Dynamic Code Update for the Efficient Usage of Security Components in WSNs

Axel Poschmann[◇], Dirk Westhoff[★], and Andre Weimerskirch[▽]

[◇] Horst Görtz Institute for IT Security
Communication Security Group (COSY)
Ruhr-Universität Bochum, Germany
[★] NEC Europe LTD
[▽] ESCRYPT Embedded Security GmbH
poschmann@crypto.rub.de, dirk.westhoff@netlab.nec.de,
aweimerskirch@escrypt.com

Abstract. Wireless sensor networks (WSN) will have a major impact on our everyday's life. A requirement for large-scaled deployment are extremely low-cost sensors though running with minimal resources regarding computational power, energy consumption, and memory size. Cryptographic schemes are highly at demand for providing security mechanisms in such WSNs. Asymmetric cryptography allows for flexible key management schemes at the cost of high resource demands whereas symmetric cryptography provides resource efficient solutions.
In this work we sketch an approach for (1) providing asymmetric cryptography during the one-time bootstrapping phase and (2) swap it by other security protocols for operation of the WSN in order to minimize memory size demands. Our mechanism is based on dynamic code update, e.g. provided by the FlexCup plug-in for TinyOS. Our approach yields the best of two worlds in order to maximize flexibility and life-span and minimize resource demands.

Keywords: wireless sensor network, WSN, hybrid security approaches, dynamic code update

1 Introduction

During the last years wireless sensor networks (WSN) became more and more interesting for the research but also for the industrial community. Nowadays it seems that there exists a huge variety of applications suited to WSNs. Such applications will fundamentally change our daily lives due to its ubiquitous facet and will result in financial savings as well as more comfort and safety.

The use of digital signatures and public-key (PK) encryption provided by asymmetric cryptography is widely discussed in the community. Often, it is stated that asymmetric cryptography is too resource demanding for the highly constrained sensor devices. However, recently several working groups came up with running implementations of asymmetric cryptography on sensor devices.

One example is the elliptic curve cryptography (ECC) library for Mica Motes [GPW+04]. However, one needs to classify the Mica Motes belonging to one of the most powerful sensor device classes. For an envisioned deployment of thousands of devices far more constrained and in particular less costly devices need to be introduced. In particular, we foresee that application specific controllers at minimal cost will be used (e.g., low-cost 4 or 8-bit controllers with very little memory). Here, the often quoted Moore's law cannot be applied in its traditional way (i.e. doubling of performance) but in an adapted way such that the prize reduces by half each 18 months. Hence, it can be foreseen that the cumulated number of all deployed sensors will regularly double but the device's resources will stay extremely low. From such observations we conclude that solutions for efficiently deploying asymmetric cryptography will still be highly in demand at any time.

The usage of asymmetric cryptography has been pointed out to be very costly in terms of storage, computation, and energy on restricted devices, e.g. 8-bit ATMEL ATmega128L micro-processor (Mica Motes). However, asymmetric cryptography in particular would be very valuable for initial bootstrapping of the security association within WSNs. In such kind of networks it is usually only possible to establish pair-wise security associations after the roll-out of the sensor nodes. In particular a pre-established secret before the roll-out can not always be assured.

From conventional network security we know that security protocols are usually based on a hybrid approach, namely a combination of asymmetric and symmetric cryptography like e.g. done in SSL, PGP, and IPSEC. Asymmetric keys are used for the key establishment and subsequent communication is secured by purely using symmetric schemes. Ideally, such an approach should also be applicable to WSNs. What brings this vision close to reality is the following observation: In the arena of WSNs and in particular for large scaled sensor networks there is the ultimative need for means of code update during runtime of the system. Otherwise such complex and highly distributed systems would not be manageable and maintainable in realistic roll-out scenarios. Although quasi-standardized OS-solutions do not explicitly support dynamic code update (e.g. TinyOS [HSW+00]), operating systems which support such mechanisms are already available (Contiki [DGV04], MANTIS OS [ABC+03]).

The main contribution of this work is *to use dynamic code update for the establishment of hybrid security concepts in WSNs*: during the initial roll-out phase of a WSN, we propose using traditional asymmetric cryptographic schemes for establishing various types of security associations. Subsequently, by dynamic code update we propose to replace the binary code for the asymmetric operations by application related binary code. We argue that binary code for asymmetric schemes essentialy is only needed during the roll-out phase of the sensor network also taking into account the moderate lifetime of a WSN.

