UbiSec&Sens
FP6-2004-IST-4
Contract no.: 26820

UbiSec&Sens

Deliverable 0.2 (updated version)

Scenario driven security analysis and architecture driven requirement specification

Editors:	Peter Langendörfer, IHP; Evgeny Osipov, LTU
Deliverable nature:	R (Report)
Dissemination level: (Confidentiality)	PU (Public)
Contractual delivery date:	30/06/2008
Actual delivery date:	30/06/2008
Suggested readers:	Experts and executives in the area of wireless sensor networks, security experts
Version:	2.0
Total number of pages:	61
Keywords:	Secure Wireless Sensor Networks, Scenario Description, Middleware, Initial Module Description

Abstract

The main aim of the Deliverable D02 is to provide security and requirement analysis for the chosen scenarios in UbiSec&Sens.

The document defines the chosen scenarios in detail and provides an overview and an evaluation of hardware and software platforms that are currently available. It discusses how the requirements that are defined by the UbiSec&Sens scenarios can be fulfilled by these platforms. Finally it defines an idealised hardware architecture as well as an optimised middleware solution and describes an initial software and hardware architecture which will be used for the demonstrator setup.

The results were strongly influenced by lifetime issues, since there is a need for finding a balance between the lifetime and the required level of security. In addition to their influence, the scenarios are defining a range of potential settings for the demonstrators. The wide range of different security settings has led to the idea of a flexible security support, backed by appropriate architecture and tool support.

Impressum

[Full project title] Ubiquitous Sensing and Security in the European Homeland

[Short project title] UbiSec&Sens

[Number and title of work-package] WP0 “Architecture”

[Document title] Scenario driven security analysis and architecture driven requirement specification

[Editor: Name, company] Peter Langendörfer, IHP; Evgeny Osipov, LTU

[Work-package leader: Name, company] Dirk Westhoff, NEC

[Estimation of PM spent on the Deliverable] 19 PM

Executive summary

The main aim of the deliverable D02 is to provide security and requirement analysis for the chosen scenarios.

In order to define a systems architecture requirements of reference scenarios have been defined and analysed. As basis for the evaluation important parameters such as energy consumption of available sensor nodes have been evaluated. Our analysis clearly indicates that a flexible architecture of our security middleware is needed, since for example not all security modules can be stored at a sensor node. Such a security middleware provides an interface to potential applications and is connected to the hardware by an hardware- and operating system-dependent abstraction layer. Application and middleware are linked at compile time, what results in minimum overhead. To update security algorithms after deployment an efficient dynamic code update mechanism is required.

With such an architecture the requirements of our scenarios can be fulfilled to a certain extent. Better service can be achieved with improved HW, e.g. cryptographic accelerators or energy harvesting mechanisms, which is also discussed in the document.

The first section is discussing sample scenarios for each of our application fields providing much more detailed description per scenario compared to deliverable D01. The settings defined for each scenario provide information about the collected data type, sampling rate, number of nodes and expected lifetime. These scenarios are no longer toy examples, they are close to realistic set-ups for specific parts of large scale applications. In case of the agriculture scenario the area for which a sensor based monitoring is described goes up to 14 ha. The vehicular scenario does not look at that scale, but focuses on sensing at specific very dangerous parts of roads where a small number of sensors still provides valuable information. Similar holds true for the Homeland security scenario. The latter scenario is described for three rooms which can be seen as a certain hotel room and the aisle, so that a specific floor can be secured with this setting. The requirements that result from these three scenarios have guided our work described in the later sections.

The second section provides an overview and evaluation of hardware and software platforms that are currently available for realisation of wireless sensor network applications. We are focusing on existing hardware architectures for which we investigate their energy efficiency with a special focus on public key cryptography, since these are the most power hungry operations a sensor node will have to execute. We also investigate the current operating systems as well as middleware approaches for wireless sensor networks. Thus, this section explores the currently available design space for realisation of WSN applications.

In section 3 we discuss the extent to which existing system architectures combined of hardware and software can fulfil the requirements that are defined by our scenarios. Here we are again focusing on energy issues and memory requirements as a second parameter. The latter clearly shows that the software packages that are installed at the sensor nodes need to be as small as possible. Additionally, there is a need for the ability to exchange parts of the software because deployment of all security modules is infeasible. The available energy is also pretty limited and allows for small duty cycles only.

We further use our results to define an idealised hardware architecture as well as an optimised middleware solution in section 4 of the document.

We also use our findings for the description of an initial software and hardware architecture which will be used for our demonstrator set-up. This is presented in the final section 5.

Lifetime issues have strongly influenced the analysis presented in this document, since there is a need for finding a balance between the lifetime and the required level of security. In addition to their influence on our research results presented here the scenarios are defining a range of potential settings for our demonstrators. The wide range of different security settings has led the idea of a flexible security support, backed by an appropriate architecture and tool support. In work package 4 we will refine the scenarios in order to elaborate concrete parameters of the demonstrators which we will realise.

List of authors

Company	Author
<IHP>	<Peter Langendörfer, Krzysztof Piotrowski, Steffen Peter>
<LTU>	<Evgeny Osipov>
<RWTH>	<Christine Jardak>
<INOV>	<Renato Nunes, Antonio Grilo>
<NEC>	<Dirk Westhoff, Alban Hessler>

Contents

Executive summary **3**

List of authors **4**

Abbreviations **9**

Definitions **10**

1 Definition of Sample Scenarios **11**
1.1 Agriculture . . . 11
1.1.1 Sensor Data Type . . . 11
1.1.2 Node Types . . . 11
1.1.3 Network Type . . . 11
1.1.4 Spatial Extension and Number of Nodes . . . 12
1.1.5 Sampling Rate and Topology . . . 12
1.1.6 Lifetime . . . 13
1.1.7 Security Requirements . . . 13
1.2 Vehicular . . . 15
1.2.1 Sensor data type . . . 15
1.2.2 Amount of Sensors . . . 15
1.2.3 Sampling Rate . . . 15
1.2.4 Transmission Rate . . . 16
1.2.5 Spatial Extension . . . 16
1.2.6 Lifetime . . . 16
1.2.7 Security Requirements . . . 16
1.3 Homeland Security . . . 20
1.3.1 Sensor Data Type . . . 20
1.3.2 Amount of Sensors . . . 21
1.3.3 Sampling Rate . . . 21
1.3.4 Transmission Rate . . . 21
1.3.5 Spatial Extension . . . 21
1.3.6 Lifetimes . . . 22
1.3.7 Security Requirements . . . 22

2 Hardware and Software Evaluations **29**
2.1 Hardware . . . 29
2.1.1 Criteria . . . 29
2.1.2 Micro Controllers / Processors . . . 29
2.1.3 Size of Memory . . . 31
2.1.4 Radio . . . 32
2.1.5 Connection to Sensors / AD Converter . . . 33
2.2 Software Architecture . . . 33
2.2.1 Selecting the Operating System . . . 33
2.3 Middleware Approaches . . . 37

3 Requirements Analysis **38**
3.1 Software Parameters . . . 38
3.2 Evaluation of Hardware with Respect to Scenarios' requirements . . . 38

4 Hardware & Software Specification 40
4.1 Hardware specification . . . 40
4.1.1 The Optimal Node Configurations . . . 40
4.1.2 Idealised Hardware . . . 42
4.1.3 Hardware Crypto Accelerators . . . 42
4.1.4 Energy Harvesting . . . 43
4.2 System Architecture . . . 43
4.2.1 Network Architecture . . . 43
4.2.2 The Toolbox Approach . . . 45
4.2.3 Middleware Compiler . . . 45
4.2.4 Middleware Architecture . . . 46

5 Initial architecture for UbiSec&Sens demonstrators 52
5.1 Hardware platforms for demonstrators . . . 52
5.2 Software platform for demonstrators . . . 52
5.2.1 Application components . . . 53
5.2.2 Services components . . . 53
5.2.3 Networking component . . . 53
5.2.4 Middleware component . . . 53
5.2.5 Internal Information Base . . . 53
5.3 UbiSec&Sens software modules for planed demonstrators . . . 54
5.3.1 Networking components . . . 54
5.3.2 Network Security Components . . . 56
5.3.3 Secure services and middleware . . . 57

List of Figures

1 Monitored area in the agriculture scenario . . . 12
2 Overall WSN roadside architecture for Intelligent Transport Systems . . . 17
3 Laboratory homeland security scenario covering 3 rooms in a single floor . . . 22
4 Network set-up consisting of a heterogeneous network and four specific device classes . . . 44
5 A mobile reader here depicted as a car may move through a WSN while retrieving data from the WSN . . . 45
6 The middleware compiler appropriate UbiSec&Sens modules are selected due to application requirements and sensor node constraints. . . . 46
7 The UbiSec&Sens Middleware architecture: control center configuration at the left sensor node configuration at the right hand side . . . 47
8 Dependency graph of asymmetric cipher mechanisms as used by the UbiSec&Sens middleware compiler . . . 48
9 Sensor node states before and after deployment . . . 49
10 Software architecture for the demonstrators setup . . . 52
11 Specific software modules included in the UbiSec&Sens security toolbox . . . 54

List of Tables

1 The spatial extension and number of nodes in the agriculture scenario 12
2 Radio range requirements in the agriculture scenario 12
3 The deployment and sampling rate details for the agriculture scenario 13
4 Attacks in the agriculture scenario 14
5 Attacks in the vehicular scenario 18
6 Attacks in the homeland security scenario 24
7 ATmega128L and MSP430F1611 on chip peripherals and parameters 30
8 Time needed by the sensor nodes to perform SSL/TLS handshake 31
9 Power consumed by the sensor nodes to perform SSL/TLS handshake 31
10 ATmega128L and MSP430F1611 current consumption in power save modes 31
11 AT45DB041B and M25P80 flash memory parameters 32
12 Current and power consumption of the ZigBee transceiver CC2420. Power consumption calculated at 3V supply voltage. Power consumption per bit at transmission speed of 250 kbit/s . . . 32
13 Power consumption of the 433 MHz and 868 MHz transceiver CC1000. Power consumption calculated at 3V supply voltage. Power consumption per bit at transmission speed of 38.4 kbit/s 33
14 Available memory in popular sensor nodes. 38
15 Memory occupied by operating system software on a node from the MICA family. 38
16 Memory footprint for selected software pieces 39
17 Lifetime of the evaluated nodes considering diverse duty cycle settings 39
18 Parameters of the STR711FR2 32 bit micro controller 41
19 Comparison of $GF(2^m)$ ECPM hardware designs. 42

Abbreviations

ADC: Analog-to-Digital Converter
ASIC: Application Specific Integrated Circuit
CDA: Concealed Data Aggregation
DTSN: Distributed Transport for Sensor Networks
DCU: Dynamic Code Update
DF: Domingo Ferrer Privacy Homomorphism
ECC: Elliptic Curve Cryptography
ECPM: Elliptic Curve Point Multiplication
FPGA: Field Programmable Gate Array
HW: Hardware
IP: Internet Protocol
KPD: Key Pre-Distribution
MAC: Medium Access Control
OS: Operating System
PANEL: Position-based Aggregator Node ELection
PH: Privacy Homomorphism
RANBAR: RANSAC-based resilient aggregation
RANSAC: RANdom SAmple Consensus
RNG: Random Number Generator
RSI: Robust Sensor Initialisation
RSSI: Received Signal Strength Indication
SANE: Secure Aggregator Node Election
SMM: State Management Module
SW: Software
TAGK: Topolgy Aware Group Keying
TAUK: Topolgy Aware Unique Keying
tinyDSM: tiny Distributed Shared Memory
tinyLUNAR: tiny Lightweight Underlay Network Ad-hoc Routing
tinyPEDS: tiny Persistent Encrypted Data Storage
USS: UbiSec&Sens
WP: Work Package
WSN: Wireless Sensor Networks

Definitions

1 Definition of Sample Scenarios

In this section we are discussing sample scenarios for each of our application fields. The intention is to provide much more detailed description per scenario as we did in D0.1. The settings defined in the following subsections provide information on data such as data type collected, sampling rate, number of nodes and expected lifetime. These scenarios are no longer toy examples, but close to realistic set-ups for specific parts of large scale applications. In case of the agriculture scenario the area for which a sensor based monitoring is described goes up to 14 ha. The vehicular scenario does not look at that scale, but focusses on sensing at specific very dangerous parts of roads where the small number of sensors still provides valuable information. Similar holds true for the homeland security scenario. The scenario is described for three rooms which can be seen as a certain hotel room and the aisle, so that a specific floor can be secured with this setting. The requirements that result from our three scenarios have guided our work described in the later chapters. The wide range of different security settings has led to the idea of a flexible security support, backed by an appropriate architecture and tool support. Lifetime issues have strongly influenced the definition of an idealised hardware etc. In addition to their influence on our research results presented here these scenarios are defining a range of potential settings for considerable demonstrators. In workpackage 4 we will refine the scenarios in order to elaborate concrete parameters of the demonstrators which we will realise.

