowski, J. Cichon, and P. Kubiak, editors, *Algorithmic Aspects of Wireless Sensor Networks – ALGOSENSORS*, vol. 4837 of *Lecture Notes in Computer Science*, pp. 150–161. Springer, 2007.

[Kro08a] I. Krontiris and T. Dimitriou. Launching a sinkhole attack in wireless sensor networks; the intruder side. In *First International Workshop on Security and Privacy in Wireless and Mobile Computing, Networking and Communications*. Avignon, France, October 2008.

[Kro08b] I. Krontiris and T. Dimitriou. Security issues in biomedical sensor networks. In *First International Symposium on Applied Sciences*

in Bio-Medical and Communication Technologies (ISABEL '08). Aalborg, Denmark, October 2008.

[Kro08c] I. Krontiris, T. Dimitriou, and T. Giannetsos. LIDeA: A distributed lightweight intrusion detection architecture for sensor networks. In *Proceeding of the 4th International Conference on Security and Privacy for Communication (SECURECOMM '08)*. Istanbul, Turkey, September 2008.

[Kro08d] I. Krontiris, T. Dimitriou, H. Soroush, and M. Salajegheh. WSN link-layer security frameworks. In J. Lopez and J. Zhou, editors, *Wireless Sensors Networks Security*, chap. 6, pp. 142–163. IOS Press, 2008.

[Kul02] J. Kulik, W. Heinzelman, and H. Balakrishnan. Negotiation-based protocols for disseminating information in wireless sensor networks. *Wireless Networks*, vol. 8(2/3):pp. 169–185, 2002.

[Kul05] S. S. Kulkarni and L. Wang. MNP: Multihop network reprogramming service for sensor networks. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pp. 7–16. 2005.

[Kum07] R. Kumar, E. Kohler, and M. Srivastava. Harbor: Software-based memory protection for sensor nodes. In *IPSN '07: Proceedings of the 6th International Symposium on Information Processing in Sensor Networks*, pp. 340–349. 2007.

[Lan06] P. E. Lanigan, R. Gandhi, and P. Narasimhan. Sluice: Secure dissemination of code updates in sensor networks. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS '06)*, p. 53. 2006.

[Laz03] L. Lazos and R. Poovendran. Energy-aware secure multicast communication in ad-hoc networks using geographic location information. *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'03)*, vol. 4, 2003.

[Laz05] L. Lazos and R. Poovendran. SeRLoc: Robust localization for wireless sensor networks. *ACM Transactions on Sensor Networks*, vol. 1(1):pp. 73–100, 2005.

[Li05] T. Li, H. Wu, X. Wang, and F. Bao. SenSec: Sensor security framework for TinyOS. In *INSS '05: Proceedings of the Second International Workshop on Networked Sensing Systems*. 2005.

[Lin99] U. Lindqvist and P. A. Porras. Detecting computer and network misuse through the production-based expert system toolset (p-BEST). In *IEEE Symposium on Security and Privacy*, pp. 146–161. 1999.

[Liu03] D. Liu and P. Ning. Location-based pairwise key establishments for static sensor networks. In *SASN '03: Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, pp. 72–82. ACM, New York, NY, USA, 2003.

[Liu05] D. Liu, P. Ning, and R. Li. Establishing pairwise keys in distributed sensor networks. *ACM Transactions on Information and System Security*, vol. 8(1):pp. 41–77, 2005.

[Liu08a] A. Liu and P. Ning. TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks. *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN 2008)*, vol. 0:pp. 245–256, 2008.

[Liu08b] A. Liu, Y.-H. Oh, and P. Ning. Secure and DoS-resistant code dissemination in wireless sensor networks using Seluge. *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN 2008)*, vol. 0:pp. 561–562, 2008.

[Loo05] C. E. Loo, M. Y. Ng, C. Leckie, and M. Palaniswami. Intrusion detection for routing attacks in sensor networks. *International Journal of Distributed Sensor Networks*, 2005.

[Lop06] J. Lopez. Unleashing public-key cryptography in wireless sensor networks. *Journal of Computer Security*, vol. 14(5):pp. 469–482, 2006.

[Luk07] M. Luk, G. Mezzour, A. Perrig, and V. Gligor. MiniSec: A secure sensor network communication architecture. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pp. 479–488. ACM, New York, NY, USA, 2007.

[Mad02] S. Madden, R. Szewczyk, M. J. Franklin, and D. Culler. Supporting aggregate queries over ad-hoc wireless sensor networks. In *WMCSA '02: Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, pp. 49–58. IEEE Computer Society, Washington, DC, USA, 2002.

[Mar00] S. Marti, T. J. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of the 6th International Conference on Mobile Computing and Networking (MobiCom '00)*, pp. 255–265. August 2000.

[Nai07] V. Naik, A. Arora, P. Sinha, and H. Zhang. Sprinkler: A reliable and energy efficient data dissemination service for extreme scale wireless networks of embedded devices. *IEEE Transactions on Mobile Computing*, vol. 6(7):pp. 777–789, July 2007.

