Vulnerabilities and Attacks in Wireless Sensor Networks

Zinaida BENENSON [a,1], Peter M. CHOLEWINSKI [b], Felix C. FREILING [a]
[a] *Laboratory for Dependable Distributed Systems, University of Mannheim, 68131 Mannheim, Germany*
[b] *SAP - Research and Breakthrough Innovation, Germany*

Abstract. We investigate how wireless sensor networks can be attacked in practice. From this we develop a generic adversary model that allows to classify adversaries according to two dimensions of power: presence and intervention. Thus, we provide a framework for realistic security analysis in wireless sensor networks.

Keywords. adversary models, attacks, threat analysis, security evaluation

1. Introduction

Sensor networks provide unique opportunities of interaction between computer systems and their environment. Their deployment can be described at high level as follows: The sensor nodes measure environmental characteristics which are then processed in order to detect events. Upon event detection, some actions are triggered. This very general description applies to extremely security-critical military applications as well as to such benign ones (in terms of security needs) as habitat monitoring.

Considering the Internet as an example, it is extremely difficult to add security to systems which were originally designed without security in mind. The goal of security is to "protect right things in a right way" [1]. Thus, careful analysis is needed concerning which things to protect against which threats and how to protect them. Of course, this analysis is only possible in context of a particular class of applications. However, it makes much sense to provide a set of abstract security requirements and a set of generic attacker models, i.e., a *framework for security analysis in wireless sensor networks*, which can be refined for particular applications.

In this chapter, we present such a framework. It provides concepts to clarify two important aspects of the security analysis in wireless sensor networks:

1. What should be protected? Here we offer a set of generic classes of requirements which can be used to structure and refine a set of concrete security

[1] Zinaida Benenson was supported by Landesstiftung Baden Württemberg as part of Project "ZeuS – Zuverlässige Informationsbereitstellung in energiebewussten ubiquitären Systemen".

requirements. We highlight the main differences between security requirements in classical systems and security requirements in wireless sensor networks.
2. Against what are we protecting the system? Here we offer a set of generic attacker models which can be used to choose and refine particular attacker models for individual systems.

Overall, attacker models in conjunction with security requirements determine the means to achieve security.

In practice it is very important to formulate *realistic* security requirements and *realistic* attacker models. Such choices guarantee that precious resources of wireless sensor networks are invested efficiently. It is therefore useful to evaluate the practicality of certain attacker models. One metric to measure practicality is to evaluate the *effort* an attacker has to invest to perform certain attacks. We contribute to this area by reporting on a number of experiments in which we attacked *real* sensor node hardware.

This chapter is structured as follows: We first give an overview of security goals in sensor networks, i.e., we approach the question "what to protect" (Section 2). We then report on experiments in attacking wireless sensor networks in Section 3. Building on these experiences we develop a generic set of attacker models in Section 4), i.e., we approach the question "against whom to protect". Finally, we briefly discuss protection mechanisms in Section 5, i.e., we approach the question "how to protect". We outline open problems and summarize in Section 6.

2. Security Goals in Sensor Networks

A sensor network can be considered as a highly distributed database. Security goals for distributed databases are very well studied: The data should be accessible only to authorized users (Confidentiality), the data should be genuine (Integrity), and the data should be always available on the request of an authorized user (Availability). All these requirements also apply to sensor networks and their users. Here, the distributed database, as well as the sensor network, are considered as a single entity from the user's point of view. Therefore, we call these security issues *outside security*. To outside security belong, e.g., query processing [23, 36, 47], access control [9] and large-scale anti-jamming services [48].

The internal organization of a distributed database and of a sensor network are quite different. Outside security, as well as all other types of interactions between the user and the corresponding system, is based on the interactions between the internal system components (servers or sensor nodes, respectively). We call security issues for such interactions *inside security*. In sensor networks, inside security realizes robust, confidential and authenticated communication between individual nodes [25, 45]. This also includes in-network processing [18, 49], data aggregation [11, 50], routing [17, 26] and in-network data storage [7, 20].

Aside from necessitating the distinction between inside and outside security, sensor networks differ from conventional distributed databases in other obvious ways: A distributed database consists of a small number of powerful servers, which are well protected from physical capture and from network attacks. The servers

use resource-demanding data replication and cryptographic protocols for inside and outside security. In contrast, a sensor network consists of a large number of resource-constrained, physically unprotected sensor nodes which operate unattended. Therefore, security measures for distributed databases cannot be directly applied to sensor networks. So even if a sensor network can be considered as a distributed system (e.g., as an ad hoc network), sensor networks have some additional constraints which make security mechanisms for distributed systems inapplicable. Apart from the obvious resource constraints, single sensor nodes are relatively unimportant for the properties of the whole system – at least, if the inherent redundancy of sensor networks is utilized in their design.

To summarize, security goals in sensor networks are similar to security goals in distributed databases (outside security) and distributed systems (inside security). So these can be taken as an orientation. While requirements are similar, many standard mechanisms to *implement* security (e.g., public key infrastructures or agreement protocols) are not applicable because they require too many resources or do not scale to hundreds or thousands of nodes. This is the dilemma of sensor networks and forces security mechanisms in wireless sensor networks to spend the "right" amount of effort in the "right" places by exploiting the natural features of sensor networks: inherent redundancy and broadcast communication.

In the remainder of this chapter, we will take a first step towards developing realistic security mechanisms. We evaluate real attacks in practice and from this evaluation derive a fine-grained set of abstract attacker assumptions.

3. Attacking Wireless Sensor Networks

In this section we take the viewpoint of an adversary who wishes to violate one of the security requirements of a WSN which were described in Section **??**. From this viewpoint, a WSN is a very interesting target because if offers a large attack surface and an interesting playground for creative attack ideas.

Of course, the many possibilties to attack WSNs include all the classical techniques known from classical system security. An adversary can eavesdrop on the communication, perform traffic analysis of the observed network behavior, he can replay old messages or inject false messages into the network. Possible are also other types of attacks that aim violating Availability (denial-of-service attacks) like jamming the wireless channel. In WSNs these attacks can be particularly important as they can cause rapid battery draining and effectively disable individual sensor nodes or entire parts of a WSN.

While there are many techniques known from other areas of security, the ability of an attacker to access (and eventually change) the internal state of a sensor node seems particularly characteristic for sensor networks. This type of attack is called *node capture* in the literature [19, 35]. Depending on the WSN architecture, node capture attacks can have significant impact. Thus, most existing routing schemes for WSNs can be substantially influenced even through capture of a minute portion of the network [26]. In the TinySec mechanism [25], which enables secure and authenticated communication between the sensor nodes by means of a network-wide shared master key, capture of a single sensor node suffices to give

the adversary unrestricted access to the WSN. Most current security mechanisms for WSNs take node capture into account. It is usually assumed that node capture is "easy". Thus, some security mechanisms are verified with respect to being able to resist capture of 100 and more sensor nodes out of 10,000 [12, 24].