With the above sketched approach we see substantial benefit due to the following reasons:

- *Cost-efficient fabrication at the manufacturer*: There is no need for a specific pre-configuration of the nodes at the manufacturer with respect to the envisioned security.
- *Flexibility with respect to various key-management models*: Depending on the concrete WSN application and its security requirements the approach is flexible enough to support various key-management models like pairwise-, groupwise or network-wide key-management during the bootstrapping phase of the system.

Finally it is obvious that dynamic code update with the objective to establish hybrid security concepts in WSNs is only feasible if binary code update is verifiable by the sensor nodes during runtime. In principle two solutions are possible:

1. Verification based on asymmetric schemes and in case of a successful verification subsequently replacement of asymmetric scheme component.
2. Verification based on some other pre-established secret, e.g. during the configuration phase at the manufacturer.

Note that although dynamic code update in principal is usable many times, we are solely aiming at a single code update directly after the roll-out phase of the WSN. More concretely, we distinguish between the binary code required during the roll-out and the initial bootstrapping of the system and binary code which is required for the subsequent operation in the application phase. In the remainder of this article we will concretize our approach for the operating system TinyOS, the de facto standard operating system for WSNs. Unfortunately TinyOS only allows to exchange the whole program code and not, as required, only parts of the application code. Therefore, several approaches have been published to expunge this shortcoming. Out of these proposals we chose FlexCup. However, FlexCup enables us to demonstrate and validate our approach on the de facto standard operating system TinyOS.

2 Related Work

The proposed idea touches related work in three different domains: constrained devices, state-of-the-art implementations of PK cryptography, and operating systems for constrained devices.

2.1 Devices Background

WSNs are composed of sensor nodes, which are typically highly constrained devices with low computational power and a small memory size. Applications shall be designed to work for months or even years without changing the battery of the sensor nodes. Therefore, power-efficiency is a crucial requirement for applications implemented on sensor nodes.

Mica Motes, developed by UC Berkeley and distributed by Crossbow Technology [xbo], are emerging as the de facto standard devices for WSN researchers.

The mote family MICA2, MICA2DOT and MICAz consist of a low-power 8-bit ATMEL ATmega128L RISC microprocessor running at 4 MHz, 128 KB of internal FLASH program memory, 4 KB of internal SRAM data memory, 512 KB of external FLASH data memory, several sensors, and a radio interface.

The TmoteSky (or Telos B) motes, developed by Moteiv and distributed by Crossbow, serve as an example of more constrained devices in terms of program memory. They consist of an ultra low-power 16-bit RISC Texas Instruments MSP40 microcontroller running at 8 MHz, 48 KB of internal FLASH program memory, 10 KB of internal SRAM data memory, 1024 KB of external FLASH data memory, several sensors, and a radio interface. Note that TmoteSky motes feature roughly only a third of program memory compared to Mica motes (48 KB vs. 128 KB) while at the same time the processor word size is doubled (16 bit vs. 8 bit).

For both device calsses power is supplied by two AA batteries, which provide approximately 1000 milliamp-hours. Sending or receiving of a byte needs far more energy than computing one byte for example with AES [WGE+05]. Also access to the FLASH memory causes relatively high energy consumption compared to pure computation. However, if only energy consumption of the processor is considered, without concerning energy consumption of memory or radio interface, than the energy consumption of an application is directly proportional to the execution time.

Since the forementioned motes are quite expensive we do not believe that real-life WSNs will consist of such modular mote families. Furthermore, since memory size is one of the biggest cost drivers for all embedded systems we believe that real-life sensors will be highly integrated ASICs with very constrained program memory (FLASH).

2.2 Public-Key Cryptography

Public-key cryptography is desired for several reasons in WSNs. Examples are authentication or key establishment. In [GPW+04] Gura et al. compare the execution time and memory usage of standardized ECC [Cer00] and RSA with different keylengths on an ATmega128 8-bit RISC processor. Their results are summarized in the upper part of Table 1, whereas time figures for the lower part are calculated from tables in [Wur03]. As one can see, processing a standardized ECC with 160 bits takes 1.62 seconds compared to 21.98 seconds for a private-key RSA-1024 operation and 0.96 milliseconds for an AES-128 encryption.