[New04] J. Newsome, E. Shi, D. Song, and A. Perrig. The Sybil attack in sensor networks: Analysis & defenses. In *IPSN '04: Proceedings of*

the third international symposium on Information processing in sensor networks, pp. 259–268. ACM, New York, NY, USA, 2004.

[Ona05] I. Onat and A. Miri. An intrusion detection system for wireless sensor networks. In *Proceeding of the IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, vol. 3, pp. 253–259. Montreal, Canada, August 2005.

[Par05] T. Park and K. G. Shin. Soft tamper-proofing via program integrity verification in wireless sensor networks. *IEEE Transactions on Mobile Computing*, vol. 4(3):pp. 297–309, 2005.

[Par06] B. Parno, M. Luk, E. Gaustad, and A. Perrig. Secure sensor network routing: A clean-slate approach. In *Proceedings of Conference on Future Networking Technologies (CoNEXT)*. December 2006.

[Pea80] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, vol. 27(2):pp. 228–234, 1980.

[Per02] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler. SPINS: security protocols for sensor networks. *Wireless Networks*, vol. 8(5):pp. 521–534, 2002.

[Pic82] R. L. Pickholtz, D. L. Schilling, and L. B. Milstein. Theory of spread-spectrum communications – a tutorial. *IEEE Transactions on Communications*, vol. 30(5):pp. 855–884, May 1982.

[Pie03] J. Pieprzyk, H. Wang, and C. Xing. Multiple-time signature schemes against adaptive chosen message attacks. In *Selected Areas in Cryptography (SAC 2003)*, pp. 88–100. Springer, 2003.

[Pla08] E. Platon and Y. Sei. Security software engineering in wireless sensor networks. *Progress in Informatics*, vol. 5:pp. 49–64, 2008.

[Prz03] B. Przydatek, D. Song, and A. Perrig. SIA: secure information aggregation in sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pp. 255–265. 2003.

[Rey02] L. Reyzin and N. Reyzin. Better than BiBa: Short one-time signatures with fast signing and verifying. In *Proceedings of the 7th Australian Conference on Information Security and Privacy (ACISP '02)*, pp. 144–153. Springer-Verlag, London, UK, 2002.

[Riv78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, vol. 21(2):pp. 120–126, 1978.

[Rom06] R. Roman, J. Zhou, and J. Lopez. Applying intrusion detection systems to wireless sensor networks. In *Proceedings of IEEE Consumer Communications and Networking Conference (CCNC '06)*, pp. 640–644. Las Vegas, USA, January 2006.

[Sal06] M. Salajegheh, H. Soroush, A. Thomos, T. Dimitriou, and I. Krontiris. .Sense, a secure framework for sensor network data acquisition, monitoring and command. In *Proceedings of the 2nd ACM Workshop on Real-World Wireless Sensor Networks (REALWSN '06)*, pp. 101–102. Uppsala, Sweden, June 2006.

[Sas03] N. Sastry, U. Shankar, and D. Wagner. Secure verification of location claims. In *WiSe '03: Proceedings of the 2nd ACM Workshop on Wireless Security*, pp. 1–10. ACM, New York, NY, USA, 2003.

[Ses04] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. SWATT: Software-based attestation for embedded devices. *Symposium on Security and Privacy*, pp. 272–282, 2004.

[Ses06] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla. SCUBA: Secure code update by attestation in sensor networks. In *WiSe '06: Proceedings of the 5th ACM workshop on Wireless Security*, pp. 85–94. ACM, New York, NY, USA, 2006.

[Sha05] M. Shaneck, K. Mahadevan, V. Kher, and Y. Kim. Remote software-based attestation for wireless sensors. In *ESAS '05: Proceedings of the European Workshop on Security in Ad-hoc and Sensor Networks*, vol. 3813 of *Lecture Notes in Computer Science*, pp. 27–41. Springer, 2005.

[Sha07] J. Shaheen, D. Ostry, V. Sivaraman, and S. Jha. Confidential and secure broadcast in wireless sensor networks. In *IEEE Personal, Indoor, and Mobile Radio Communications (PIMRC '07)*. Athens, Greece, September 2007.

[Sil05] A. P. da Silva, M. Martins, B. Rocha, A. Loureiro, L. Ruiz, and H. C. Wong. Decentralized intrusion detection in wireless sensor networks. In *Proceedings of the 1st ACM International Workshop on Quality of Service & Security in Wireless and Mobile Networks (Q2SWinet '05)*, pp. 16–23. ACM Press, October 2005.

[Sir01] A. Siraj, S. Bridges, and R. Vaughn. Fuzzy cognitive maps for decision support in an intelligent intrusion detection system. In *IFSA World Congress and 20th North American Fuzzy Information Processing Society (NAFIPS) International Conference*, vol. 4, pp. 2165–2170. July 2001.