1.1 Agriculture

1.1.1 Sensor Data Type

The following sensor data types were provided by the owners of the Naegele vineyard in Germany.

Humidity is measured on the plants and in the ground.

Light is a considerable factor to measure for long term analysis. However, measuring the intensity of light remains of a minor interest for the vineyard owners, since it is currently unclear how to use or interpret such information in a vineyard context.

1.1.2 Node Types

Types of nodes to be used in this scenario:

Sensor nodes are equipped with measurement units (sensors) of one of the two types defined above. The sensor nodes are statically configured during the WSN roll-out.

The sensor nodes do not perform aggregation. Since humidity and light are relatively stable factors over a long period of time, the frequency of sensor readings is several readings per day.

Aggregator nodes are selected sensor nodes which in addition to sensing are dedicated for more advanced processing of the information collected by other sensor nodes. The aggregator nodes are dynamically elected during the network lifetime.

Periodically aggregator nodes transmit the processed information to the sink node, potentially over a multihop path.

Sink node is a gateway node between WSN and the control network. The sink node can either be mobile or static. We also refer to the mobile sink node as to a mobile reader.

1.1.3 Network Type

The type of the network is a hierarchical grid with two hierarchy levels.

Table 1: The spatial extension and number of nodes in the agriculture scenario

Area	Distance between nodes [m]	Topology	Number of nodes	Number of sensors
1 ha	25m	Grid	25	25 humidity sensors 8 light sensors
14 ha	25m	Grid	25x14 = 350	25x14 = 350 humidity sensors 8x14 = 112 light sensors
0.45 ha	25m	Grid	25/2 = 13	25/2 = 13 humidity sensors 8/2 = 4 light sensors

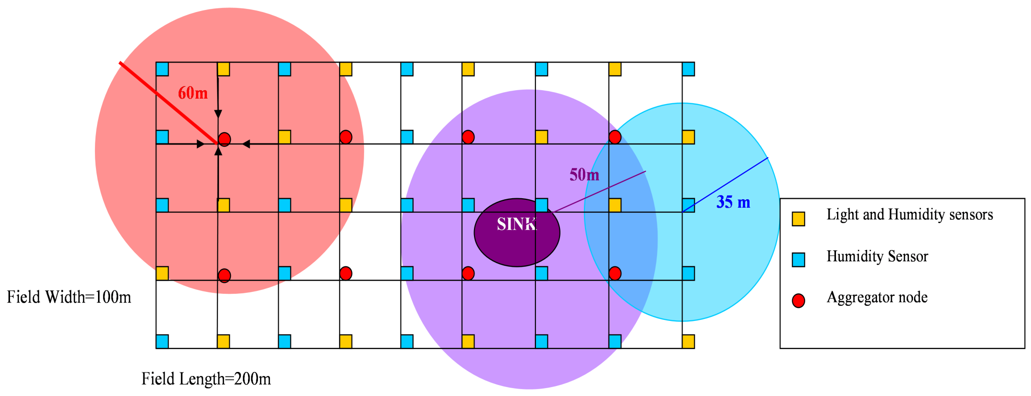

Figure 1: Monitored area in the agriculture scenario

1.1.4 Spatial Extension and Number of Nodes

According to the interest expressed by the vineyard owners a distance of 25 m between sensors is sufficient to measure humidity. Assuming a rectangular deployment area (100x200 m) the nodes are placed on a grid with the size of a cell 25m x 25m. In total 45 nodes are required. All nodes are equipped with humidity sensors. 15 of them should be equipped with light sensors. The distribution of the sensors in a sample vineyard is schematically shown in Figure 1. The figure also shows the position and the coverage of the aggregators, the simple nodes and the sink.

The range requirements of the network are shown in table 2

1.1.5 Sampling Rate and Topology

Table 3 presents the frequency of data dissemination and deployment details. For the sensor and aggregator nodes, i.e., nodes that are battery powered, this frequency indicates how many times per day the node has to wake up to sense (and/or receive in case of an aggregator) and to send the reading or aggregated readings.

Note that in order to measure the soil humidity some nodes equipped with the humidity sensors should

Table 2: Radio range requirements in the agriculture scenario

Node type	Range [m]	Comments
Aggregator nodes	60	In order to allow point to point communication with the neighbouring aggregators.
Sensor nodes	35	In order to have at least one aggregator in its communication range.
Static sink	50	A multihop routing protocol is needed to route the data from the aggregators to the sink.
Mobile reader	25	A multihop routing protocol is needed to route the data from the aggregators to the reader.

Table 3: The deployment and sampling rate details for the agriculture scenario

Node type	Selection	Functionality	Sending frequency	Deployment
Aggregator nodes	Using aggregator node election protocol	- sensing - aggregating readings	1 aggregated reading / day	- mounted above the ground (either on a pole or on the plants supporting infrastructure)
Sensor nodes	statically	- sensing	6 readings / day	- placed directly on the ground
Sink node	statically	- gathering the readings from WSN		- fixed or mobile sink

technologically be placed close to the ground. The humidity sensor nodes measuring the humidity level on a plant should be placed higher. All nodes with the light sensors should be mounted on poles for more precise measurements. It is foreseen that the higher located sensor nodes will have better radio coverage, therefore the role of an aggregator will migrate between these nodes. In order to balance the energy consumption an aggregator node election protocol should be deployed in these nodes. The precise placement of particular nodes will be defined during the deployment time after detailed investigation of the deployment area. The placement of nodes will ensure the radio coverage requirements specified above.

1.1.6 Lifetime

As described in the “Scenario Definition and Initial Threat Analysis” deliverable (D0.1), the monitoring period in a vineyard is equal to the vegetation period (second half of April - end of August). However, for specific types of measured factors the monitoring period may be more restricted. For example in the drying time (July-August) humidity measurements are done more frequently. Thus, we can assume that the expected lifetime of the sensor network shall be at least 5 months. In order to prolong the networks lifetime an aggregator node election protocol will be deployed in all nodes with potential aggregator function as described above. The precise rate of aggregator nodes re-election will be identified prior to deployment.

1.1.7 Security Requirements

The application of wireless sensor networks (WSN) technology in agriculture converges to a task of correlating the micro-climatic, bio-chemical factors during different growth stages of plants to the quality of the final product. The role of the sensor network in this case is to provide constant monitoring of these factors in an automatic way and dynamic delivering the measured data to the farmer.

In our scenario we aim at enhancing the quality of grapes what depends on two major factors during its growth: The moisture of the ground and the quantity of light a plant gets. Therefore, we implement a vineyard WSN equipped with moisture and light sensors. The WSN measures periodically the latter two factors and reports them periodically to the farmer.

Attacks from human beings or wild animals and unintended accidents due to agricultural engines can lead to a partial destruction of the WSN resulting in a loss of the measured data. In order to tackle these problems, measurements have to be stored in geographically distant nodes and allow the farmer to request this information when needed.

Since the target of this sensor network is enhancing the quality of the final wine product, malicious attacks from competitors should be taken in account. The large surface of a vineyard makes it impossible to be fully controlled against intruders. Therefore a faulty node could be discreetly inserted in the WSN in order to inject erroneous readings. A solution against this attack is realized by implementing a plausibility check of the measurements and discarding the erroneous ones based on statistics.

Table 4: Attacks in the agriculture scenario

Attack name	**Goal**	**System parts to attack**	**Technical means needed to attack**	**Technical attack description**	**Effort to succeed**	**Attacker type**	**UBI supported countermeasures**	**Effort to succeed after applying UBI countermeasures**
Manipulating sensors readings	Data poisoning	Sensors	Source of light or moisture applied to sensors	Sensor readings are manipulated with non-natural techniques	cheap	Clever Outsiders	The application of RANBAR on aggregator nodes	Depends on the size of the region under RANBAR supervision
Natural destruction	Loosing parts of the networks and store measurements	Sensor nodes	Animals, climate factors (rain)	Animals eating or smashing the electronic devices. Leaking sensor boxes.	cheap	$<$ Clever Outsiders	Installing sensors in waterproof transparent boxes and mount them on wooden poles.	Only Hard physical attacks can still destroy the motes.
Intended destruction	Interrupt the service / Destroy specific data	Sensor nodes	-	The attacker destroys random or specific nodes with an hammer	cheap	Clever Outsiders	Implementing TinyPEDS storing the collected data from one region on a remote aggregator	The attack is mitigated. Some data might be lost despite replication. Service might be interrupted if the connectivity of the WSN is brought down to a critical point.
Manipulating data messages	Inserting messages	Network communication	Laptop, sensor nodes, powerful wireless device	Attacker succeeds to imitate our message format and send jams to aggregator nodes	minimal	Knowledgeable Insiders	The application of RANBAR on aggregator nodes	Depends on the size of the region under RANBAR supervision

We sum up the major attacks which we have tackled in our vineyard application in Table 4. Referring to these security threats, a secure agriculture WSN application for vineyard monitoring shall be equipped with the following modules:

1. Persistent and replicated storage of monitored data at various nodes

   The tinyPEDS module provides persistent and replicated data storage. The readings collected from a group of motes in a region are backed up on another geographical region. This guarantees the availability of the measurements whenever an attack results in a partial handicap of the network.

2. Plausibility check of monitored data

   RANBAR is a plausibility check that runs on the aggregator nodes in order to discard poisoned readings inserted by an intruder. This module is desirable when the integrity of the data is important. It is based on statistical model that can probabilistically filter out malicious values after receiving several measurements.

3. WSN access control supporting relaxed mobility

   When a farmer needs to retrieve the data from the network, a reactive protocol is needed in order to route the query towards the destination. For this reason we use the Lightweight UNderlay Adhoc Routing (LUNAR) adapted for ad-hoc and wireless sensor networks called tinyLUNAR. The tinyLUNAR is a reactive end-to-end connection oriented routing protocol that uses a label switching forwarding technology. It offers a native support for data-centric and address-centric communications.

4. Reliability

   The reading values of the light and the moisture vary slowly in time as well as geographically. These climatic and bio-chemical facts yield a sort of a natural replication of the measurements resulting into a remedy against loosing measurement packets before storage. While reliability before storage is elective, it is mandatory for querying data from the network. Once the need of retrieving the data rises up, the querying mechanism needs to be reliable. This reliability is ensured by the nanoTCP transport protocol, an end-to-end connection oriented protocol using acknowledgements and retransmissions.

5. Energy efficiency and aggregation

   In order to extend the network life time, we decrease the number of messages communicated. Our chosen way is to adopt clustered network architecture. Sensors of each cluster collect readings periodically and send them to an elected aggregator who aggregates the readings and sends a single message with the computed measurement values to be backed-up. Being an aggregator node means higher energy consumption. Therefore a ”fair” load balancing scheme is needed which efficiently distributes the load between the nodes. We use PANEL for the periodical aggregator election based on a moving reference point. The closest node to the reference point in a cluster is chosen as the aggregator of the cluster.

1.2 Vehicular

1.2.1 Sensor data type

For the Vehicular WSN we plan to use sensors that measure the temperature on the road as well as detect a moving obstacle on the road (see Figure 2).

1.2.2 Amount of Sensors

The amount of sensors per critical region on the road should not be more than 10 sensor nodes plus one sink node.

1.2.3 Sampling Rate

Continuously at the sensor nodes, temporary at the sink node.

1.2.4 Transmission Rate

The road condition at critical days (indicated by the weather forecast) is monitored continuously; let's say every 10-15 minutes in a push mode. A push mode is only required in case the road temperature is in the range of the freezing point. For the measurement of movement pattern, we use the pull mode in case a moving object is detected.

1.2.5 Spatial Extension

We are aiming to use WSN technology in front of tunnels or bridges (temperature) or at curves on a country road. The spatial extension of the WSN is rather limited, say 100x20m with not more than 10 sensor nodes and one sink node.

1.2.6 Lifetime

The lifetime should be as long as possible. To flatly balance the energy a potential demonstrator will contain a derivate of a non-manipulable aggregator election protocol.