These figures underline that even a fast implementation of a PK cipher is still up to three orders of magnitude slower on 8-bit RISC processors than a symmetric cipher. In addition PK ciphers generally need more data memory and have a larger code size, which is caused by their larger operands length. Since a longer execution time increases power consumption, it is desirable to use symmetric ciphers instead of PK ciphers whenever possible. We conclude that PK cryptography should be used very rarely in WSNs.

Algorithm	μ-processor	time [ms]	RAM [B]	code [B]
ECC serp160r1	ATmega128	1,620	282	3682
ECC serp192r1	ATmega128	2,480	336	3979
ECC serp224r1	ATmega128	4,380	422	4812
RSA-1024 public	ATmega128	860	542	1073
RSA-1024 private	ATmega128	21,980	930	6292
RSA-2048 public	ATmega128	3,840	1332	2854
RSA-2048 private	ATmega128	166,520	1853	7736
RC5-32/12/16	ATmega32	0.28	104	1170
AES-128	ATmega163	0.96	8	1326
IDEA	ATmega32	0.7	104	824
Skipjack	ATmega32	0.68	0	708
RC4	ATmega32	0.02	258	332

Table 1. Performance of symmetric and asymmetric ciphers on 8-bit ATMEL AVR RISC μ-processors

2.3 Operating Systems for Constrained Devices

Several operating systems for constrained devices have recently been developed, such as TinyOS [HSW+00], MANTIS OS [ABC+03], Contiki [DGV04], and SOS [HKS+05] amongst many others. TinyOS is the de facto standard operating system for WSNs. It is written in the event-driven programming language nesC [GLvB+03], which is specially suited for the needs of network embedded systems.

Although TinyOS allows to exchange the whole program image, unfortunately unlike MANTIS OS, Contiki, or SOS, it does not allow for separately exchanging system components and application components. However, to expunge this shortcoming several approaches have been published, such as Mate [LC02], Impala [LM03], Deluge [HC04], FlexCup [MGL+06], and MiLAN [HMCP04].

In the remainder of this article, we will focus on FlexCup. FlexCup is an application that consists of a compiler-extension (FlexCup-Analyzer), a middleware component (code distribution algorithm), a stand-alone operating system (FlexCup-Linker), and a kernel component (FlexCup-Bootloader). It is part of the TinyCubus project [MLM+] and is based on TinyOS. To guarantee full portability to other frameworks FlexCup is written in ANSI C. FlexCup can update parts of TinyOS as well as exchange application components power-efficiently [MGL+06].

In the next section we provide a detailed description of all steps performed during a code update with FlexCup. Recall that we chose FlexCup even though it has a rather high memory requirement, because it enables us to demonstrate our approach on the de-facto standard operating system TinyOS.

3 Our Approach

Deployment scenarios of WSNs can be split into at least two phases: bootstrapping phase and normal operating phase. During bootstrapping phase, nodes perform operations which are only necessary for initialization of the system, while in the latter case nodes perform their real duties, i.e. sensing and sending data (sensors nodes) or receiving, processing and sending data (aggregator nodes) to the base station. E.g. the UbiSec&Sens project[1] provides many security components which are purely needed during the bootstrapping phase: topology aware group keying (TAGK), a key pre-distribution scheme which supoports the end-tp-end encryption of convergecast traffic, and concepts like concealed data aggregation (CDA [DW06]). Such concepts require different forms of homomorphic encryption transformations which are fully useless during the bootstrapping phase. On the other hand, a set of security components is purely required during the normal operating phase. For example, when techniques are required to persistently store encrypted data within the network (TinyPEDS [JG06]) or when end-to-end encryption of convergecast traffic with in-network processing is required (CDA [DW06]). Taking into account that TmoteSky motes have only 48 KB of program memory (and that real-life deployed sensors may have even less), it is obvious that not all applications will fit at the same time on a mote. Therefore we propose to use dynamic code update to replace components which are only used during the initial bootstrapping phase by components which are purely required in normal operating mode.