[Sli02] S. Slijepcevic, M. Potkonjak, V. Tsiatsis, S. Zimbeck, and M. B. Srivastava. On communication security in wireless ad-hoc sensor networks.

In *WETICE '02: Proceedings of the 11th IEEE International Workshops on Enabling Technologies*, pp. 139–144. IEEE Computer Society, Washington, DC, USA, 2002.

[Sor07] H. Soroush, M. Salajegheh, and T. Dimitriou. Providing transparent security services to sensor networks. In *ICC'07: Proceedings of the IEEE International Conference on Communications*. Glasgow, Scotland, June 2007.

[ssl01] OpenSSL. `http://www.openssl.org`, 2001.

[Sta02] F. Stajano. *Security for Ubiquitous Computing*. John Wiley and Sons, February 2002.

[Sta03] T. Stathopoulos, J. Heidemann, and D. Estrin. A remote code update mechanism for wireless sensor networks. Tech. Rep. CENS-TR-30, University of California, Los Angeles, Center for Embedded Networked Computing, November 2003.

[Sun06] K. Sun, P. Ning, and C. Wang. TinySeRSync: Secure and resilient time synchronization in wireless sensor networks. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pp. 264–277. ACM, New York, NY, USA, 2006.

[Szc08] P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab. NanoECC: Testing the limits of elliptic curve cryptography in sensor networks. In *Proceedings of the 5th European conference on Wireless Sensor Networks (EWSN)*, vol. 4913 of *Lecture Notes in Computer Science*, pp. 305–320. Springer, 2008.

[Uhs07] L. Uhsadel, A. Poschmann, and C. Paar. Enabling Full-Size Public-Key Algorithms on 8-bit Sensor Nodes. In *Proceedings of European Workshop on Security in Ad-Hoc and Sensor Networks (ESAS 2007)*, vol. 4572 of *LNCS*, pp. 73–86. Springer-Verlag, 2007.

[Ulm00] C. Ulmer. Wireless sensor probe networks - SensorSimII. `http://www.craigulmer.com/research/sensorsimii/`, 2000.

[Č06] S. Čapkun and J.-P. Hubaux. Secure positioning in wireless networks. *IEEE Journal on Selected Areas in Communications*, vol. 24(2):pp. 221–232, 2006.

[Wes06] D. Westhoff, J. Girao, and M. Acharya. Concealed data aggregation for reverse multicast traffic in sensor networks: Encryption, key distribution, and routing adaptation. *IEEE Transactions on Mobile Computing*, vol. 5(10):pp. 1417–1431, 2006.

[Woo02] A. D. Wood and J. A. Stankovic. Denial of service in sensor networks. *Computer*, vol. 35(10):pp. 54–62, 2002.

[Woo03a] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pp. 14–27. 2003.

[Woo03b] A. D. Wood, J. A. Stankovic, and S. H. Son. JAM: A jammed-area mapping service for sensor networks. In *RTSS '03: Proceedings of the 24th IEEE International Real-Time Systems Symposium*, p. 286. IEEE Computer Society, Washington, DC, USA, 2003.

[Xu03] Y. Xu. Energy-aware object tracking sensor networks. In *ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing System - Doctoral Symposium*. 2003.

[Xue03] Q. Xue and A. Ganz. Runtime security composition for sensor networks (SecureSense). In *IEEE Vehicular Technology Conference (VTC Fall 2003)*. October 2003.

[Yan06] Y. Yang, X. Wang, S. Zhu, and G. Cao. SDAP: A secure hop-by-hop data aggregation protocol for sensor networks. In *MobiHoc '06: Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, pp. 356–367. ACM, New York, NY, USA, 2006.

[Yan07] Y. Yang, X. Wang, S. Zhu, and G. Cao. Distributed software-based attestation for node compromise detection in sensor networks. In *SRDS '07: Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems*, pp. 219–230. IEEE Computer Society, Washington, DC, USA, 2007.

[Yan08] Y. Yang, S. Zhu, and G. Cao. Improving sensor network immunity under worm attacks: A software diversity approach. In *MobiHoc '08: Proceedings of the 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing*. 2008.

[Ylo96] T. Ylonen. SSH: Secure login connections over the Internet. In *SSYM'96: Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography*, pp. 4–4. USENIX Association, Berkeley, CA, USA, 1996.

[Zha03] Y. Zhang, W. Lee, and Y.-A. Huang. Intrusion detection techniques for mobile wireless networks. *Wireless Networks*, vol. 9(5):pp. 545–556, 2003.

[Zha05] Q. Zhang, P. Wang, D. S. Reeves, and P. Ning. Defending against sybil attacks in sensor networks. In *ICDCSW '05: Proceedings of the Second International Workshop on Security in Distributed Computing Systems*, pp. 185–191. IEEE Computer Society, Washington, DC, USA, 2005.

[Zhu03] S. Zhu, S. Setia, and S. Jajodia. LEAP: efficient security mechanisms for large-scale distributed sensor networks. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pp. 62–72. ACM, New York, NY, USA, 2003.