In this section, we determine the actual cost and effort needed to attack currently available sensor nodes. We especially concentrate on *physical attacks* which require direct physical access to the sensor node hardware. As sensor nodes operate unattended and cannot be made tamper proof because they should be as cheap as possible, this scenario is more likely than in most other computing environments. Physical attacks of some form are usually considered a prerequisite to perform node capture.

Another possibility to gain access to the internal state of a sensor node is to exploit some bugs in software running on the sensor nodes or on the base stations. These attacks directly correspond to well-known techniques from classical software security. If a software vulnerability is identified by the attacker, an attack can be easily automated and can be mounted on a very large number of nodes in a very short amount of time. Such attacks are relatively well-understood in software security [22]: Possible countermeasures include a heterogenous network design and standard methods from software engineering [21, 22, 28, 34, 44]. As these attacks are not characteristic for WSNs, we do not consider such attacks in detail. This area is however an interesting direction for future work.

In the following, we first give some background on physical attacks on sensor node hardware. Then we report on the effort needed to attack some current sensor nodes. Where appropriate we also discuss countermeasures that increase the effort to mount a successful attack. Overall, this section gives insight into *practical* attacks on WSNs which motivate the abstract attacker models which follow in Section 4.

3.1. Background on Physical Attacks on Sensor Nodes

3.1.1. Tampering vs. Physical Attacks

Tampering attacks on embedded systems, that is, on microcontrollers and smart cards, have been intensively studied, see e.g. 1, 3, 38. The term "tampering" is well accepted in the research community to designate attacks on components that involve modification of the internal structure of a single chip. At the same time, there are also conceivable attacks on sensor node hardware which are carried out on the circuit board level and therefore, do not really fit this accepted usage. In order to avoid this terminology problem, we use the term "physical attack" to refer to all attacks requiring direct physical access to the sensor node.

Skorobogatov [37] describes in depth tampering attacks on microcontrollers, and classifies them in the three categories of *invasive*, *semi-invasive*, and *non-invasive* attacks. Invasive attacks are those which require access to a chip's internals, and they typically need expensive equipment used in semiconductor manufacturing and testing, as well as a preparation of the chip before the attack can begin. Semi-invasive attacks require much cheaper equipment and less time than the invasive attacks, while non-invasive attacks are the easiest.

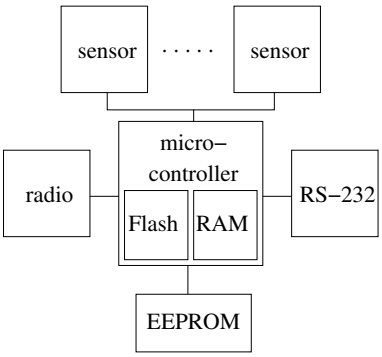

Figure 1. General schematic view of sensor node hardware.

All of these attacks, including the so-called low-cost attacks, if applied to sensor nodes, would require that the nodes be removed from the deployment area and taken to a laboratory. Even if in some cases, the laboratory could be moved into the deployment area in a vehicle, all attacks would require at least disruption of the regular node operation. Most of the invasive and many of the semi-invasive attacks also require disassembly or physical destruction of the sensor nodes.

The existing literature on physical attacks usually assumes that an attacker can gain unsupervised access to the system to be attacked for an extended period of time. This is a sensible assumption for systems such as pay-per-view TV cards, pre-paid electricity tokens, or GSM SIM cards. Attacks which may take days or even weeks to complete present a real threat to the security of these systems.

In wireless sensor networks, however, regular communication with neighboring nodes is usually part of normal network operation. Continuous absence of a node can therefore be considered an unusual condition that can be noticed by its neighbors. This makes time a very important factor in evaluating attacks against sensor nodes, as the system might be able to detect such attacks while they are in progress and respond to them in real-time. If the amount of time needed to carry out various attacks is known, the frequency with which neighbors should be checked can be adapted to the security goals and the anticipated attacker model.

3.1.2. Current Sensor Node Hardware

Currently available sensor nodes typically consist of embedded hardware with low power consumption, and low computing power. A typical sensor node contains some sensors (light, temperature, acceleration etc.), a radio chipset for wireless communication, an EEPROM chip for logging sensor data, a node-to-host communication interface (typically a serial port), and a microcontroller which contains some amount of flash memory for program storage and RAM for program execution. Power is provided by batteries.

Typical choices for the microcontroller are the 8 bit Atmel ATmega 128 or the 16 bit Texas Instruments MSP430 family, with the amount of RAM varying between 2 kB and 10 kB and flash memory ranging from 48 kB to 128 kB. External EEPROM memory can be as small as 8 kB or as large as 1 MB. The speed of radio communications is in the order of 100 $^{\text{kbit}}/_{\text{s}}$.

Figure 2. Current sensor node hardware: Mica 2 by Crossbow, Berkeley [27]; Tmote sky by moteiv, Berkeley [32]; and Embedded Sensor Board by ScatterWeb, Berlin [10]

The most interesting part for an attacker will be the microcontroller, as control over this component means complete control over the operation of the node. However, the other parts might be of interest as well in certain attack scenario.

Figure 1 shows a general schematic view of the hardware of current sensor nodes, while Figure 2 shows photographs of some concrete models available today. The Crossbow Mica2 nodes [15, 16] (Fig. 2, left) use the 8 bit Atmel ATmega 128 microcontroller [5] with 4 kB RAM and 128 kB integrated flash memory, the Atmel AT45DB041B 4 Mbit flash memory chip [4], and the Chipcon CC1000 radio communications chipset [13] with a maximum data rate of 76.8 $^{\text{kbit}}/_{\text{s}}$. Programming is done via the Atmel's serial programming interface by placing the node in a special interface board and connecting it to an RS-232 serial port on the host.

The Telos motes [31, 32] (Fig. 2, center) by Moteiv utilize the Texas Instruments MSP430 F1611 microcontroller [42, 43], providing 10 kB of RAM and 48 kB flash memory. The EEPROM used is the 8 Mbit ST Microelectronics M25P80 [39], and the radio chipset is the Chipcon CC2420 [14], whose maximum data rate is 250 $^{\text{kbit}}/_{\text{s}}$. Programming is performed by connecting to the USB interface and writing memory with the help of the MSP430 bootloader [40]. A JTAG interface is available as an alternative programming method and can also be used for debugging.

The Embedded Sensor Boards [10] (Fig. 2, right) from ScatterWeb GmbH are built around the Texas Instruments MSP430 F149 microcontroller [41, 43] featuring 2 kB of RAM and 60 kB flash memory, the Microchip 24LC64 [29] 64 kbit EEPROM, and the RFM TR1001 radio chipset [30] with a maximum data rate of 19.2 $^{\text{kbit}}/_{\text{s}}$. Programming is done either through a JTAG interface or over-the-air using a gateway.