For example, PK cryptography is very inefficient on constrained devices, especially in terms of power consumption. On the other hand, by using the PK component, nodes are enabled to establish various types of security relations during bootstrapping. This is essentially true for different forms of pair-wise, group-wise, or network-wide key-establishment. Note that we do not focus on a specific key-distribution scheme. However, our approach supports any key-distribution scheme and provides all means for designing such flexible and efficiently at the same time. Therefore, we propose to use PK cryptography only during bootstrapping to establish security relations. Subsequently, the PK and TAGK components are not required anymore and are substituted by components which support the system in its normal operating phase. Note that this scenario enables to e.g. use highly optimized implementations of a PK cryptographic scheme without considering too much the code-size, since the PK component is used just a few times during bootstrapping.

However, to prevent an adversary from placing false or malicious code on the node, it is crucial that new components are verified by the node before installation. Recall that in our scenario, nodes use PK cryptography purely during bootstrapping. Therefore, in our scenario nodes verify the first code update by an asymmetric scheme to check components.

[1] UbiSec&Sens is a STREP of the EU Framework Programme 6 for Research and Development with the aim to provide a toolbox of cryptographic applications. For further informations see http://www.ist-ubisecsens.org

Requirements: We assume that it is possible to fabricate sensor nodes with a customer specific initial setup, e.g. in our scenario the nodes are delivered with a minimal operating system, FlexCup, a PK cryptographic scheme (PK component), and a built-in public-key.

Network Model: In our considerations we clearly focus on highly constrained devices in terms of energy, memory space, and computational power; Mica-Motes serve as an example for upper-bound devices. We further assume that all nodes, since they are deployed with the initial setup described above, are able to perform a code-update. Contrary to sensor nodes, the base station is not expected to be constrained in terms of energy, memory, and computational power. All devices communicate via a wireless broadcast medium, e.g. IEEE 802.15.4 LR-WPAN. We assume the communication to be bi-directional and in the general case multihop. The code-update is multicast traffic from the sink-node to the sensor nodes, whereas the principle traffic-pattern for monitored data is convergecast from many nodes to the sink node.

Adversary Model: The well-known Dolev-Yao threat model [DY83] is considered to be the de-facto standard threat model for formal analysis of crypto-protocols. However, in the context of WSNs it is not suitable, because it assumes secure devices as endpoints of an insecure channel. In WSNs secure devices mean tamper-resistant devices, because otherwise adversaries might pick up devices and tamper it. However, tamper-resistance is an expensive task, if even achievable, and furthermore, we are talking about WSNs consisting of hundreds or thousands of nodes. Therefore we generally assume devices in our model to be unprotected or at most partly-protected. Partly-protected means that devices can be tampered but for relative high costs and only after a certain delay in time, providing therefore a higher security level than unprotected devices. Furthermore, we assume that there is a rather short time window between deployment and initial code update, where no adversary is able to tamper or steal a node. However, with respect to our implicitly considered adversary model, eavesdropping of the whole communication is possible at any time.

Let us have a closer look on what exactly has to be done for a successful substitution of the PK component for components supporting the system's operating phase (see Figure 1). At this point we want to emphasize that since symmetric schemes like AES or RC5 are already available when using IEEE 802.15.4 or TinySEC, the security model of TinyOS, respectively, we do not deal with uploading such code anymore.

1. **Meta-Data Generation -** First, FlexCup-Analyzer analyzes the executable program file, the component object files, and the map file of the linker. With the gained knowledge, FlexCup-Analyzer produces the meta-data in binary format. The meta-data consist of informations about the whole program, component specific informations, symbol informations, and the relocation