1.2.7 Security Requirements

We foresee that in the near future, two types of wireless networks will operate in an integrated manner aiming at an increased level of public safety and liability; Vehicular Ad Hoc Networks (VANETs) and Wireless Sensor Networks (WSNs).

The WSN-VANET scenario is aiming at a WSN roadside architecture for provisioning the two complementary services; the accident prevention and the post-accident investigation. The envisioned WSN Security architecture is stimulated by the understanding that WSN roadside islands will only be rolled-out when hardware costs are close to the minimum. Therefore, we are aiming at purely SW based security solutions which do not rely on costly HW components like road side units (RSU) or tamper resistant modules on sensor nodes.

The main objective of the WSN is to detect danger that neither the driver nor the car sensors can easily detect. These can be, for example, formation of ice at very particular segment of the road or animals irrupting out of woods. WSNs deployed at those areas could collect and process the data in order to clean it, and finally, if needed, send critical data or warnings to the car. The On-Board Unit (OBU) of the car decides how to react on this piece of information, e.g., warn the driver, trigger the automatic speed reduction, or engage further probing.

To improve the safety of other drivers and virtually extending the range of communication of the WSN, the car will try to geo-broadcast the warning to car that might drive into the danger. By using position-based ad-hoc network routing, only the cars in the region of interest (defined by the first car) will receive the warning packet.

To support the post-accident investigation service, sensor nodes continuously measure the road condition and store this information within the WSN itself. Storing the road condition over the long time may be of interest for a forensic team. In contrast to the accident prevention service such a liability service will be limited to a well specified group of end-users, e.g., insurance companies or the road patrol. Information stored within the WSN will be helpful to judge a driver's driving style according to the road condition at the moment of an accident.

The type of attacks we have to face for such a service is eavesdropping over the wireless and/or bogusly getting access to the WSN. In addition to these two, since the data are stored for a relative long time within the roadside WSN, data shall not be stored in plaintext. An attacker, who physically reads out the whole WSN or a fraction of it, would gain knowledge of the stored data. We also emphasize, that for the post-accident investigation service the integrity and the resilience of the stored environmental data is required. Also, it would be beneficial that monitored data are persistently stored at many nodes to prevent data loss in case nodes disappear or simply get stolen.

1.2.7.1 Attacks on the Road Safety applications and its counter-measures

The motivation of the attacker can differ:

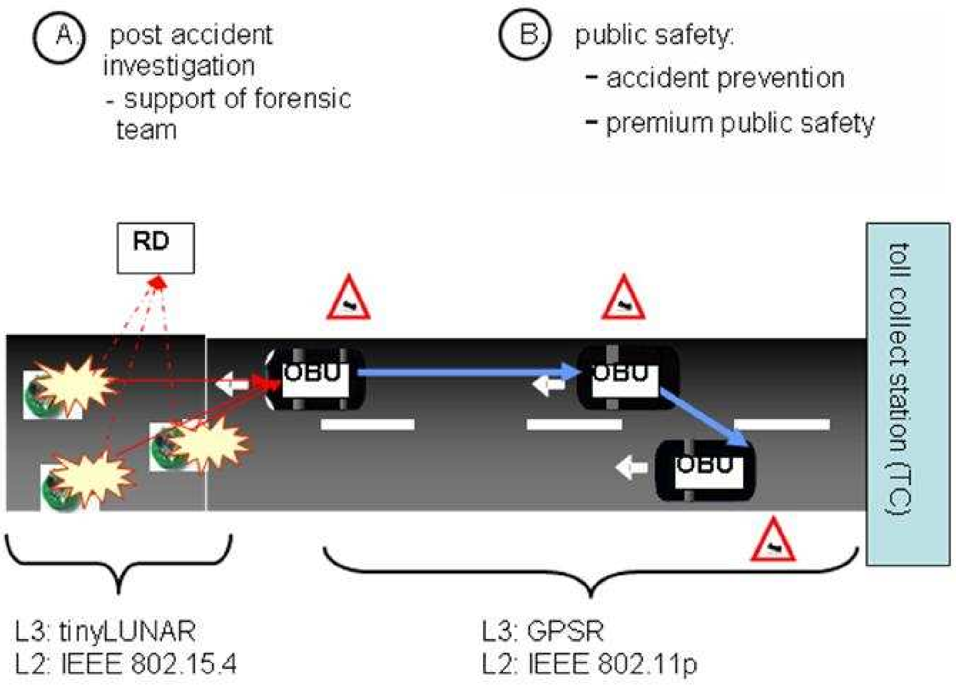

Figure 2: Overall WSN roadside architecture for Intelligent Transport Systems

- An attacker is willing to harm people or provoke accidents. To achieve this, she will either try to modify data or to disrupt the service. For example, the WSN could be corrupted to tell ”there is no danger”, while in reality, there is currently a high risk on the road.
- The attacker is simply malicious, and wants to disrupt the service. She will launch DoS attacks, jamming attacks, or destroy physically nodes.
- The attacker is greedy. She would use the WSN data for her own business. For example, stealing WSN that she does not own to build some valuable database.

Table 5: Attacks in the vehicular scenario

Attack name	**Goal**	**System parts to attack**	**Technical means needed to attack**	**Technical attack description**	**Effort to succeed**	**Attacker type**	**UBI supported countermeasures**	**Effort to succeed after applying UBI countermeasures**
Manipulating sensors readings	Data poisoning	Sensors	Cold/hot packs as much as sensors in a certain region	Sensor readings are manipulated directly	100-250 depending on prices for hot/cold packs	< Clever Outsiders	RANBAR if applied to much broader region	Depends on the size of the region under RANBAR supervision
Manipulating road conditions monitored	Disturbing the traffic	Network communication	Powerful wireless communication device(s) Laptop)	manipulate routing tables to become part of most of the routes attacking CDA schemes	1. Some hours traffic monitoring 2. just communicate fake data from captured motes	Knowledgeable Insiders	provable secure routing, improved/combined CDA schemes	Hard Attack is mitigated to have the least impact on the system
DoS	Degrade WSN lifetime, interrupt service	Network communication	Laptop	Attack broadcasts many messages, which will be received and computed by sensors	Minimal	Knowledgeable Insiders	Query authentication. If too many forgeries detected, go to sleep to save energy, if scenario permits	Attack is mitigated. Attacker needs more resources to succeed
Jamming	Interrupt service	Network communication	Laptop, powerful wireless device	Attacker jams the wireless medium for one or more nodes	Minimal	Clever Outsiders	Not tackled by Ubisec&Sens because the problem is at the physical layer	-
Eavesdropping / Memory extraction	Appropriate data for personal use	Network communication, memory protection	Laptop, wireless communication, specialised electronic tools	Attacker attacks physically a sensor node and reads out its memory	Expensive: thousands of euros. Possibly time consuming	Knowledgeable Insider - Funded Organisations	Use of Privacy Homomorphism to conceal data in an efficient way	Infeasible

Continued on next page

Table 5 – continued from previous page

Attack name	**Goal**	**System parts to attack**	**Technical means needed to attack**	**Technical attack description**	**Effort to succeed**	**Attacker type**	**UBI supported countermeasures**	**Effort to succeed after applying UBI countermeasures**
Manipulating Node Election	Redirect traffic to corrupted nodes	Network protocols	Laptop, specialised electronic tools	After successful corruption of a few nodes, the attacker tries to redirect to these nodes by electing them aggregator at any round	Expensive: thousands of euros. Possibly time consuming	Knowledgeable Insider	Use of secure aggregator election: SANE or PANEL with security extension	Infeasible
Destruction of nodes	Interrupt the service / Destroy specific data	-	None	The attacker destroys random or specific nodes with an hammer	Cheap	$<$ Clever Outsiders	tinyPEDS provides self-organisation (with the help of node election and flexible routing) and replication of data	The attack is mitigated. Some data might be lost despite replication. Service might be interrupted if the connectivity of the WSN is brought down to a critical point.

Clearly, the priority for the WSN operator is to thwart harmful attacks as it could endanger people life. We sum up the different attacks on the road safety application in Table 5. With respect to these security threats, a security architecture for a roadside WSN serving as accident prevention shall be equipped with the following modules:

1. WSN access control supporting relaxed mobility

   The reader device from the police or the insurance company is not necessarily required to pass the WSN with high velocity (if at all). Our routing and forwarding solution, tinyLUNAR, currently supports well nomadic mobility pattern. For the access control, with the Canetti-Benenson authentication scheme, the query can be initially broadcast from spatially anywhere in the WSN, thus mobility is well supported.

2. Persistent and replicated storage of monitored data at various nodes

   The tinyPEDS provides persistent and replicated data storage. If the attacker tries to destroy the data on the nodes, she will have to destroy many of them at very different places, since the data is replicated.

3. Encrypted storage of aggregated data which can still be applied to simple in-network computing

   If an attacker tries to read out memory, she will fail as the decrypting key is not on the node itself, and therefore she cannot read it.

4. Integrity of monitored data

   Monitored data should be authenticated to verify the originator as well as to ensure that the data have not been manipulated at transit. Memory protection on the node is a hard task since the hardware is not tamper-resistant. However, we could use at our advantage the dis-symmetry of the WSN: the reader device is trusted, along with being much more powerful. Therefore, memory protection can be ensured by using MAC with one-way chained keys. To protect aggregation, Manulis resilient data aggregation scheme can be used if the data is not encrypted.

5. Resilience and plausibility check of monitored data

   Monitored data should, besides being authenticated, also be recognised if same manipulated data have already 'wrongly' been monitored. In the vehicular scenario, the first aggregation will not be encrypted, allowing the aggregator to detect outliers thanks to the RANBAR module and discard them from the aggregated value.

1.3 Homeland Security

The Homeland Security scenario will illustrate the usage of a WSN to physically secure an area, such as a building floor, against unauthorised access. This scenario has practical applicability in many situations and could be used, for example, by a special operations team to secure an area that is going to be visited by a VIP. In this way, after a thorough inspection by officers, the WSN would be installed and configured allowing the area to be left unattended or under the supervision of a very reduced task force.

The WSN prototype for Homeland Security applications will integrate relevant routing, security and reliability components from WP1-WP3. This is an "In-lab" prototype with 15-25 nodes running in a secure way.

1.3.1 Sensor Data Type

The Homeland Security application scenario will use the following types of sensors:

Movement detector infrared/microwave sensors that detect people's presence or movement;

Acoustic sensor microphone-based sensor used to detect voices, objects falling or other sounds;

Smoke detector sensors that detect smoke due to a fire or significant amounts of particles or dust in the air;

Tamper detector micro switches or other types of electrical switches that may detect the repositioning of the WSN node, the opening of its enclosure or other type of interference with it;

Intrusion detector magnetic switches, pressure switches or other types of switches used, for example, to detect the opening of doors or windows.

The interfaces with the sensors described above use both digital and analog inputs. Most sensors are digital (binary output: active/inactive) but the acoustic sensor is analog and will be read using an Analog-to-Digital Converter (ADC). This approach allows a simple adaptation in the future to other types of sensors to allow, for example, the detection of biological threats or anomalous radiation levels, broadening the application area of the WSN to more vast and complex security scenarios.

1.3.2 Amount of Sensors

As described in the project's Technical Annex, the Homeland Security WSN prototype is supposed to use a number of nodes in the range 15 to 25. We will target the use of 20 nodes, 18 of which will include sensors and the remaining 2 will be used as sink nodes (see Figure 3 for an illustration). We anticipate the use of the following sensors:

- 8 movement detectors;
- 4 acoustic sensors;
- 1 smoke detector;
- 5 Tamper/Intrusion detectors (electrical switches);

1.3.3 Sampling Rate

The different sensors will have to be sampled at an adequate rate in order to allow gathering of relevant information with the required temporal detail (e.g., evolution of an analog quantity) and without loosing sporadic events (for example, short pulses in electrical switches). Given the nature of the sensors and considering the mechanical natural of the electrical switches, we propose the use of a 100ms period between samples, i.e., sampling rate of 10Hz. This sampling rate has an impact in power consumption and it should be as low as possible.

1.3.4 Transmission Rate

In the Homeland Security application scenario sensors will be sampled and processed at an adequate rate, but will only generate messages whenever some predefined level is exceeded (in case of analog sensors) or when a binary sensor is activated. This is expected to occur infrequently and the alarm information to be transmitted will occupy only a few bytes.