3.1.3. Possibilities for Physical Attacks

Concrete physical attacks on sensor nodes can be classified in several ways. Our classification takes our previous considerations into account that sensor nodes are more or less in permanent contact with each other. This means that long interruptions of regular operation can be noticed and acted upon. Therefore, attacks which result in a long interruption (e. g. because the node has to be physically removed from the network and taken to a distant laboratory) are not as dangerous as those which can be carried out *in the field*. Therefore, in the following we concentrate on this type of attacks as well as on possible countermeasures to be employed by a sensor network.

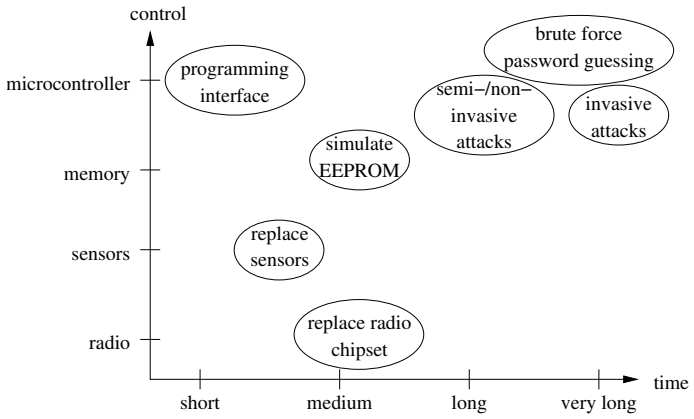

Figure 3. Design space for physical attacks on sensor nodes.

The two main categories that we use for classifying physical attacks are (1) the degree of control over the sensor node the attacker gains; and (2) the time span during which regular operation of a node is interrupted [6]. Figure 3 illustrates this design space and classifies example attacks from the forthcoming Section 3.2 according to its criteria.

3.2. Examples of In-the-Field Attacks and Countermeasures

As explained above, we especially consider attacks which can be mounted without noticeable interruption of the regular sensor node operation. We now discuss some attacks in detail.

3.2.1. Attacks via JTAG

The IEEE 1149.1 JTAG standard is designed to assist electronics engineers in testing their equipment during the development phase. Among other things, it can be used in current equipment for on-chip debugging, including single-stepping through code, and for reading and writing memory.

A JTAG Test Access Port (TAP) is present on both the Atmel and Texas Instruments microcontrollers used on the sensor nodes described above. All sensor nodes examined by us have a JTAG connector on their circuit board allowing easy access to the microcontroller's TAP. While the capabilities offered by JTAG are convenient for the application developer, it is clear that an attacker must not be provided with the same possibilities. Therefore it is necessary to disable access to the microcontroller's internals via JTAG before fielding the finished product.

The MSP430 has a security fuse which can be irreversibly blown (as described in the data sheet) to disable the entire JTAG test circuitry in the microcontroller. Further access to the MSP430's memory is then only possible by using the Bootstrap Loader. The ATmega128 requires the programmer to set the appropriate fuses and lock bits, which effectively disable all memory access via JTAG or any other interface from the outside.

If JTAG access is left enabled, an attacker equipped with an appropriate adapter cable and a portable computer is capable of taking complete control over

the sensor node. Even if there is no JTAG connector provided on the circuit board, attackers can still get access to the JTAG ports by directly connecting to the right pins on the microcontroller which can be looked up in the datasheet. Typical data rates for JTAG access are 1–2 $^{\mathrm{kB}}/_{\mathrm{s}}$, so reading or writing 64 kB of data takes between 30 and 70 s. However, there are specialized programming devices on the market which can attain much higher data rates. One such device claims to be able to program 60 kB of memory in a mere 3.4 s.

3.2.2. Attacks via the Bootstrap Loader

On the Telos nodes, the canonical way of programming the microcontroller is by talking to the Texas Instruments specific bootstrap loader (BSL) through the USB interface. The bootstrap loader [40] is a piece of software contained in the ROM of the MSP430 series of microcontrollers that enables reading and writing the microcontroller's memory independently of both the JTAG access and the program currently stored on the microcontroller.

The BSL requires the user to transmit a password before carrying out any interesting operation. Without this password, the allowed operations are essentially "transmit password" and "mass erase", i.e. erasing all memory on the microcontroller.

The BSL password has a size of $16 \cdot 16$ bit and consists of the flash memory content at addresses 0xFFE0 to 0xFFFF. This means in particular that, immediately after a mass erase operation, the password consists of 32 bytes containing the value 0xFF. The memory area used for the password is the same that is used for the interrupt vector table, i.e. the BSL password is actually identical to the interrupt vector table. The interrupt vector table, however, is usually determined by the compiler and not by the user, although Texas Instruments documents describe the password as user-settable.

Finding out the password may be quite time-consuming for an attacker. However, such an investment of time may be justified if a network of nodes all having an identical password is to be attacked. Therefore, an evaluation of the possibility to guess the password follows.

Brute Force. As the password is composed of interrupt vectors, certain restrictions apply to the values of the individual bytes. This section examines the expected size of the key space and estimates the expected duration of a brute force attack on the password.

Initially, the key space has a size of $16 \cdot 16\,\mathrm{bit} = 256\,\mathrm{bit}$. Assuming a typical compiler (mspgcc 3.2 [33] was tested), the following restrictions apply:

- All code addresses must be aligned on a 16 bit word boundary, so the least significant bit of every interrupt vector is 0. This leaves us with a key space of $16 \cdot 15\,\mathrm{bit} = 240\,\mathrm{bit}$.
- The reset vector, which is one of the interrupt vectors, is fixed and points to the start of the flash memory, reducing the key space to $15 \cdot 15\,\mathrm{bit} = 225\,\mathrm{bit}$.
- Interrupt vectors which are not used by the program are initialized to the same fixed address, containing simply the instruction `reti` (return from interrupt). As a worst case assumption, even the most basic program will

still use at least four interrupts, and therefore have a key space of at least $4 \cdot 15\,\text{bit} = 60\,\text{bit}$.

- Code is placed by the compiler in a contiguous area of memory starting at the lowest flash memory address. Under the assumption that the program is very small and uses only $2\,\text{kB} = 2^{11}\,\text{B}$ of memory, we are left with a key space of a mere $4 \cdot 10\,\text{bit} = 40\,\text{bit}$.

We conclude that the size of the key space for every BSL password is at least 40 bit.

A possible brute force attack can be performed by connecting a computer to the serial port (the USB port, in the case of Telos nodes) and consecutively trying passwords. This can be done by executing a modified version of the `msp430-bsl` [33] program that is normally used for communicating with the BSL.