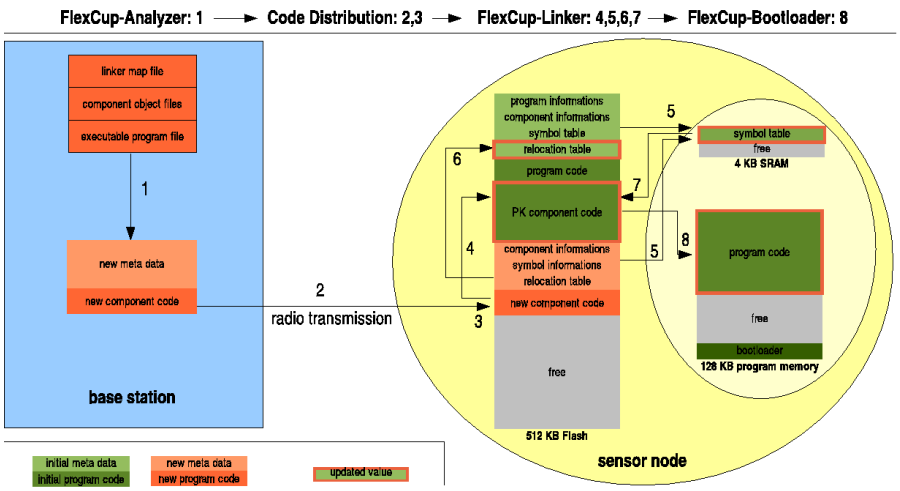

Fig. 1. FlexCup Steps for Dynamic Code Update

table. When substituting the PK component by an application specific component, FlexCup-Analyzer produces the meta-data for the application specific component. FlexCup-Analyzer performs all steps at the base station.

2. **Code Distribution -** During the second step, the application specific components together with their meta-data are transmitted from the base station to the node. Here FlexCup uses a specific code distribution algorithm, which is related to Deluge [HC04].
3. **Data Storage on the Node -** Once data of the application specific component are received at the radio interface of the node, they are stored in the external FLASH memory. In the FLASH memory also a complete copy of the program memory is stored. This image can be copied into the FLASH memory right before code update starts.
4. **Verification of code and substitution of PK component -** Recall that in our scenario every node is deployed with a build-in public-key. Moreover, every component's code is digitally signed by the code publisher's private-key, which is usually the private-key of the owner of the WSN.
   - **4.1 Verification of the component's digital signature -** Nodes verify the the digital signature of the received binary code of the component.
   - **4.2. Substitution of the PK component -** Now FlexCup-Linker can start to work. First, it substitutes the PK component code by the application specific component code. If the code size of the application specific component is larger than the code size of the PK component, FlexCup-Linker moves following program code backwards, if code size is smaller it moves it forth, respectively.
5. **Symbol Table Update -** Next, FlexCup-Linker updates the symbol table. Therefore it uses the SRAM to merge information of the new component's

meta-data with the old symbol table. If entries have changed, the MSB of the symbolidentifier is set, thus indicating the linker to patch the references.

6. **Relocation Table Substitution -** FlexCup-Linker now substitutes the old relocation table with the new one, which is part of the meta-data of the application specific component. This is done by simply copying the new relocation table to the position of the old one. In the layout of the program memory, there is a buffer between a relocation table and the next code segment. Therefore, only if the new relocation table exceeds the old one plus the buffer size, there is additional overhead.
7. **Update References -** FlexCup-Linker is able to patch all references, by examining the updated symbol table. When finished, FlexCup-Linker resets the "update"-flags in the symbol-table.
8. **Copy to Program Memory and Reset -** Finally, FlexCup-Bootloader copies the new program code from FLASH memory to program memory. Afterwards FlexCup-Bootloader jumps to memory address 0x000000, thus realizing a reset of the node.

After the code update the node's program code no longer contains components which are only required during bootstrapping, for example the PK component. It rather consists of TinyOS and components which supports the WSN in its normal operating phase. Surely this may be again a security-related component, e.g. CDA for ensuring end-to-end encryption for convergecast traffic or TinyPEDS for secured storing environmental data in a distributed manner. If no further updating of the component's code is desired, even the FlexCup component can be replaced.

4 Open Issues and Conclusion

The scenario described in Section 3 enables various further possibilities, such as autonomous code download with respect to the relative position of a node within a WSN. Once all sensor nodes are deployed with the basic setup, the base station can broadcast different application components for various purposes in the network. Sensor nodes then can autonomously decide which application component they need to download. Sensor nodes need to know their relative position in the WSN topology or at least their distance in terms of hops to the sink node, though.

Beside the upcoming possibilities of this scenario, there are several issues to be solved. Most important of all is the verification of the components for code update. There are two general approaches to cope with it:

1. Verification based on asymmetric schemes and in case of a successful verification subsequently replacement of binary code for asymmetric scheme.
2. Verification based on some other pre-established secret, e.g. a shared key which was established during bootstrapping.

Both of these approaches need further investigations and an accurately defined adversary model.