Besides the alarm information described above, and because of security reasons, each sensor will have to send periodically an ”alive” report message informing it is working properly and that it has not detected anything abnormal. To guarantee that a sensor does not stop fulfilling its mission without notice (because it was damaged, tampered with, cannot communicate due to RF interference or simply because it run-out of battery power), we propose that each node should send an ”alive” message once each minute.

This traffic can be easily accommodated considering the data link transmission rate but may have an impact in power consumption. To minimise this, some sort of data aggregation of these ”alive” messages should be implemented, to reduce the amount of messages sent to the sink nodes.

1.3.5 Spatial Extension

As described in the ”Scenario Definition and Initial Threat Analysis” deliverable (D0.1), the inter-node spacing varies according to type of sensor and application. For presence/intrusion detection, an inter-node spacing of 5 m is considered acceptable.

For the Homeland Security WSN prototype, we will consider a scenario with 3 rooms, each about 20 m^2. Each room will have 6 nodes, as illustrated in figure 3, for a total of 18 nodes, all equipped with sensors. These nodes may elect among them nodes that will perform aggregation functions. Additionally, there will be a sink node associated with the portable reader and a sink node associated with the Control Center. In total, an application scenario could use 20 nodes.

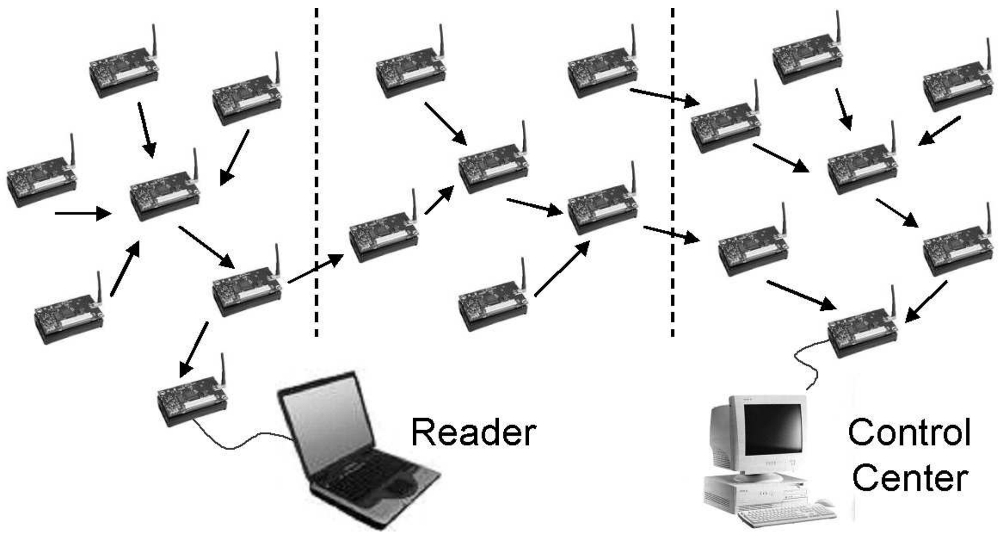

Figure 3: Laboratory homeland security scenario covering 3 rooms in a single floor

1.3.6 Lifetimes

As described in the ”Scenario Definition and Initial Threat Analysis” deliverable (D0.1), the lifetime of a security WSN may range from 10 hours to a few days, which is very common for high-risk security events. A few very unusual missions may last for longer periods.

For the Homeland Security WSN prototype, a lifetime of 24 or 48 hours is considered acceptable, since it is enough for most security events.

1.3.7 Security Requirements

Several techniques might be employed by an attacker in order to jeopardize detection and/or tracking by the Homeland Security WSN (HS-WSN). Each of these techniques presents different degrees of complexity, as well as different difficulties in terms of the equipment and technical skills required to apply them. From the point of view of the attacker, it is also important to apply such techniques without being detected, otherwise the interference with the HS-WSN will itself lead to an alert situation and trigger the action of the security teams.

1. To physically damage the sensor nodes. In case the attacker succeeds damaging the HS-WSN devices, she will henceforth be able to penetrate the target area and carry out its mission undetected. Another possible objective is to destroy the data stored locally at the sensor nodes.

2. To change the location and/or direction of the sensors. In case of directional sensors, if the attacker succeeds directing them away from the action spots (e.g., changing the direction of a movement detector), she will henceforth be able to penetrate the target area and carry out her mission undetected. The attacker can also try to change the location of the sensors, so that even if detected, her position will not be correctly estimated, preventing the system to track her across the target area.

3. To jam the HS-WSN radio transmissions. By preventing HS-WSN inter-node communication, the attacker does not allow detecting sensors to send their alert messages to the monitoring platforms, allowing her to carry out her mission undetected. It can constitute a preliminary attack to perform attacks of type 1 or 2 (see above).

4. To replay messages intercepted from the HS-WSN. The attacker may try to penetrate the HS-WSN using compatible devices to transmit extra traffic through the HS-WSN (e.g., dummy status messages) in order to overload network resources, preventing the transmission of critical data like intrusion alerts. This extra traffic is composed of HS-WSN transmissions intercepted by the attacker and hence are valid from the point of view of the protocol message structure.

5. To forge false messages transmitted through the HS-WSN. The attacker may try to penetrate the HS-WSN using compatible devices to transmit extra traffic through the HS-WSN (e.g., dummy status messages) in order to overload network resources, preventing the transmission of critical data like intrusion alerts. Alternatively, the attacker may forge false alarms in order to generate mistrust regarding the HS-WSN system, or to overload the security teams, turning their attention away from spots where the attacker intends to penetrate the target area.

6. To intercept HS-WSN transmissions in order to guess if/when she is being detected. The attacker may try to know if/when she is being detected by passively intercepting and analysing HS-WSN traffic. This will allow her to constantly adapt her actions in order not to be seized by the intervening security teams.

We sum up the major attacks which we have tackled in our homeland security application in Table 6.

Table 6: Attacks in the homeland security scenario

Attack name	**Goal**	**System parts to attack**	**Technical means needed to attack**	**Technical attack description**	**Effort to succeed**	**Attacker type**	**UBI supported countermeasures**	**Effort to succeed after applying UBI countermeasures**
To replay HS-WSN messages	To overload the network or to keep sending ”keep-alive” messages from destroyed sensors	Sensors, network, monitoring platforms	Wireless communication device(s) compatible with HS-WSN protocols	To replay new messages obeying to the protocols in use in the HS-WSN, and hence are interpreted as messages generated by HS-WSN nodes	It requires having at least a node whose PHY and Data Link layers are compatible with those of the target WSN	Knowledgeable Insiders	Inclusions of sequence numbers in the message payload	With sequence numbers, the attack is only possible when the sequence number repeats, which does not happen during a single mission.
To change the location and/or direction of the sensors	To carry out an intrusion without being detected/tracked	Sensors	None	To turn the sensors towards useless directions, and/or to change their location	Assuming that the sensors are well positioned and concealed, this attack can be difficult unless an attack of type 3 is previously carried out	Clever Outsiders	The delivery of alarm messages is guaranteed by the DTSN transport protocol	It is still difficult, since even if the position of a sensor node is changed to a harmless position, the attacker must be careful not to be detected by this or other sensor node in the course of the attack, otherwise the corresponding alarm is surely delivered to the monitoring platform, even if the channel was temporarily jammed to cover the attack

Continued on next page

Table 6 – continued from previous page

Attack name	**Goal**	**System parts to attack**	**Technical means needed to attack**	**Technical attack description**	**Effort to succeed**	**Attacker type**	**UBI supported countermeasures**	**Effort to succeed after applying UBI countermeasures**
To forge false messages transmitted through the HS-WSN	To overload the network, to generate mistrust, to overload the intervening security teams	Sensors, network, monitoring platforms	Wireless communication device(s) compatible with HS-WSN protocols	To forge new messages obeying to the protocols in use in the HS-WSN, and hence are interpreted as messages generated by HS-WSN nodes	It requires having at least a node whose PHY and Data Link layers are compatible with those of the target WSN	Knowledgeable Insiders	Inclusion of message integrity check, obtained using encryption keys. Both symmetric and asymmetric ciphers are supported	It requires a more powerful CPU and knowledge to perform cryptanalysis in order to obtain the encryption keys from the plaintext and integrity check
To jam the HS-WSN radio transmissions	To carry out an intrusion without being detected/tracked	Network communication	High Tx power wireless communication device(s) (PHY layer only). In case the HS-WSN operating frequency is not known, a signal detector is needed	To transmit an high Tx power signal that interferes with HS-WSN communications	To locally jam RF communications using an high Tx power device is not difficult, it just requires knowledge of the WSN operating frequencies. The cost is in the same order of magnitude of a single sensor node. In case signal detection is needed, the cost rises significantly	Clever Outsiders	”Keep-alive” messages sent by sensor nodes, with associated timer at the sink node	Can be carried out, but is always detected, generating alarm. This will trigger the intervention of security teams in the area of the affected sensors.

Continued on next page

Table 6 – continued from previous page

Attack name	**Goal**	**System parts to attack**	**Technical means needed to attack**	**Technical attack description**	**Effort to succeed**	**Attacker type**	**UBI supported countermeasures**	**Effort to succeed after applying UBI countermeasures**
To intercept HS-WSN transmissions in order to guess if/when he is being detected	to constantly adapt action in order not to be seized by security teams	Network communication	Wireless communication device(s) compatible with HS-WSN protocols. A Laptop or PDA can be used, but it is not mandatory (e.g., if the wireless device has output LEDs)	To passively snoop the HS-WSN traffic using compatible wireless devices able to understand HS-WSN protocols, and look for relevant data fields that can reveal the alarm status of the surrounding sensors. A simple version of this attack consists of simply inferring alarms from the amount of detected traffic	It requires having at least a network node plus PDA/Laptop, able to receive and interpret the WSN messages	Knowledgeable Insiders	The messages are encrypted, in a way that the resulting cipher is not the same for the same plaintext in two different messages. Since this requires synchronization between sender and receiver, the sequence numbers are used as Initialization Vectors (IV). Both symmetric and asymmetric ciphers are supported. In order to deny the attacker the possibility of differentiating the ”keep-alive” messages from alarms based on the periodicity of the former, the ”keep-alive” messages shall be sent at irregular intervals, according to a pseudo-random pattern	It requires a more powerful CPU and knowledge to perform cryptanalysis in order to obtain the encryption keys from the plaintext and integrity check. Traffic pattern analysis becomes more difficult and requires long-term statistical analysis of inter-message intervals

Continued on next page

Table 6 – continued from previous page

Attack name	**Goal**	**System parts to attack**	**Technical means needed to attack**	**Technical attack description**	**Effort to succeed**	**Attacker type**	**UBI supported countermeasures**	**Effort to succeed after applying UBI countermeasures**
To physically damage the sensor nodes	To carry out an intrusion without being detected/tracked. To destroy the data stored locally at the sensor nodes	Sensors	At most an hammer or other simple tool, but usually requires destroying the sensor from a distance using a long-reach tool or even a shooting weapon	To eliminate or damage the sensor nodes	Assuming that the sensors are well positioned and concealed, this attack can be difficult unless an attack of type 3 is previously carried out	Clever Outsiders	”Keep-alive” messages sent by sensor nodes, with associated timer at the sink node. TinyDSM supports distributed storage of the events detected by sensor nodes, thus minimizing the chance that they are lost when the nodes are destroyed	Can be carried out, but is always detected, generating alarm. This will trigger the intervention of security teams, and eventual replacement/repositioning of the destroyed nodes

In order to counter these threats, The UbiSec&Sens HS-WSN shall includes the following mechanisms:

1. Issuance of ”Keep-alive” messages from the sensor nodes to the monitoring station. This will allow permanent monitoring of the status of each sensor node, detecting any malfunctions that might arise either due to technical problems (e.g., node naturally running out of energy) or an attack, in particular attacks of type 1, 3 and 4. The failure to receive a ”keep-alive” message during an given interval of time (this is a configuration parameter) will be considered equivalent to an alarm situation and will trigger the intervention of the security teams.

2. Guaranteed delivery of alarm messages based on the DTSN transport protocol. In case an attack of type 2 is temporarily covered by an attack of type 3, it may happen that the attacker is detected by the intrusion detection device before of while she changes its position, but the sensor node will not be able to immediately alert the monitoring station, since the RF channel is being jammed. DTSN will ensure that once the attack of type 3 is finished (the attacker may not want it to last for a long time, otherwise the absence of ”keep-alive messages” will reveal the attack - see above), the sensor node will resume retransmission of the alarm message, which will then reach the monitoring station and trigger the intervention of the security teams.