The rate at which passwords can be guessed was measured to be approximately 12 passwords per second when the serial port was set to 9600 baud. However, the MSP430 F1611 used on the Telos nodes is capable of a line speed of 38,400 baud, and at this speed, approximately 31 passwords can be tried per second. Finally, the BSL program normally waits for an acknowledgment from the microcontroller after each message sent over the serial line. If this wait is not performed, the speed of password guessing rises to 83 passwords per second.

The maximum speed of password guessing in practice can therefore be assumed to be 2^7 passwords per second. This is quite close to the theoretical limit of $38,400\,{}^{\text{bit}}/_{\text{s}} \cdot (256\,{}^{\text{bit}}/_{\text{pw}})^{-1} = 150\,{}^{\text{pw}}/_{\text{s}}$.

Recalling that the key space has a size of at least 40 bit, we can now conclude that a brute force attack can be expected to succeed on the average after $2^{40-7-1}\,\text{s} = 2^{32}\,\text{s} \approx 128\,\text{a}$. As 128 years is well beyond the expected life time of current sensor nodes, a brute force attack can be assumed to be impractical.

Knowledge of the Program. One consequence of the fact that the password is equal to the interrupt vector table is that anyone in possession of an object file of the program stored on a sensor node also possesses the password. Worse, even someone who only has the source code of the program still can get the password if he has the same compiler as the developer, since he can use this compiler to produce an image from the source code identical to the one on the deployed nodes.

The secret key in the current TinySec implementation, for example, is contained in the image but does not influence the interrupt vector table. If TinySec were ported to Telos motes, the source code and the compiler used would be sufficient information for an attacker to extract the secret key material. The same holds for any kind of cryptographic mechanism where the key material does not influence the interrupt vectors.

Another way of exploiting the identity of the password and the interrupt vector table is to take one node away from the network and attack the microcontroller on this node with classic invasive or semi-invasive methods. The absence of the node from the network will probably be noticed by the surrounding nodes and its key(s) will be revoked. However, once the attacker succeeds with her long-term attack and learns the BSL password of the one node, it is trivial for her to attack all of the other nodes in the field if they all have the same BSL password.

If an attacker knows the BSL password, reading or writing the whole flash memory takes only about 25 s. In order to avoid these forms of attack, *interrupt*

vector randomization [6] can be used. This significanyly raises the bar for an attacker but is not yet part of standard software distributions.

3.2.3. Attacking the External Flash

Some applications might want to store valuable data on the external EEPROM. For example, the Deluge implementation of network reprogramming in TinyOS stores program images received over the radio there. If these images contain secret key material, an attacker might be interested in reading or writing the external memory.

Probably the simplest form of attack is eavesdropping on the conductor wires connecting the external memory chip to the microcontroller. Using a suitable logic analyzer makes it easy for the attacker to read all data that are being transferred to and from the external EEPROM while she is listening. If a method were found to make the microcontroller read the entire external memory, the attacker would learn all memory contents. This kind of attack could be going on unnoticed for extended periods of time, as it does not influence normal operation of the sensor node.

A more sophisticated attack would connect a second microcontroller to the I/O pins of the flash chip. If the attacker is lucky, the mote microcontroller will not access the data bus while the attack is in progress, and the attack will be completely unnoticed. If the attacker is skillful, she can sever the direct connection between the mote microcontroller and the flash chip, and then connect the two to her own microcontroller. The attacker could then simulate the external memory to the mote, making everything appear unsuspicious.

Of course, instead of using her own chip, the attacker could simply do a "mass erase" of the mote's microcontroller and put her own program on it to read the external memory contents. This operation is even possible without knowledge of the BSL password. While this causes "destruction" of the node from the network's point of view, in many scenarios this might not matter to the attacker.

The exact amount of time required for the attacks proposed above remains to be determined. It should be noted that some of the attacks outlined above require a high initial investment in terms of equipment and development effort. A possible countermeasure could be checking the presence of the external flash in regular intervals, putting a limit on the time the attacker is allowed to disconnect the microcontroller from the external flash.

3.2.4. Sensors

Sensor nodes rely on their sensors for information about the real world, so the ability to forge or suppress sensor data can be classified as an attack. For instance, a surveillance system might be tricked into thinking that the situation is normal while the attacker passes unnoticed through the area under surveillance.

Replacing sensors on the different types of nodes varies in difficulty between child's play and serious electrical engineering, mostly depending on the type of connection between the microcontroller circuit board and the sensors. A pluggable connection—as present on the Mica2 motes—requires an attacker to spend only a few moments of mechanical work. If, on the other hand, the sensors are integrated

into the printed circuit board design, replacing them involves tampering with the conductor wires, cutting them, and soldering new connections. The amount of time required for this will vary with the skill of the attacker, but it can be assumed to be in the order of minutes.

3.2.5. Radio

Finally, the ability to control all radio communications of a node might be of interest to an attacker, e.g., in order to mount an attack using deliberate collisions on the medium access level. We are unaware of any work trying to evaluate the effort necessary to perform such an attack and compare its effort with other attacks that target resource exhaustion and denial-of-service.

3.3. Summary

To summarize, we can classify the above attacks into three categories depending on the effort necessary. We list these classes in order of increasing severity:

1. The class containing the "easy" attacks: Attacks in this class are able to influence sensor readings, and may be able to control the radio function of the node, including the ability to read, modify, delete, and create radio messages without, however, having access to the program or the memory of the sensor node. These attacks are termed "easy" because they can be mounted quickly with standard and relatively cheap equipment.
2. The class containing the "medium" attacks: Attacks in this class allow to learn at least some of the contents of the memory of the node, either the RAM on the microcontroller, its internal flash memory, or the external flash memory. This may give the attacker, e.g., cryptographic keys of the node. These attacks are termed "medium" because they require non-standard laboratory equipment but allow to prepare this equipment elswehere, i.e., not in the sensor field.
3. The class containing the "hard" attacks: Using attacks in this class the adversary complete read/write access to the microcontroller. This gives the attacker the ability to analyze the program, learn secret key material, and change the program to his own needs. These attacks are termed "hard" because they require the adversary to deploy non-standard laboratory equipment in the field.

A different way to classify the above attacks is to look at the time during which the node cannot carry out its normal operation. Here we have the following classes:

1. Short attacks of less than five minutes. Attacks in this class mostly consist of creating plug-in connections and making a few data transfers over these.
2. Medium duration attacks of less than thirty minutes. Most attacks which take this amount of time require some mechanical work, for instance (de-) soldering.
3. Long attacks of less than a day. This might involve a non-invasive or semi-invasive attack on the microcontroller, e.g., a power glitch attack where the

timing has to be exactly right to succeed, or erasing the security protection bits by UV light.
4. Very long attacks which take longer than a day. These are usually invasive attacks on the electronic components with associated high equipment cost.