In this article we presented a new approach for efficient application of public-key cryptographic schemes in wireless sensor networks. We pointed out that energy inefficiency of asymmetric schemes is not an obstacle for usage, when only used a few times during bootstrapping. By substituting the public-key cryptographic scheme with an application specific component for usage in the operation phase, even devices with very limited memory resources are able to benefit from public-key cryptographic schemes. Moreover, by substituting applications specific to the nodes current needs, in general devices with less program memory are required. Hence, even low-cost devices with very limited memory resources can meet requirements to provide a wide range of applications. This will significantly lower cost for WSNs and may boost deployment of new WSNs.

5 Acknowledgments

The authors would like to thank Matthias Gauger for insightful comments on the FlexCup approach. The work presented in this paper was supported in part by the European Commission within the STREP UbiSec&Sens of the EU Framework Programme 6 for Research and Development (www.ist-ubisecsens.org). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the UbiSecSens project or the European Commission.

References

[ABC+03] H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, J. Deng, and R. Han. MANTIS: System Support for umltimodAl NeTworks of In-Situ Sensors. *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pages 50–59, 2003.

[Cer00] Certicom Research. SEC 2: Recommended Elliptic Curve Domain Parameters. Standards for Efficient Cryptography Version 1.0, September 2000.

[DGV04] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. *lcn*, 00:455–462, 2004.

[DW06] M. Acharya. D. Westhoff, J. Girao. Concealed Data Aggregation for Reverse Multicast Traffic in Wireless Sensor Networks: Encryption, Key Predistribution and Routing. regular paper at IEEE Transactions on Mobile Computing, August 2006.

[DY83] D. Dolev and A. Yao. On the Security of Public Key Protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, 1983.

[GLvB+03] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC Language: A Holistic Approach to Networked Embedded Systems. *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pages 1–11, 2003.

[GPW+04] N. Gura, A. Patel, A. Wander, H. Eberle, and S.C. Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004)*, 2004.

[HC04] J.W. Hui and D. Culler. The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale. *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 81–94, 2004.

[HKS+05] C.C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava. A Dynamic Operating System for Sensor Nodes. *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 163–176, 2005.

[HMCP04] WB Heinzelman, AL Murphy, HS Carvalho, and MA Perillo. Middleware to Support Sensor Network Applications. *Network, IEEE*, 18(1):6–14, 2004.

[HSW+00] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System Architecture Directions for Networked Sensors. *SIGOPS Oper. Syst. Rev.*, 34(5):93–104, 2000.

[JG06] E. Mykletun T. Araki. J. Giroa, D. Westhoff. TinyPEDS: Persistent Encrypted Data Storage in Asynchronous Wireless Sensor Networks. to appear in Elsevier Ad Hoc Journal, 2006.

[LC02] Philip Levis and David Culler. Mate: A Tiny Virtual Machine for Sensor Networks. *SIGOPS Oper. Syst. Rev.*, 36(5):85–95, 2002.

[LM03] T. Liu and M. Martonosi. Impala: A Middleware System for Managing Autonomic, Parallel Sensor Systems. *Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 107–118, 2003.

[MGL+06] Pedro José Marrón, Matthias Gauger, Andreas Lachenmann, Daniel Minder, Olga Saukh, and Kurt Rothermel. FlexCup: A Flexible and Efficient Code Update Mechanism for Sensor Networks. In *Proceedings of the Third European Workshop on Wireless Sensor Networks (EWSN 2006)*, pages 212–227, February 2006.

[MLM+] PJ Marron, A. Lachenmann, D. Minder, J. Hahner, R. Sauter, and K. Rothermel. TinyCubus: A Flexible and Adaptive Framework Sensor Networks. *Wireless Sensor Networks, 2005. Proceeedings of the Second European Workshop on*, pages 278–289.

[WGE+05] A.S. Wander, N. Gura, H. Eberle, V. Gupta, and S.C. Shantz. Energy Analysis of Public-key Cryptography for Wireless Sensor Networks. *Third IEEE International Conference on Pervasive Computing and Communications (PERCOM?05)*, 2005.

[Wur03] Ulrich Wurst. Einsatz kryptographischer Verfahren auf stark ressourcenbeschränkten Geräten. Master's thesis, University Stuttgart, November 2003. (in german only).

[xbo] Crossbow Technology, Inc. http://www.xbow.com.