3. Inclusion of sequence numbers in the message payload. A smart attacker might try to covertly overload the HS-WSN simply by replaying a huge quantity of status messages that would then be transmitted all the way to the monitoring station, leading to the battery depletion of nodes along the used routing paths. While the ”keep-alive” messages allow the detection of the situation after the attack has already produced its effect, the use of sequence will allow immediate detection of the attempt to replay any message.

4. Inclusion of an integrity-check sequence (ICS) within the packet payload (based on wither symmetric or asymmetric cryptography). This functionality is used to authenticate the sensor nodes. Since only the sensor nodes and the monitoring station know the secret keys used either produce and/or validate the ICS, the attacker will have a very low probability of success correct message forging. Consequently, such an attempt will be promptly detected. Although asymmetric presents a higher security level, it is too time-consuming from point of view of the scenario requirements (the alarm delivery latency should stay below 5 seconds), and thus symmetric cryptography will in practice be preferred.

5. Encryption of ”keep-alive” and alarm messages. This functionality will prevent the attacker from correctly interpreting the ”keep-alive” and alarm messages, being then unable to distinguish them. This will deny her the capability to know when she is being detected, buying time to intervention of the security teams in case a detection alarm is issued by a sensor node. Although asymmetric presents a higher security level, it is too time-consuming from point of view of the scenario requirements (the alarm delivery latency should stay below 5 seconds), and thus symmetric cryptography will in practice be preferred.

6. Avoidance of periodicity of message generation. Although the previous functionality denies the attacker the possibility to interpret the HS-WSN messages based on the content, analysis of the channel activity may be enough to identify and distinguish different message types. As such, it is of utmost importance to avoid the periodicity of maintenance messages, like ”keep-alive” messages, by introducing some randomness for the generation inter-packet intervals.

2 Hardware and Software Evaluations

In this chapter we present hardware and software solutions that are currently available for realisation of wireless sensor network applications. We are focussing on existing hardware architectures for which we investigate their energy efficiency with a special focus on public key cryptography, since these are the most power hungry operations a sensor node will have to execute. We also investigate the current operating systems as well as middleware approaches for wireless sensor networks. By that this chapter explores the currently available design space for realisation of WSN applications. We will use our results to define an idealised hardware architecture as well as an optimised middleware solution later in this document. We will also use our findings for the description of an initial soft- and hardware architecture which will be used for our demonstrator set-up.

2.1 Hardware

In this section we will describe and evaluate the off-the-shelf sensor nodes and their components. The sensor nodes will be evaluated with respect to available resources (processing power, memory, sensors, type of radio device) and, after that, with respect to the needed resources (energy).

2.1.1 Criteria

The main issue the designers of Wireless Sensor Networks have to cope with is the limited energy. Of course, this problem depends on the application and specific implementation but if the nodes are not constraint with respect to energy then most of the problems do not exist. Thus, since they are usually powered by batteries, we assume that the nodes in a Wireless Sensor Network have only limited amount of energy available. This implies the need for energy saving mechanisms in order to extend the lifetime of the network and, on the other hand, to reduce the maintenance effort. This becomes even more important if replacing of batteries in the application field is not practicable or even not possible. Thus, here we will focus on nodes based on 8-bit and 16-bit processing units as they currently promise the best energy consumption to processing power ratio. There are also nodes available that are equipped with 32-bit processing units, but they require much more energy what limits their lifetime.

The applicability of a Wireless Sensor Network is strongly dependent on the overall lifetime of the network as a set of nodes. There are many ways to define the lifetime of a network, e.g., first dead node, a fixed percentage of dead nodes, lack of area coverage or partitioning of the network. The kind of definition is application dependent, but in general, the more efficient a single node is with respect to energy consumption the longer the overall lifetime of the network.

In the following subsections we will present the MICA family [17] and the TMote Sky [6] as examples of the off-the-shelf wireless sensor nodes.

2.1.2 Micro Controllers / Processors

Here we will try to evaluate the processing units of the sensor nodes. The members of the MICA family (MICA2DOT, MICA2 and MICAz) use the 8-bit ATmega128L[4] micro controller from ATMEL. The second group includes sensor nodes based on the 16-bit MSP430F1611[22] from Texas Instruments, like TelosB[16] and Tmote Sky[6]. The Tmote Sky node is actually slightly modified TelosB, but these changes do not influence the performance of the node. Thus, the features of these two nodes are the same, unless otherwise noted.

In the first step we evaluate these micro controllers using general information from their specifications. Both ATmega128L and MSP430F1611 can run with maximum clock frequency of 8 MHz (at 3 V supply voltage) and as usual for nowadays micro controllers are equipped with diverse on-chip peripherals. Table 7 presents the comparison of the peripherals available and some parameters of both micro controllers.

In the next step we use the information from the micro controllers' documentations to calculate the overall energy consumption and also the amount of energy consumed per clock cycle. In each case the estimated power consumption is calculated at 3V power supply voltage and at maximum clock frequency specified for the node.

MICA2DOT	ATmega128L at 4 MHz	→ 5.5 mA	→ 16.5 mW	→ 4.125 nJ per clock cycle
MICA2, MICAz	ATmega128L at 7.37 MHz	→ 10 mA	→ 30 mW	→ 4.07 nJ per clock cycle
Tmote Sky	MSP430F1611 at 8 MHz	→ 4.8 mA	→ 14.4 mW	→ 1.8 nJ per clock cycle

Table 7: ATmega128L and MSP430F1611 on chip peripherals and parameters

Peripheral or parameter	ATmega128L	MSP430F1611
word width	8 bit	16 bit
voltage range	2.7 - 5.5 V	1.8 - 3.6 V
voltage range (flash programming)	2.7 - 5.5 V	2.7 - 3.6 V
program memory	128 kB flash	48 kB flash
data memory	4 kB RAM 4 kB EEPROM	10 kB RAM 256 B flash
AD converter	8 channel 10 bit	8 channel 12 bit
Hardware multiplier	Yes	Yes
External memory interface	Yes (64 kB)	No
Serial interfaces	2 USART, SPI, I2C	2 USART (2 SPI, I2C)

The performance ratio between MICA2DOT and MICA2 or MICAz (MICAx) can be estimated easily since both use the ATmega. The amount of clock cycles will not change for a calculation and the only difference will be the time needed to perform it. Thus, the performance ratio between nodes belonging to the MICA family is equal to the clock frequency ratio, i.e., pure speed ratio. If we had used the performance figures of the MICA2DOT as one unit, then the performance of MICAx would be about 1,85. However, this increased performance results in increased energy consumption, what actually results in roughly the same energy costs of a calculation on all nodes from the MICA family.

The comparison of energy consumed per clock cycle on both MSP430 and ATmega shows that the MSP430 requires only about 44% of the energy consumed by ATmega running at about the same clock frequency. However, the question is what is the performance ratio between these two micro controllers. In this case the estimation is not that straight-forward as for the nodes of the MICA family because the MSP430 operates on 16-bit words and ATmega on 8-bit words. That is the reason why these numbers only show the needed amount of energy and do not really compare the computing power of each processing unit.

We will estimate the performance ratio between these micro controllers using public key cryptography calculations. The reason for this choice of evaluation method is twofold. On the one hand, our project is security related and on the other hand the public key cryptography operations are the most demanding ones. As the measurement data we use measurements from [11]. In this paper the authors present the time needed by TelosB and MICA nodes to perform the server side handshake step of the secure SSL/TLS communication. These measurements were recorded for two kinds of underlying cryptosystem, i.e., for RSA and for ECC based handshake. In order to make the results independent from the type of radio device, table 8 presents the time needed for the calculation only.

We will further normalise the computational performance of all the nodes using the result of the worst one. Combining the ratio with the previously presented power consumption of each node we estimate the energy needed by these public key cryptography operations on each sensor node.

The modulo exponentiation with the big private exponent is the main and most expensive part of the full RSA-1024 handshake. The complete handshake needs about 22 seconds on MICA2DOT, 12 seconds on MICAx and about 5.7 seconds on TelosB sensor node.

In the case of full ECC-160 handshake, where the main and most expensive operation is the scalar point multiplication, the time needed was 1.6 second on MICA2DOT, 0.87 second on MICAx and 0.5 second on TelosB.

Based on the measurements for the ECC handshake the computing performance of the TelosB is about 3.2 compared to the performance of the MICA2DOT. The TelosB is also about 1.75 times faster than the MICAx nodes. This is the advantage of the 16-bit processing unit of the TelosB.

Knowing the time needed by each type of node we estimate the power consumed by the nodes while calculating the above mentioned operations (see Table 9). Based on these results we create another factor, the power consumption ratio—the power consumed by the cryptographic operations normalised using the power

Table 8: Time needed by the sensor nodes to perform SSL/TLS handshake

Sensor node	RSA-1024 handshake	Performance ratio (RSA)
MICA2DOT	22.00 s	1.00
MICA2/MICAz	12.00 s	1.83
TelosB	5.70 s	3.86

Sensor node	ECC-160 handshake	Performance ratio (ECC)
MICA2DOT	1.60 s	1.00
MICA2/MICAz	0.87 s	1.85
TelosB	0.50 s	3.20

Table 9: Power consumed by the sensor nodes to perform SSL/TLS handshake

Sensor node	RSA-1024 handshake	Power consumption ratio (RSA)
MICA2DOT	363.00 mJ	1.00
MICA2/MICAz	360.00 mJ	0.99
TelosB	82.10 mJ	0.23

Sensor node	ECC-160 handshake	Power consumption ratio (ECC)
MICA2DOT	26.40 mJ	1.00
MICA2/MICAz	26.10 mJ	0.99
TelosB	7.20 mJ	0.27

consumed by the least effective node.

Since the clock cost is almost the same for all nodes of the MICA family the power consumption will also be the same. What is interesting, the TelosB node equipped with the MSP430 micro controller needs only 23–27% of the power consumed by the ATmega based MICA nodes performing the same operation.

Additional information needed to compare both micro controllers is the current consumed in power safe modes. These modes are used in case there are no calculation tasks for the controller and it is waiting for external or internal interrupts. Both ATmega128L and MSP430F1611 support several power save modes, but the most interesting are those, where the power consumption is minimised but the interrupts and internal peripherals like the watchdog timer are still enabled. Table 10 presents the current consumed by the micro controllers in comparable power save modes.

2.1.3 Size of Memory

As already mentioned the ATmega128L is equipped with 128 kB flash code memory, 4 kB EEPROM data memory and 4 kB RAM. The MSP430F1611 has 48 kB flash code memory, 256 bytes flash data memory and 10 kB RAM. This shows that the ATmega128L provides more code memory, more nonvolatile data memory, but less RAM. But the ATmega micro controller can use up to 64 kB external RAM when needed. However, the external RAM imposes additional power consumption. In case of the MSP430, the RAM memory cannot

Table 10: ATmega128L and MSP430F1611 current consumption in power save modes

Power save mode	ATmega128L	MSP430F1611
CPU OFF, peripherals at 4 MHz	max 2.5 mA	max 0.4 mA
CPU OFF, peripherals OFF WDT ON, interrupts ON	15 μA	5 μA

Table 11: AT45DB041B and M25P80 flash memory parameters

Parameter	AT45DB041B	M25P80
operating voltage	2.7 - 3.6 V (2.5 - 3.6 V)	2.7 - 3.6 V
standby current	max 10 μA typ 2 μA	max 50 μA
deep power down current		max 10 μA
WRITE current	max 35 mA typ 15 mA	max 15 mA
READ current (at 20 MHz SPI)	max 10 mA typ 4 mA	max 4 mA

Table 12: Current and power consumption of the ZigBee transceiver CC2420. Power consumption calculated at 3V supply voltage. Power consumption per bit at transmission speed of 250 kbit/s

Type of transmission	Current [mA]	Power [mW]	Power per bit [μWs/bit]
RX	18.8	56.4	0.226
TX -25 dBm	8.5	25.5	0.102
TX -15 dBm	9.9	29.7	0.119
TX -10 dBm	11.0	33.0	0.132
TX -5 dBm	14.0	42.0	0.168
TX 0 dBm	17.4	52.2	0.209

be extended.

Both micro controllers can use an external serial flash data memory. Such memory chips with SPI interface are used on all sensor nodes. The nodes from the MICA family use external 4 Mbit (512 kB) flash chip AT45DB041B and the Tmote Sky uses 8 Mbit (1 MB) M25P80 chip. Table 11 provides some information about their parameters. It shows that the M25P80 chip consumes more energy in the standby mode, but needs much less in active modes. However, the M25P80 supports deep power down mode, that reduces its power consumption and requires only one instruction to enter and one to leave the mode. The advantage of the AT45DB041B flash is that it is also available in a 2.5 V version.