In thw following section we take these practical insights as the basis for devising a framework of adversary models that can be used for security analysis of WSNs.

4. Adversary Models

Adversary models should be determined with respect to applications. Who are adversaries and what goals do they have? A military sensor network has other security requirements than a network for habitat monitoring. The adversaries can be classified according to the following parameters: goals, intervention, presence, and available resources. We first give an overview over these parameters and then treat the paramaters *intervention* and *presence* in detail later.

4.1. Overview

4.1.1. Goals

When designing security mechanisms in practice, it helps to know the *goals* of the adversary as precisely as possible. Which of the three classical security requirements (Confidentiality, Integrity, Availability) does the adversary try to violate? If the data are valuable (i.e., legitimate users have to pay) or privacy relevant, the adversary would try to gain unauthorized access. If the data are critical (e.g., building or perimeter security), the adversary would try to modify data, such that the alarm is not raised in case of intrusion. Also a denial-of-service attack can successfully disable the network (violating Availability).

Identifying the goals of the adversary is probably the most difficult aspect of security analysis in practice. Therefore the three security requirements Confidentiality, Integrity, and Availability are often treated in a uniform manner (all of them are goals of equal importance).

4.1.2. Presence

The parameter *presence* basically identifies *where* the adversary acts in a wireless sensor network. The basic distinguishing factor is the number and location of the nodes which are in the range of his influence. We distinguish *local*, *distributed* and *global* adversaries.

4.1.3. Intervention

While the presence parameter identifies *where* the adversary can act, the *intervention* parameter identifies *what* the adversary can do in these places. We distinguish *eavesdrop*, *crashing*, *disturbing*, *limited passive*, *passive*, and *reprogramming* adversaries. These classes offer incremental steps of power: Briefly spoken, an eavesdropping adversary can only listen to communication, a crashing adversary

can additionally destroy sensor nodes, a disturbing adversary can additionally upset sensors by manipulating their location or their readings, a limited passive adversary can additionally open a node and steal all its secrets by temporarily removing it from the network, a passive adversary can steal all secrets without displacing the node, and a reprogramming adversary can additionally take full control of a node and cause it to act in arbitrary ways.

4.1.4. Available Resources

There are several resource classes to consider: funding, equipment, expert knowledge, time. In the world of tamper resistance, the adversaries are divided into three classes: *clever outsiders*, *knowledgeable insiders* and *funded organizations* [1]. Commodity sensor networks are likely to be attacked by clever outsiders (who are probably just trying things out) and possibly by knowledgeable insiders, if a substantial gain from the attack can be expected. Available resources are interdependent with the parameters *intervention* and *presence*. However, the precise connection is not always clear. For example, a global adversary need not to be a funded organization. A hacker could subvert the entire sensor network by exploiting some software bug. Or, if a local adversary manages to capture a sensor node and read out its cryptographic keys, he can turn into a distributed or global adversary, depending on how many sensor nodes he is able to buy and deploy as clones of the captured node.

4.1.5. Outlook and Notation

In our view, the presence and intervention parameters are the most relevant ones for basic security analysis. This is why we now look at these two in detail. Before we present them, we need some formal notation.

We model a sensor network as a graph $G = (V, E)$ where the set V is the set of sensor nodes and the set $E \subseteq V \times V$ defines a neighborhood relation. The *number of all sensor nodes* $|V|$ is denoted N. Two sensor nodes v_1 and v_2 are *neighbors* if and only if they are in the neighborhood relation, i.e., $(v_1, v_2) \in E$. A set of nodes $\mathcal{V}$ is *connected* if and only if for all $v_1, v_2 \in \mathcal{V}$ holds that either $(v_1, v_2) \in E$ or $(v_2, v_1) \in E$. Note that in our notation E is *not* necessarily reflexive and transitive because we wish to model sensor networks at a very low layer, i.e., without any routing or transport mechanisms.

4.2. Presence

We now discuss the different and increasingly severe levels of the presence parameter. These levels are defined in an incremental fashion, i.e., every level includes the behavior of the previous one. This implies a total order of parameter values defined by the subset relationship of behaviors. We begin our explanations by starting with the weakest level first. In general, an attacker model is a (time-dependent) subset relationship of the entire set of sensors.

- local adversary
  A *local adversary* can influence a small localized part of the network [9], for example he has one receiver which can only eavesdrop on several meters,

local → distributed → global

Figure 4. Adversary presence parameter.

or can manipulate only the sensor node which is the closest to him (for example, it is installed in his office).
Formally, an adversary is local if his range of influence contains a small connected subset $S \subset V$ of all sensor nodes. The set S can also be given as a Boolean function $S : V \mapsto \{\text{true}, \text{false}\}$.Such a set is small if its size is several magnitutes smaller than the number of total nodes, i.e., $|S| << N$. This set can also change over time, but changes are rather slow.
In general it is always possible to define S in a time-dependent manner, i.e., $S : V \times T \mapsto \{\text{true}, \text{false}\}$ where T is the domain of time values.

- distributed adversary
  A *distributed adversary* models either a mobile adversary (a car with receiver driving around) or managed to install his own sensor nodes in the sensor field and coordinates them [2].
  Formally, an adversary is distributed if its range of influence consists of multiple unconnected small subsets of nodes of the sensor network. As an example of such an adversary, consider the assumption that sensor nodes are attacked randomly following a uniform distribution [50]. In such a case, compromised nodes are distributed over the entire network, but not wide enough to actually see, hear or listen anything.
- global adversary
  A *global adversary* is the most powerful level of presence an adversary can exhibit. Such an adversary can analyze the complete network, hence his area of influence consists of *all* sensor nodes.

We define the relation → to order attacker models in the direction of stronger (i.e., more severe) attacker behavior. Formally, model $A \rightarrow B$ if and only if all behaviors which are possible in A are also possible in B (behavior subsetting). The levels of ability ordered with respect to → are depicted in Figure 4.

4.3. Intervention

Now that presence has been investigated and an ordered set of presence abilities has been given, the same needs to be done for the intervention capabilities an adversary can be ascribed.

The intervention order is constituted of the following levels, starting from the weakest and proceeding towards the strongest. Every level contains the behaviors of all preceding levels.

- eavesdropping adversary
  An eavesdropping adversary can just listen to network traffic, do nothing else, in particular no jamming etc. Her can then analyse this traffic either online or offline.
- crashing adversary
  A crashing adversary exhibits the simplest for of failure: The node simply stops to operate (execute steps of the local algorithm). Elsewhere this is

eavesdrop → crash → disturbing → limited passive → passive → reprogramming

Figure 5. Adversary intervention parameter.

also called *failstop adversary* [8] A *crashing* adversary attacks sensors such that they completely break down. The sensors can be destroyed or drained of energy.

- disturbing adversary
  A *disturbing* adversary can try to partially disturb protocols, even if he does not have full control over any sensors. He can selectively jam the network, or fool some sensors into measuring fake data. For example, he can hold a lighter close to a temperature sensor [47] or re-arrange the topology of the sensor network by throwing sensors around.
- limited passive adversary
  An adversary of this level can retrieve all information on a node, but in order to do so he needs to completely remove the node physically from the network. This means that the danger of being caught is higher.
- passive adversary
  A passive adversary can directly go and open up arbitrary sensor nodes to get the information currently stored in such a node, there is not need to leave the network to do so. Furthermore, such an adversary is assumed to be able to modify the data the node contains.
- reprogramming adversary
  A *reprogramming* adversary can run arbitrary programs on a sensor node. This can be achieved by exploiting some software bug, or by probing out cryptographic keys and other secret information and then cloning the node as described above. The problem of *node capture* is typical for a malicious adversary.

The order above defines the different abilities of intervention that can be ascribed to an adversary. These can be ordered according to the relation → as defined above (subsets on behaviors), see Figure 5.

4.4. Adversary Model Lattice

Having defined ordered levels for both presence and intervention, those two abilities are now combined to form the complete adversary models. An adversary model $\mathcal{A}$ is a pair (Presence, Intervention), where the first component specifies the level of presence the adversary model assumes and analogously the second component states the level of intervention. A fully specified adversary model thus looks like $\mathcal{A}(local, disturbing)$, in case of an adversary with local presence and disturbing skills. The result is a partial order of adversary models. This partial order is depicted in Figure 6 as a lattice. It can be seen that the most powerful adversary is $\mathcal{A}(global, reprogramming)$ (in the upper right corner) and the weakest is $\mathcal{A}(local, eavesdrop)$ (in the bottom left corner). As it is a partial order not all adversary models are comparable in the sense that one model is truly stronger than another. $\mathcal{A}(global, eavesdropping)$ and $\mathcal{A}(local, reprogramming)$ are such a pair, as neither of them is stronger than the other. However some models can

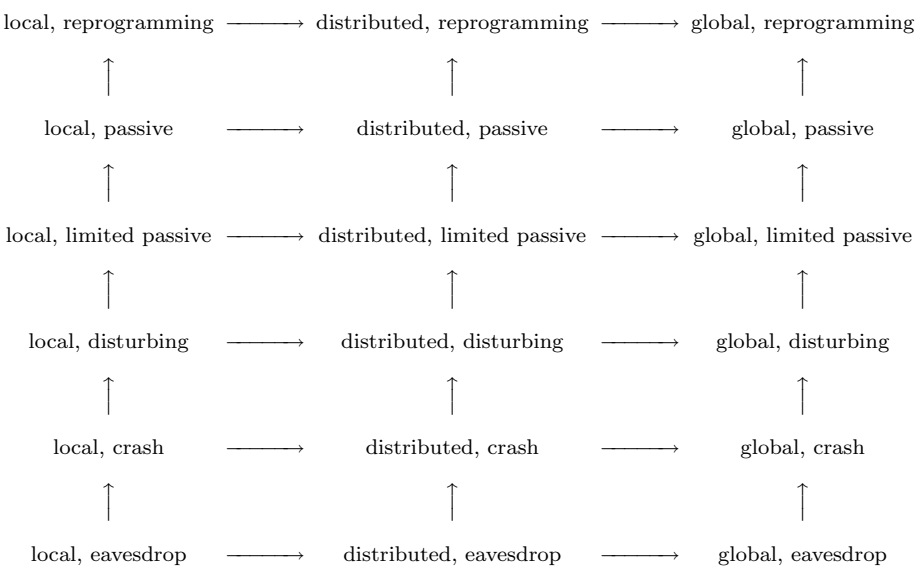

Figure 6. Adversary Model Lattice

be put into relation. We take the relation $\rightarrow$ to be the combination of the trace subsetting ordering defined for presence and intervention parameters above. The resulting partial order is depicted in Figure 6.

There are issues that only concern one ability, either presence analysis or presence intervention, thus being ignorant about the other ability. For such cases, a $*$ symbol can be inserted for notational convenience into the appropriate component of the model to stand of all different levels possible. For instance, wanting to talk about an adversary that has local presence skills and neglecting completely intervention, the adversary model $\mathcal{A}(local, *)$ would be appropriate to refer to such an adversary.

The ordering of the levels of abilities yields the nice property that a statement ascribing an adversary of a low level (for example a $\mathcal{A}(local, *)$) to perform a malicious action, will automatically ascribe the same for adversaries of higher levels ($\mathcal{A}(local, *)$ and $\mathcal{A}(global, *)$). Similarly, it should be clear that an algorithm that can cope with a $\mathcal{A}(global, passive)$ adversary will be able to sustain all other presented adversaries. However, as the lattice presents only a partial order, an algorithm coping with a $\mathcal{A}(local, \text{limited passive})$ adversary will not necessary work as well with a $\mathcal{A}(global, eavesdrop)$.

4.5. Summary

We have presented an abstract framework for modeling attackers in WSNs. We claim that due to our experiments described in Section 3 these abstract classes model well the critical differences between the hardness of sultions.

5. Discussion of Protection Mechanisms

In Section 3 we systematically investigated physical attacks on current sensor node hardware, paying special attention to attacks which can be executed directly in the deployment area, without interruption of the regular node operation. We found out that most serious attacks, which result in full control over a sensor node (*node capture*), require absence of a node in the network for a substantial amount of time. We also found simple countermeasures for some of the most serious attacks.

Thus, in order to design a WSN secure against those node capture attacks described in this chapter, the following steps should be applied:

- take standard precautions for protecting microcontrollers from unauthorized access;
- choose a hardware platform appropriate for the desired security level, and keep up-to-date with new developments in embedded systems security;
- monitor sensor nodes for periods of long inactivity;
- allow for revocation of the authentication tokens of suspicious nodes.

Standard precautions for protecting microcontrollers from unauthorized access, such as disabling the JTAG interface, or protecting the bootstrap loader password, are an absolute prerequisite for a secure sensor network. We developed a method of protecting the bootstrap loader password by randomization of the interrupt vector table. This allows the developers to make source code of their products public without fearing that their WSN can be taken over by everybody who compiles the source code using the same compiler, thus obtaining the same interrupt vector table, and therefore, the same BSL password.

As security is a process, not a product, system designers should keep up-to-date with the developments in attacks on embedded systems. The security of important systems should be constantly re-evaluated to take new discoveries into account, as newly found attack methods on microcontrollers or previously unknown vulnerabilities might make a previously impossible low-cost attack in the field possible.

The level of security required from the application should also be kept in mind when choosing hardware. In some cases it might make sense to build additional protection, such as a secure housing, around a partially vulnerable microcontroller.