2.1.4 Radio

All four types of sensor nodes use single chip transceivers. MICA2 and MICA2DOT use 433 MHz or 868 MHz radio chip CC1000 [21] and MICAz and TelosB use ZigBee 2.4 GHz radio chip CC2420 [20], both from Chipcon (now part of Texas Instruments). The two radio types differ in performance. ZigBee devices transmit data with 250 kbit/s data rate with maximum power of 0 dBm and CC1000 chip allows data rates up to 76.8 kbit/s with maximum power of 10 dBm (433 MHz) or 5 dBm (868 MHz). The MICA nodes that use the cc1000 chip use Manchester encoding reducing the maximum transmission rate to 38.4 kbit/s.

The power consumption data for both chips are shown in Table 12 and Table 13. This data shows that the higher power consumption of cc2420 is compensated by the lower cost of per bit transmission.

The cc1000 transceiver uses 3-wire configuration interface and 2-wire data interface. It also provides analog RSSI signal that can be connected to one of the micro controllers ADC inputs. The cc2420 chip uses SPI compatible 4-wire interface for configuration and data. Additionally it provides digital signals for clear channel assessment (CCA) two input and output FIFO interface signals (FIFO and FIFOP) and timing signal SFD. The RSSI and LQI values can be accessed over the SPI interface, i.e., by reading internal registers.

The standard SPI interface used by the cc2420 transceiver reduces the programming effort needed to interface the chip.

Table 13: Power consumption of the 433 MHz and 868 MHz transceiver CC1000. Power consumption calculated at 3V supply voltage. Power consumption per bit at transmission speed of 38.4 kbit/s

Type of transmission	Current [mA]	Power [mW]	Power per bit [μWs/bit]
433 MHz			
RX	7.4	22.2	0.578
TX -20 dBm	5.3	15.9	0.414
TX -5 dBm	8.9	26.7	0.696
TX 0 dBm	10.4	31.2	0.812
TX 5 dBm	14.8	44.6	1.160
TX 10 dBm	26.7	80.1	2.086
868 MHz			
RX	9.6	28.8	0.750
TX -20 dBm	8.6	25.8	0.672
TX -5 dBm	13.8	41.4	1.078
TX 0 dBm	16.5	49.5	1.290
TX 5 dBm	25.4	76.2	1.984
TX 10 dBm	——	——	——

2.1.5 Connection to Sensors / AD Converter

There are several options how to connect sensors to the considered micro controllers. Depending on the kind of sensor the data transmission may be digital (connected to SPI or I2C interface) or analog (via ADC).

The considered sensor nodes come per default without any sensors. The Tmote Sky node can be ordered with a sensor suite, i.e., temperature, relative humidity and light sensors. These are mounted directly on the sensor node board. The temperature and humidity sensor SHT11 (or SHT15) [34] is a digital one, the light sensors are two photodiodes connected to the ADC inputs of the micro controller. The SHT11 delivers 12 bit data, requires supply voltage between 2.4 and 5.5 V and consumes about 550 μA during measuring and maximum 1 μA in sleep mode.

The sensor equipment for the MICA family is more flexible, i.e., there are external sensor boards that can be connected to the main sensor node. For instance, the MTS300 board [15] provides temperature (thermistor), light (photoresistor) and acoustic sensors. The latter is a microphone with preamplifier with digitally controlled gain. The components used for the acoustic sensor require a minimum operating voltage of 2.4 V (the MAX4466 preamlifier) and 2.7 V (the digital potentiometer AD5242), the consumed current is about 25 μA in active mode. The current flow in the thermistor and photoresistor circuits is about 270 μA assuming 3 V supply voltage while active (the resistance of the active element together with an additional resistor is about 11 Kohm). All these sensors are connected to the ADC inputs of the micro controller.

2.2 Software Architecture

2.2.1 Selecting the Operating System

Aiming at creating flexible network architecture the choice of an appropriate operating system is essential. In this section we summarise three existing mainstream operating systems (OS) for wireless sensor networks. For the analysis we chose TinyOS [14] as de-facto standard operating system supplied with sensor nodes and two operating systems which outlive their experimental phase: MANTIS OS [2] and Contiki [7]. TinyOS developed at the University of California, Berkeley is a completely event-driven system. On contrary, the second operating system represents a class of multi-threaded operating systems, like those we all experience in all current computers. The design of Contiki combines the two previous paradigms. We present an overview of these operating systems focusing on the richness of provided functionality and flexibility of the application development process.

There are two general approaches that are used in the operating systems design: The event-driven and multi-threaded approaches. The major idea behind a purely event-driven system is as follows. The execution of a certain task is implemented as a handler of either an internal (a request from an internal event scheduler) or external (HW interrupt) event. Once a particular event handler is called, the task's code is executed until it is completed. On contrary, in the threaded approach the execution of a task can be interrupted and the processor resources are reallocated to another task. In this way it is the task of the kernel to keep the execution of different programs consistent.

Below we present the results of the cross comparison of the three operating systems mentioned above. In this document we mainly highlight the advantages and limitations of the systems and do not present the detailed description of the OS's. Therefore, for these details we refer the reader to the corresponding system specifications. We present the comparison against the following factors.

- The underlying OS design paradigm;
- The degree of usage in the WSN community;
- The type of license for the development and modifications to the core functionality;
- OS structure and flexibility of code updates;
- The programming concept for writing applications;
- Potential ability for integration with general purpose network simulators.

The conclusions below were drawn based on our extensive experience of programming under TinyOS and preliminary practical analysis of Contiki OS. We did not perform practical investigations of MANTIS by the reason described below in the text. The conclusions about this system are the result of an extensive study of the available documentation.

2.2.1.1 TinyOS

TinyOS is an operating system designed specifically for wireless sensor nodes at the University of California, Berkeley. Below we list the highlights of TinyOS.

- TinyOS is a component based operating system that uses the event-driven design paradigm. In addition to this TinyOS implements a concurrency model which allows for two distinct execution modes: asynchronous and synchronous executions. In the synchronous mode a computational task once scheduled runs until the completion. In asynchronous mode a running task can be interrupted by an external HW interrupt. In this case the CPU resources are given to the interrupt handler code. Note that in TinyOS there is no dynamic context switching. This means that the programmer has to protect the critical variables manually ("atomic" declaration) when there is a risk of its modification during the asynchronous execution mode.
- TinyOS is de-facto standard operating system supplied with commonly used for WSN research Berkeley motes.
- A complete binary image of TinyOS kernel together with all applications is built during the compilation. When a sensor node needs another functionality, which is not present in the original image, another complete image should be downloaded to the node. Normally a sensor node keeps several binaries with different functionality stored in the re-writable flash memory.
- TinyOS specifies own extension to the standard C, called NesC. All applications are written in NesC. Upon compilation the NesC code first translates to ANSI C and the resulting intermediate file is compiled to the binary image.
- TinyOS is supplied with own simulation facility, named TOSSIM. It is a tool that is primarily used for debugging the TinyOS functionality. It may also be used for simple networking simulations. However, TOSSIM is not a general purpose network simulator, therefore, complex simulations with heterogeneous and sophisticated network settings and scenarios are not possible.

The major advantage of Tiny OS is the minimal code size amongst all three considered systems. The event-driven nature of the OS is proven to be efficient for a large class of WSN applications. However, the system has a number of serious limitations which makes its usage in the scope of UbiSec&Sens project questionable. Note, that we intentionally do not talk about drawbacks of the system but about its limitations for the UbiSec&Sens project.

The major limitation of a purely event-driven OS is a known problem of application blocking because of the execution of a time consuming code. This problem is especially critical when a sensor node performs crypto operations. The experience described in [2] demonstrates that such operations as well as complex processing with float type numbers may result in the buffer overflow problem for the packets waiting for transmission. This problem can be relaxed to some extent by advanced programming skills of the developer, however this might also be problematic for security related applications as we describe below. As in the UbiSec&Sens project security is the key aspect in every work package the above mentioned problem places serious limitation on usage of TinyOS as a ”working horse” operating system.

Regarding the code size, which as it is mentioned above is the smallest of the three systems, the fact that several complete images with different functionalities should be stored inside the sensor node seriously relaxes this advantage. In order to ensure the level of network flexibility specified in the UbiSec&Sens project having several images inside the sensor nodes is unavoidable. As a result potentially large amount of memory, which otherwise would be used for storing the measured information would be consumed for the system purposes. From the point of view of the code distribution for update purposes, broadcasting the entire system image is not efficient from the bandwidth and energy consumption prospective. There exists a solution [24] which allows modular updates of running applications. Maté is virtual machine implemented in TinyOS. A Mate application is a byte-code which is transmitted over the network and interpreted by the virtual machine on each sensor node.

2.2.1.2 MANTIS

MANTIS is an operating system designed at the University of Colorado using an opposite to TinyOS paradigm. MANTIS is a purely threaded operating system. Below we list the highlights of MANTIS.

- MANTIS is an operating system that uses time-sliced multi-threading design paradigm. A running task in this system may be interrupted during the execution and the control is moved to a concurrent task. When interrupted a run-time context of the task is saved and afterwards restored upon regaining the CPU resources.

- MANTIS is currently a completed product with implementations available for Mica motes and the development environment for major operating systems.

- Permission to use, copy, modify, and distribute MANTIS is regulated by the eCOS-style license that allows developers to keep their own non-OS code (e.g., applications, drivers), but requires modifications to the core MANTIS to be given back to the project.

- MANTIS OS have a structure of a general purpose operating system. It consists of a kernel with common to all applications functionalities, the device drivers, and a set of applications which run as concurrent threads. The operating system allows reprogramming (updating the code) on different levels of granularity. In the extreme cases either an entire binary image can be updated or a particular thread can be reprogrammed. The dynamic reprogramming capability is implemented as a system call library. Each application may write a modified code to this library through system calls.

- MANTIS uses the standard ANSI C for writing the kernel and applications. This allows for more convenient application development process and high level of portability between different general purpose operating systems.

- MANTIS is supplied with own development tool chain, which include diverse simulation and debugging facilities. It is possible, for example, to perform heterogeneous experiments with virtual nodes running as processes on stationary PCs and real nodes with running MANTIS OS. However, the issue of integration with a general purpose network simulator is not addressed. Nevertheless, taking into account the underlying C programming language this task seems feasible.

The major advantage of the operating system is the eliminated problem of application blocking during the execution of a computation-expensive part of the code. For the implementation of threads a conventional multithreading approach is used. Namely, once a task is interrupted its run-time context is saved in the RAM memory. Upon regaining the CPU resources the context is restored. In MANTIS the context of single thread consumes 128 B of memory. In this way MANTIS supports up to several tens of concurrent threads on a sensor node with 4 kB of RAM (Mica2 motes).

2.2.1.3 Contiki

Contiki is an operating system developed at SICS, Swedish Institute of Computer Science. It is an event-driven operating system with a possibility of multithreading. Below we list the highlights of Contiki.

- In an original way Contiki combines the advantages of the event-driven and multi-threading OS design paradigms. The kernel is organised as an event scheduler that passes the CPU control to multiple concurrent threads. In contrast to the time-sliced approach the control is passed between the processes by submitting an event to the event list of the scheduler.
- Contiki is a relatively young operating system. Developed originally for ”ancient” Commodore 64 platform it was the only operating system with full IP networking capabilities for this type of computers. Currently the ports of Contiki exist for all commonly used research sensor platforms. The system is supplied with the development tool chain for Linux and Cygwin environments.
- Contiki OS has a structure of an usual personal computer OS. It consists of a compact kernel with device drivers, common libraries and a set of applications. The operating system allows reprogramming (updating the code) on different levels of granularity. It is for example possible to update the entire binary image, specific drivers, and service libraries. A particular application or part of the OS can be dynamically replaced over the wireless network interface. The code is distributed as binary executable files. Upon reception, the code is dynamically linked, initialised and launched by the operating system.
- Contiki uses the standard ANSI C for writing the kernel and applications. This allows for more convenient application development process and high level of portability between different general purpose operating systems.
- Contiki is supplied with own simulation facility. The standard development tool chain for a particular sensor platform is used for the development and debugging. In the case of Telos motes based on TI MSP 430 microcontroller a MSP specific gcc compiler and debugger are used. As it is the case with other sensor OS, the issue of integration with a general purpose network simulator is not addressed in Contiki. Nevertheless, taking into account the underlying C programming language potentially this task seems feasible.