Finally, the removal of a sensor node from the deployment area can be noticed by its neighbors using, e. g., heartbeat messages or topology change notifications, as well as by the sensor node itself using, e. g., acceleration sensors. Appropriate measures can then be taken by the network as well as by the node itself. The network might consider a node that has been removed as "captured" and revoke its authorization tokens or initiate recovery when this node returns to the network [46]. The node itself might react to a suspected physical attack by erasing all confidential material stored on it.

Mechanisms should be developed that allow a sensor node which has been absent for too long from the network to be revoked by its neighbors. This is our future work. Note that depending on the WSN design, local revocation could

be insufficient. For example, if an attacker removes a single sensor node from the network and successfully extracts the node's cryptographic keys, the attacker would be able to *clone* nodes, to populate the network with new sensor nodes which all use the cryptographic keys of the captured sensor node. Thus, a WSN should also be protected from node cloning.

In general, for passive adversaries, encryption techniques often suffice to ensure inside security. Symmetric key encryption techniques will be favored over asymmetric ones because of the computational advantage and the comparatively small key sizes. The problems arise in the setup phase of the network where shared secrets need to be distributed either by the manufacturer at production time or by clever protocols at deployment time [2, 24].

For stronger adversaries, active attacks like impersonation and node capture must be taken into account. Ideally, sensor nodes should be made tamper proof to prevent node capture, e.g., by applying technology known from smart cards or secure processing environments. There, memory is shielded by special manufacturing techniques which makes it more difficult to physically access the stored information [1]. Similarly, sensor nodes could be built in a way that they loose all their data when they are physically tampered with by unauthorized entities.

For large sensor networks, cost considerations will demand that sensor nodes are not tamper proof. Therefore, node capture must be taken into account. A first step to protect a sensor network from node capture against a local or partially present adversary is to use locally distributed protocols that can withstand the capture of a certain fraction of nodes in the relevant parts of the network. For example it is possible to devise access control protocols that work even if up to t out of n nodes in the communication range of the adversary are captured [9]. While these protocols can exploit broadcast communication, they are still relatively resource intensive.

A very general approach to counter node capture is to increase the effort of the adversary to run a successful attack. For example, data may be stored at a subset of nodes in the network and continuously be moved around by the sensors to evade the possible access by the adversary [7]. This makes it harder for the adversary to choose which node to capture. In the worst case, the adversary must capture much more than t out of n nodes to successfully access the data in question.

A similar technique that exploits the inherent redundancy of a sensor network and shields against even global adversaries is to devise it as a virtual minefield. A certain fraction of sensors has special alerting software enabled which scans the communication in its vicinity and reacts to local and remote manipulations by issuing a distress signal in its environment or otherwise cause an unpleasant experience to the adversary. Thus, these sensors act as "mines" in the network which the adversary tries to avoid since capturing them needs more resources than capturing a normal sensor node. If it is not possible for the adversary to distinguish these special sensors from normal sensors, the ratio between the fraction of "mines" and the effort to capture them determines the incentive of the adversary to attack the system. For example, if 10% of the sensors are virtual mines and it takes 1000 times more effort to capture a mine than to capture a

normal node, the adversary will waste a lot of resources if he needs to capture even a small subset of sensors without raising an alarm.

6. Summary and Conclusions

We have described security goals, adversary models and protection mechanisms which are relevant and specific for sensor networks. There are a lot of interesting problems and open questions in this area:

- *Realistic* adversary models should be derived with respect to existing and future applications. Here, experiences with GSM and WLAN security (and security failures) can be used as a guideline, but every application needs to define its own adversary model to be able to talk about security.
- What other attack possibilities exist for sensor networks and how much effort do they cost to be pursued? For example, are software-based node capture attacks a real threat? Are side-channel attacks on sensor nodes possible? We believe both to be true, but we are unaware of any work which has tried it.
- As cross-layer integration is especially important for resource-constrained sensor nodes, careful design decisions must be taken concerning which security means to put into which layer. For example TinySec [25], a link layer encryption and message integrity protection mechanism, is integrated into the radio stack of MICA Motes.
- Building secure sensor networks, especially with respect to active adversary, remains a challenge. Can it be done by combining existing solutions, such as random key predistribution, secure routing, secure data aggregation, or would it be too expensive in terms of energy?

Overall, we speculate that probabilistic algorithms which exploit the redundancy of the sensor network to cause high effort for the adversary will be good candidates to establish security in such networks. In a sense, these algorithms mimic guerrilla tactics: evasion and disguise. They can be simple, yet unpredictable. However, their simplicity implies that they are not suitable to establish perfect security. The security goals of sensor networks will be probabilistic and depend on the strength of the adversary.

Acknowledgements

We wish to thank Alexander Becher and Maximillian Dornseif for many helpful discussions and the delightful cooperation in breaking sensors.

References

[1] Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems.* John Wiley & Sons, Inc., 2001.

[2] Ross J. Anderson, Haowen Chan, and Adrian Perrig. Key infection: Smart trust for smart dust. In *ICNP*, pages 206–215, 2004.

[3] Ross J. Anderson and Markus G. Kuhn. Low cost attacks on tamper resistant devices. In *Proceedings of the 5th International Workshop on Security Protocols*, pages 125–136, London, UK, 1998. Springer-Verlag.

[4] Atmel Corp. AT45DB041B datasheet. Atmel document no. 3443, available at `http://www.atmel.com/dyn/resources/prod_documents/doc3443.pdf`.

[5] Atmel Corp. ATmega128 datasheet. Atmel document no. 2467, available at `http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf`.

[6] Alexander Becher, Zinaida Benenson, and Maximillian Dornseif. Tampering with motes: Real-world physical attacks on wireless sensor networks. In *Security in Pervasive Computing, Third International Conference, SPC 2006*, Lecture Notes in Computer Science 3934, pages 104–118. Springer, 2006.

[7] Zinaida Benenson, Peter M. Cholewinski, and Felix C. Freiling. Simple evasive data storage in sensor networks. In *IASTED PDCS*, pages 779–784, 2005.

[8] Zinaida Benenson and Felix C. Freiling. On the feasibility and meaning of security in sensor networks. In *Proceedings 4th GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze", Zurich, Switzerland*, March 2005.

[9] Zinaida Benenson, Felix C. Gärtner, and Dogan Kesdogan. An algorithmic framework for robust access control in wireless sensor networks. In *Second European Workshop on Wireless Sensor Networks (EWSN)*, January 2005.

[10] FU Berlin. ScatterWeb Embedded Sensor Board. Online at `http://www.inf.fu-berlin.de/inst/ag-tech/scatterweb_net/esb/`.

[11] Erik-Oliver Blass, Joachim Wilke, and Martina Zitterbart. A Security–Energy Trade-Off for Authentic Aggregation in Sensor Networks. In *IEEE Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON), Extended Abstract*, pages 135–137, Washington D.C., USA, September 2006. ISBN: 1-4244-0732-X.

[12] Haowen Chan, Adrian Perrig, and Dawn Song. Random key predistribution schemes for sensor networks. In *IEEE Symposium on Security and Privacy*, pages 197–213, May 2003.

[13] Chipcon AS. CC1000 datasheet. Available at `http://www.chipcon.com/files/CC1000_Data_Sheet_2_3.pdf`.

[14] Chipcon AS. CC2420 datasheet. Available at `http://www.chipcon.com/files/CC2420_Data_Sheet_1_2.pdf`.

[15] Crossbow, Inc. MICA2 data sheet. Available at `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf`.

[16] Crossbow, Inc. MPR, MIB user's manual. Available at `http://www.xbow.com/Support/Support_pdf_files/MPR-MIB_Series_Users_Manual.pdf`.

[17] J. Deng, R. Han, and S. Mishra. A performance evaluation of intrusion-tolerant routing in wireless sensor networks. In *2nd IEEE International Workshop on Information Processing in Sensor Networks (IPSN 2003)*, April 2003.

[18] Tassos Dimitriou and Dimitris Foteinakis. Secure and efficient in-network processing for sensor networks. In *First Workshop on Broadband Advanced Sensor Networks (BaseNets)*, 2004.

[19] Laurent Eschenauer and Virgil D. Gligor. A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 41–47. ACM Press, 2002.

[20] Abhishek Ghose, Jens Grossklags, and John Chuang. Resilient data-centric storage in wireless ad-hoc sensor networks. In *MDM '03: Proceedings of the 4th International Conference on Mobile Data Management*, pages 45–62. Springer-Verlag, 2003.

[21] Mark G. Graff and Kenneth R. van Wyk. *Secure Coding: Principles and Practices*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2003.

[22] Michael Howard and David LeBlanc. *Writing secure code*. Microsoft Press, pub-MICROSOFT:adr, second edition, 2003.

[23] Lingxuan Hu and David Evans. Secure aggregation for wireless networks. In *SAINT-W '03: Proceedings of the 2003 Symposium on Applications and the Internet Workshops (SAINT'03 Workshops)*, page 384. IEEE Computer Society, 2003.

[24] Joengmin Hwang and Yongdae Kim. Revisiting random key pre-distribution schemes for wireless sensor networks. In *SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, pages 43–52. ACM Press, 2004.

[25] Chris Karlof, Naveen Sastry, and David Wagner. TinySec: A link layer security architecture for wireless sensor networks. In *Second ACM Conference on Embedded Networked Sensor Systems (SensSys 2004)*, November 2004.

[26] Chris Karlof and David Wagner. Secure routing in wireless sensor networks: Attacks and countermeasures. *Elsevier's Ad Hoc Network Journal, Special Issue on Sensor Network Applications and Protocols*, September 2003.

[27] Geoff Martin. An evaluation of ad-hoc routing protocols for wireless sensor networks. Master's thesis, University of Newcastle upon Tyne, May 2004.

[28] Gary McGraw and John Viega. *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison-Wesley, September 2001.

[29] Microchip Technology. 24AA64/24LC64 datasheet. Available at `http://ww1.microchip.com/downloads/en/DeviceDoc/21189K.pdf`.

[30] RF Monolithics. TR1001 datasheet. Available at `http://www.rfm.com/products/data/tr1001.pdf`.

[31] moteiv Corp. Telos revision B datasheet. Available at `http://www.moteiv.com/products/docs/telos-revb-datasheet.pdf`.

[32] moteiv Corp. Tmote Sky datasheet. Available at `http://www.moteiv.com/products/docs/tmote-sky-datasheet.pdf`.

[33] `http://mspgcc.sourceforge.net/`.

[34] Holger Peine. Rules of thumb for secure software engineering. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 702–703, New York, NY, USA, 2005. ACM Press.

[35] Adrian Perrig, John Stankovic, and David Wagner. Security in wireless sensor networks. *Commun. ACM*, 47(6):53–57, 2004.

[36] Bartosz Przydatek, Dawn Song, and Adrian Perrig. SIA: Secure information aggregation in sensor networks. In *ACM SenSys 2003*, Nov 2003.

[37] Sergei P. Skorobogatov. Semi-invasive attacks - a new approach to hardware security analysis. Technical report, University of Cambridge, Computer Laboratory, April 2005. Technical Report UCAM-CL-TR-630.

[38] Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. In *CHES '02: Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, pages 2–12, London, UK, 2003. Springer-Verlag.

[39] STMicroelectronics. M25P80 datasheet. Available at `http://www.st.com/stonline/products/literature/ds/8495.pdf`.

[40] Texas Instruments. Features of the MSP430 bootstrap loader (rev. B). TI Application note SLAA089B, available at `http://www-s.ti.com/sc/psheets/slaa089b/slaa089b.pdf`.

[41] Texas Instruments. MSP430 F149 datasheet. Available at `http://www-s.ti.com/sc/ds/msp430f149.pdf`.

[42] Texas Instruments. MSP430 F1611 datasheet. Available at `http://www-s.ti.com/sc/ds/msp430f1611.pdf`.

[43] Texas Instruments. MSP430x1xx family: User's guide. TI Application note SLAU049E, available at `http://www-s.ti.com/sc/psheets/slau049e/slau049e.pdf`.

[44] John Viega, Matt Messier, and Gene Spafford. *Secure Programming Cookbook for C and C++*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2003.

[45] Harald Vogt. Exploring message authentication in sensor networks. In *Security in Ad-hoc and Sensor Networks (ESAS), First European Workshop*, volume 3313 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 2004.

[46] Harald Vogt, Matthias Ringwald, and Mario Strasser. Intrusion detection and failure recovery in sensor nodes. In *Tagungsband INFORMATIK 2005, Workshop Proceedings*, LNCS, Heidelberg, Germany, September 2005. Springer-Verlag.

[47] David Wagner. Resilient aggregation in sensor networks. In *SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, pages 78–87. ACM Press, 2004.

[48] A. Wood, J. Stankovic, and S. Son. JAM: A mapping service for jammed regions in sensor networks. In *In Proceedings of the IEEE Real-Time Systems Symposium*, December 2003.

[49] Sencun Zhu, Sanjeev Setia, and Sushil Jajodia. LEAP: efficient security mechanisms for large-scale distributed sensor networks. In *Proceedings of the 10th ACM conference on Computer and communication security*, pages 62–72. ACM Press, 2003.

[50] Martina Zitterbart and Erik-Oliver Blaß. An Efficient Key Establishment Scheme for Secure Aggregating Sensor Networks. In *ACM Symposium on Information, Computer and Communications Security*, pages 303–310, Taipei, Taiwan, March 2006. ISBN 1-59593-272-0.