In comparison to the previously considered operating systems the size of Contiki’s kernel is larger than in TinyOS but smaller than in MANTIS. Despite of slightly larger size than TinyOS, Contiki has a number of functional advantages over the former. The major one is a flexible combination of the event-driven kernel and the multithreaded library. In the multi-threaded mode each thread requires a separate stack. As in MANTIS the size of the thread’s context in RAM is 128 B. In comparison to MANTIS, the less restrictive license type allows for a contribution-oriented experimentation with the core functionality of the operating system.

Applications in Contiki are compiled independently of the kernel. The resulting executable binary can be uploaded to the sensor nodes over the network. The size of the transmitted code is much smaller than that of the kernel. This functionality makes Contiki very suitable for a wireless sensor network with diverse target application areas.

The networking functionality in Contiki is presented by a highly optimised and compact implementation of the entire TCP/IP protocol stack included as a part of the kernel. This issue can be considered both as an advantage and to a certain degree as a limitation of the system. On one hand, it is nice to have a working communication stack out of the box and be able to communicate with the sensor nodes using conventional network protocols. On the other hand, we believe that the full TCP/IP stack is not always needed in wireless sensor networks and therefore should be included in the architecture as an optional functional component.

A clear advantage of the system is the usage of the standard ANSI C for programming the kernel and applications. This makes Contiki potentially suitable for easier integration with a general purpose network simulator.

2.2.1.4 Summary

There are numbers of existing operating systems which appeared to be outside the scope for this study. Amongst the most interesting solutions we highlight JavaOS [40] from SUN Microsystems, FreeRTOS [39] and SOS [41].

In the context of wireless sensor networks there are pros and cons of using each of the above reviewd operating systems. For example, the major advantage of the event-driven OS is low memory consumption during its execution. However, the major disadvantage of such OS is a potential for blocking the execution of other tasks during servicing the time-consuming operations, e.g., cryptographic operations. On the other hand, the major advantage of purely thread-based OS is a concurrent execution of multiple processes. However, amongst the disadvantages of this approach the most critical one is RAM memory consumption during the context switching.

2.3 Middleware Approaches

Wireless sensor networks are mainly used to gather data about a certain environment (see for example [30]). This especially holds true for the application fields selected in this project. Due to this focus the research in the middleware area has somewhat concentrated on supporting data storage and retrieval issues in WSNs. We reviewed approaches such as tinyDB [25], Cougar [44] and Hood [42] to name just a few in milestone M3.1 to which we refer for detailed information.

There are several approaches towards flexible middleware for wireless sensor networks. They try to provide application independent support to applications but are mainly focussing on communication issues in one form or another. In [45] authors introduce the concept of reconfigurability for middleware in pervasive computing. Here the major part (if not all components of the middleware) is located on a PDA device and the task of the middleware is merely the discovery provision of available data. In [37] the authors propose an application independent scheme for defining groups of sensors to provide easy adaptability of a WSN to new applications. Here part of the adaptation logic is placed on the sensor nodes. A similar approach exploiting roles of sensor node is proposed in [23].

Our middleware approach differs from those cited above by that we are focusing on a very specific functionality i.e., security instead of trying to provide a communication or programming abstraction. In our approach flexibility is addressing support of a wide range of application and by individual support of the security needs of each application. I.e., we are trying to provide a tailor made security solution for each application. In order to achieve this goal we are working towards a middleware compiler which selects security modules based on application and sensor node requirements and constraints respectively. In this area some work has been done, but none focuses on WSN and security issues but aims at a similar goal i.e., providing tool support for development of a certain middleware. Most approaches are based on model driven architecture (MDA) [3]. The tool sets Cadena [13], VEST [36] and CoSMIC [1] are MDA based and try to support development of platforms for embedded systems. By that, they provide functionality similar to our approach, but the difference is that we focus on security and do not use MDA but defined our platform architecture independently of any formal model. In addition only VEST supports the modelling of security aspects.

In [8] the authors discuss the integration of security aspects into a formal method based development of networked embedded systems. The focus of the security analysis language (SAL) is merely on information flow between networked entities. By that, it might be a way to model security requirements of applications residing on top of the UbiSec&Sens modules and to verify whether or not our middleware compiler selected the correct security modules.

	Tmote Sky (in kBytes)	MICA (in kBytes)
ROM program memory	48	128
RAM for variables	10	4
External Flash for data storage	1024	512

Table 14: Available memory in popular sensor nodes.

3 Requirements Analysis

In this chapter we discuss to which extent the existing system architectures composed of hardware and software can fulfil the requirements defined by our scenarios. Here we are again focusing on energy issues and use memory as a second parameter. The latter clearly shows that the software packages that are installed at the sensor nodes need to be as small as possible and that there is a very strong need to be able to exchange parts of the software configuration, since the deployment of all security modules is infeasible. The available energy is pretty limited as well, allowing only small duty cycles.

3.1 Software Parameters

The design of software for wireless sensor devices is a challenging task that must be done under strong consideration of the hardware constraints. Table 14 summarises the available memory resources for the two sensor node families mentioned in section 2.

The extremely limited computational resources of wireless sensor nodes place hard requirements on the software. It is important to note that the operating system itself consumes a substantial part of the node's memory. Table 15 shows that about 50% of MICA node RAM memory is already consumed by the operating system.

Table 16 shows the memory footprint for the software modules grouped by categories where UbiSec&Sens toolbox will provide a contribution. The data in the table are given for sensor motes running TinyOS operating system.

3.2 Evaluation of Hardware with Respect to Scenarios' requirements

The main requirement that has to be satisfied by the hardware is the lifetime of the sensor nodes. Table 17 presents the theoretical lifetime of each node assuming that in the active mode the micro controller is working all the time and the radio chip is transmitting 20% of the active time with 0 dBm output power. In the non active mode the micro controller is in power save mode and radio chip is switched off. The Table shows also the lifetime of each node for duty cycle different than 100%. The available amount of energy is taken from [29].

It must be emphasised that the comparison here is only duty cycle based, i.e., the actual amount of computations each node is able to process in these diverse duty cycle settings differ as mentioned in section 2.1.2.

Anyway, Table 17 shows that for the agriculture scenario the duty cycle cannot be much more than 1 % to reach the requested lifetime of 5 months. In the homeland security scenario if the requested lifetime is about two days, all nodes manage this lifetime even with 100 % duty cycle. But if the requested lifetime shall be

Functionality	Memory footprint	
	RAM(Bytes)	ROM(Bytes)
Basic Send/Receive over radio interface functionality	86	2988
Processor specific management functionality	18	1876
Operating system specific	2000	4236
Total:	2104	9000

Table 15: Memory occupied by operating system software on a node from the MICA family.

Table 16: Memory footprint for selected software pieces

SW Category	Software item	RAM, Bytes	ROM, Bytes	Comments
Application	TinyPeds (excl. dependencies)	500	1000	
Reliable Transport	DTSN	N/A	6459	Configuration: maximum number of supported flows, and tx/rx window sizes + caching window
Networking	TinyLUNAR	1738	3400	1783B is the memory consumed by interface supporting functionality. The implementation of Tiny Lunar is under intensive development. The memory footprint may differ considerably in the future.
	TinyAODV (Intel)	337	2750	
	DSDV (INOV)	N/A	1123	
	PathDCS	N/A	1764	Routing protocol for data centric storage
	MintRoute	N/A	1400	
	BVR	N/A	1411	Beacon vector routing
Supporting software	TinyRNG	463	10532	Random number generator [10] compiled for Mica2 motes
	ElGamal over ECC	700	4400	Security modules for privacy homomorphism

several days or weeks there must be a kind of duty cycle management in order to extend the lifetime. The same holds for the vehicular scenario, where the lifetime shall be as long as possible.

Nevertheless, if the required lifetime exceeds the manageable lifetime with a reasonable duty cycle, i.e., the duty cycle is not enough to perform all needed calculations, additional energy sources, like solar cells, might be used. This holds especially for outdoor scenarios.

Table 17: Lifetime of the evaluated nodes considering diverse duty cycle settings

Duty cycle [%]	Tmote Sky	MICAz	MICA2	MICA2DOT
	Lifetime in days			
100	3.15	1.93	1.96	1.93
80	3.93	2.41	2.45	2.41
50	6.28	3.86	3.91	3.85
20	15.69	9.62	9.74	9.57
10	31.38	19.15	19.39	19.00
5	62.50	37.92	38.30	37.45
1	300.50	173.61	177.56	164.28

4 Hardware & Software Specification

In this section we discuss the architecture of systems that are build using the UbiSec&Sens toolbox. In the first subsection we develop an idealised sensor node hardware. The major idea is to show which parameter settings would provide easy development of WSN applications allowing for strong security means and long lifetime as well as supreme sensing and storage capabilities. Thereafter, we shortly explain which hardware platforms will be used for the planned demonstrators and how they differ from the idealised sensor nodes. In the systems' architecture subsection we first introduce the different parties that are interacting in a WSN application, as well as the basic network architecture. The major part of the subsection is devoted to the idealise middleware architecture. We will discuss this architecture from different views showing special aspects with respect to life cycle of the overall system and to roles of involved devices.

4.1 Hardware specification

In this section we will discuss the hardware that would satisfy the requirements for a WSN the most. This discussion will have two parts, first, a sensor node that can be built right now using components available on the market, and second, an idealised node with custom made ultra optimised chips that may become truth in several years. Additionally, we will discuss the application of hardware accelerators for the most energy consuming operations, e.g., symmetric cryptography. This falls somewhere between the up to date sensor node approach and the futuristic one, because there are already some hardware accelerators available, but their application still requires additional effort in custom chip design.

4.1.1 The Optimal Node Configurations

The combination of on-the-shelf components can lead to two possible semi-optimal node configurations depending on the available energy and required lifetime. They are both semi-optimal because each solution has some drawback.

The first configuration is a powerful node with great processing power and huge memories. An on-the-shelf example of such a node is the Imote2 [18] from CrossBow. This node is based on the Intel XScale PXA271 32-bit processor usually used in Personal Digital Assistant (PDA) devices. According to the documentation this node has 256 kB Static RAM, 32 MB DRAM and 32 MB flash memory. The clock frequency is between 13 and 416 MHz what makes it actually a small computer. The radio chip is the cc2420 chip known from MICAz and Tmote Sky / TelosB nodes. Powered by three AAA battery cells, this node consumes about 390 μA in deep sleep mode (i.e., even memory clock is stopped) and about 44 mA in active mode (at 13 MHz) with radio on, what reduces its lifetime if additional energy sources like solar cells are not used.

Obviously, such a node is not the target platform for an application that shall run for several weeks or months, but anyway it may be a potential platform for short period computation hungry applications, e.g., like our homeland security scenario. Assuming the node is powered by the AA batteries like MICAx and Tmote Sky nodes are, and the available energy is calculated for the same conditions, this node's lifetime is about two months if in deep sleep mode only. On the other hand, if in active mode only, the lifetime is about 14 hours. The node is powered by three AAA battery cells, what actually causes that the amount of available energy is even smaller.

That was an extreme example of an on-the-shelf powerful sensor node. The high energy consumption is the reason we do not consider this kind of hardware any further. The following examples will be more theoretical hardware configurations with gravity point at either computational power or lifetime of the node.

In order to create a sensor node that may run for a long time there is a need to identify components that require the least energy. Of course it will be appreciated if the performance of these would also be quite good. Section 2 shows that the 16 bit MSP430 micro controller is quite powerful and energy efficient as well. Comparing to the 8-bit ATmega it is the best choice for the sensor node micro controller. The only disadvantage is quite small code flash memory.

There are other micro controllers on the market as well, but none of them reaches the energy efficiency of the MSP430. There is for instance a more energy efficient version of the Atmel micro controller than the one used in the MICA nodes—the ATmega1281 [5]. This micro controller is a little bit less energy efficient than

Table 18: Parameters of the STR711FR2 32 bit micro controller

Parameter	Value
operating voltage	3 - 3.6 V
active current at 16 MHz (all peripherals ON)	27 mA (from flash) 23 mA (from RAM)
active current at 16 MHz (all peripherals OFF)	21 mA (from flash) 16 mA (from RAM)
low power waiting for interrupt current (CPU clock OFF, peripherals at slow CLK)	37 μA
stop current (CPU clock OFF, peripherals clock OFF)	18 μA (RTC ON) 10 μA (RTC OFF)
standby current	10 μA

the MSP430, offers more code space and external memory interface, accepts voltage down to 1.8 V, but is still 8 bit, thus its computational power is comparable to the one provided by its older brother (ATmega128L).

Another component that fits into the requirements is the radio transceiver from Nordic Semi nRF24L01 [33]. According to its documentation the chip requires about 12 mA in transmit and receive mode, what is much less than the cc2420 for the same output power at the same supply voltage (TX: 17.4 mA, RX: 18.8 mA)(more about cc2420 in Table 12). More, the nRF24L01 transceiver manages the transmission rate of 2 Mbits per second what makes the energy cost per bit even lower. However, the protocol is not ZigBee compliant and the cc2420 offers an on board AES hardware accelerator for the transmitted data.

Another part is the external data flash memory. Here one can go in the direction of reducing the current consumption and choose smaller one, like for instance the 25AA1024 [19] 128 kB SPI EEPROM chip from Microchip, or choose a bigger one, like the M25P80 or AT45DB041B flash as in Tmote Sky or MICA nodes that need only a little bit more (about twice) energy while reading or writing and need supply voltage higher than 2.5 or 2.7 V but provide more space to store the collected data.

If the computational power of the MSP430 micro controller is not enough and the design of the sensor node application allows higher energy consumption then a 32 bit micro controller may be chosen. But instead of using the XScale processor we propose applying an ARM based micro controller. A perfect example could for instance be the single voltage STR711FR2 [38] 32 bit micro controller from STMicroelectronics. It has 256 kB code flash, 64 kB RAM, 16 kB data flash on board and external memory interface as well. Multiple communication interfaces simplify the connection to sensors etc. But what is important, it may run up to 60 MHz delivering reasonable computing power but also consumes reasonable amount of energy (see Table 18). This micro controller running from flash at 16 MHz consumes about three times that much energy as the ATmega128L but offers probably much more than three times the computation power. But the very important thing is that in the low power mode and waiting for external or internal wake-up sources the ST711FR2 needs only 18 μA or 37 μA current, respectively. This is 10 (or 20) times less compared to the processing unit of the Imote2 node, what results in 10 (or 20) times longer lifetime in sleep mode. In standby mode the current consumption goes even lower, but in this case the unit is reseted after wake-up.

The disadvantage here is the need for a quite high supply voltage, i.e., 3 V. To cope with this requirement, either more than two 1.5 V batteries or a more sophisticated power supply approach is needed. An example here could be a step-up/step-down DC/DC converter that delivers 3 V from input voltage e.g., between 1.8 V and 5.5 V. In this case, however, the additional energy costs have to be considered. That is why the accepted operating voltage for micro controller and other components is that important.

Adding some application specific sensors to the node accomplishes the design of an energy efficient node built from on the shelf components. The choice here is to use simple analog components like thermistors, photodiodes and photoresistors or more sophisticated and maybe more accurate digital sensors. The advantage of the latter is that they deliver the data in digital form, thus problems with conversion, normalisation and linearity do not exist.

Table 19: Comparison of $GF(2^m)$ ECPM hardware designs.

Ref	Field	Platform	Time	Size
IHP	GF(2^{163})	0.25 μm ASIC	0.08	1.0mm^2, 35Kgates
IHP	GF(2^{163})	Xilinx XC2VP70	0.11	5598 LUTs
[32]	GF(2^{163})	0.13μm ASIC	0.19	117.5 Kgates
[12]	GF(2^{163})	Xilinx XCV2000E	0.14	19508 LUTs
[27]	GF(2^{167})	Xilinx XCV400E	0.21	3002 LUTs
[31]	GF(2^{191})	Xilinx XCV3200E	0.06	≈30000 LUTs
[35]	GF(2^{163})	Xilinx XCV2000E	0.05	25763 LUTs
[43]	GF(2^{191})	0.35 μm ASIC	6.21	1.31mm^2
[9]	GF(2^{233})	0.13 μm ASIC	6.68	71 Kgates

4.1.2 Idealised Hardware

If we think of an ideal sensor node and the availability of components does not really matter then the previous subsection gives an overview of its desired features. It should be as computationally powerful as possible and should need almost no energy, or even harvest the energy it needs. An additional advantage might be the small size of the complete node, possibly a single chip solution.

One possibility to achieve this may be application of highly efficient hardware accelerators. There are many fields where they can be applied, from the most energy hungry operations like public key cryptography to the network protocol layers. Since pure hardware solutions may be a kind of non flexible it should be combined with a piece of software, however, its processing should be very efficient.

The next subsections gives more details on the available and future solutions in hardware accelerators for public key cryptography operations and on potential approaches of harvesting energy.

4.1.3 Hardware Crypto Accelerators

The existence of hardware accelerators for cryptographic primitives would help to relax energy constraints and thus allow for better security at low lifetime penalty. In this subsection we are discussing some hardware accelerators for elliptic curve cryptography.

Optimisations of implementations and applied algorithms turn many cryptographic algorithms more and more suitable for wireless sensor networks. However, if we consider implementations of elliptic curve cryptography for WSNs it still takes hundreds of milliseconds to complete a cryptographic operation. Even if time consumption can be tolerated, the required energy is a serious problem. Every millisecond the node is calculating is a millisecond that brings it closer to death. Hardware accelerators are considerable means to reduce time and this way also the energy required for cryptographic operations. If, as described in [28] a 163 bit ECC operation merely requires 13μJ, it corresponds to a node computation time of less than 1 millisecond. It is a value that is negligible in comparison to other operations or even the needed transmission power.

In the literature several hardware accelerators for ECC have been proposed (see Table 19). However, most of them are focused either on absolute best speed or minimum gate area. None of both approaches yields to best results regarding the requirements of WSNs. The fast implementations [31, 35, 12] need a high amount of area so that it foils the idea of small cheap devices. In contrast, the small units [43, 27] usually need so much time that the total energy (i.e., power x time) per cryptographic operation is very high. Within the UbiSec&Sens project IHP developed an ECC accelerator that promises to minimise the total required energy while limiting the needed gate area. The reported energy consumption is 13μJ for a 163 bit EC point multiplication and 27μJ for a 233bit ECPM requiring gate areas of 1.0mm^2 and 1.4mm^2, respectively (IHP CMOS 0.25μm).

Table 19 shows the comparison of the IHP design and other known hardware implementations of accelerators for the EC scalar point multiplication. Due to different hardware configurations and different amount of functionality, the numbers cannot be compared directly.

For example, the design presented in [32] supports not only ECs based on GF(2^m) but the curves on prime fields $GF(p)$, as well. This renders this ASIC design to the most configurable EC co-processor. The hardware proposed in [12] also supports not only one curve but all ECs based on binary extension fields $GF(2^m)$ up to a size of $m = 256$. Both these designs achieve flexibility at the cost of large area and poor performance compared to the IHP implementation. The design described in [27] is a very area efficient implementation of an EC, based on $GF(2^{167})$. It does not reach the speed of our design but it is very small. With a LUT number of 3002, it requires about half of the area of the IHP design on the FPGA. Unfortunately, no data of this design's energy consumption has been published. From our knowledge the fastest ECC design was reported in [35]. It requires merely 50 μs for a 163 bit ECPM, i.e., it is 40% faster than the IHP design, but with its 25000 LUTs it is about five times larger than the IHP implementation. In contrast, the ASIC design presented in [43] is very small and manufactured in the 0.35 μm technology has a size of 1.31mm^2. It supports two fields, but requires more than 6 ms for an ECPM. The energy consumption reported for this design is 213μJ for an ECPM in $GF(2^{191})$, what is about the tenfold of the IHP design. Another design that reports the power consumption is the commercially offered one presented in [9]. With this design a 233 bit ECPM requires 6 ms and a total energy of 140μJ for the 50 MHz design, manufactured as 0.13 μm ASIC.

The comparison clearly indicates that the IHP approach outperforms all other designs if area and processing time are taken into account. From our knowledge it is the most energy efficient implementation that has been reported. Consequently it is recommended to take it as reference implementation for an ECC accelerator for WSNs.

4.1.4 Energy Harvesting

Though today's sensor nodes usually depend on battery power it is considerable that they can harvest the energy they need. The most obvious and known variation is the application of solar energy. If the nodes are deployed outside where they can harness the sun as voltage producer it would be a great opportunity to prolong the life time of the network without the need to change batteries anymore.

Beside this well known solar approach several new sources for energy harvesting are emerging. In application areas that have high temperature differences it is possible to exploit the Seebeck effect that directly produces voltage. Another source of voltage can be kinetic or mechanical strain. Due to the piezoelectric effect some crystals are able to generate voltage in response to mechanical effects. The mechanic strain can be caused by the movement of the device, but also by seismic vibrations and even sound waves. However, the gained electrical energy by today's technologies is mostly not sufficient to supply a complete sensor node. A last possibility for energy harvesting is the harnessing of ubiquitous radio waves as source of voltage (like RFIDs). But unless the device is not close to the radio source it is very unlikely to gain enough energy to drive a sensor node.

Finally, it can be stated that energy harvesting technologies exist, but their potential application is very depended on the environment. Anyway, the efficiency of these technologies is still very limited. However, for a small part of the node, maybe a wake-up-mechanism, they are considerably applicable, while the actual node hardware can still be driven by traditional battery power.

4.2 System Architecture

In this subsection we differentiate between the network architecture and the middleware architecture. The former reflects the network entities and their interaction, whereas the latter concerns the software architecture of each entity.

4.2.1 Network Architecture

We are considering heterogeneous networks, i.e., networks consisting of a wired and a wireless part. The latter is formed by sensor nodes which can be connected via gateways to a wired network. In the wired network a control center resides where the application is running. We do not require connectivity for 24 hours seven days a week, but allow or even better require the wireless part of the network to run correct for longer periods w/o connectivity to its control center.

We allow more than one gateway to be part of the network which has significant benefits regarding load balancing inside the WSN due to allowed alternate paths to different WSN areas. In addition, multiple gateways improve reliability i.e., help to prevent WSN partitioning or isolation of WSN parts.

Figure 4 shows such a network and identifies four types of entities:

- Sensor nodes
- Gateways
- Readers

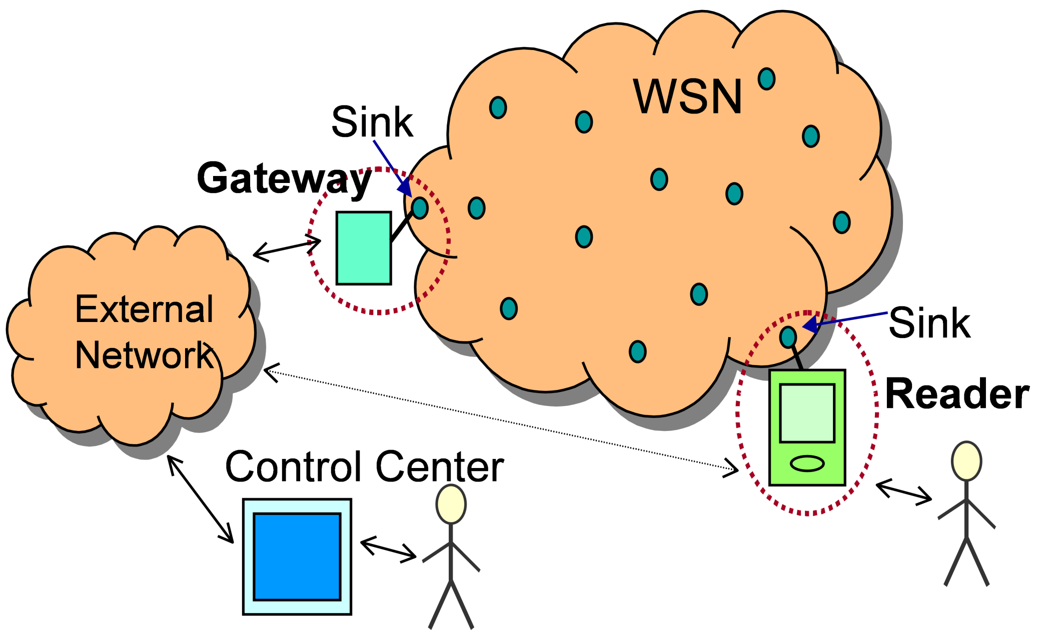

Figure 4: Network set-up consisting of a heterogeneous network and four specific device classes

- Control Center

A Gateway is a fixed base station. It is constituted by a WSN node (sink) and a PC or equivalent computing platform. A gateway links the WSN to the Control Center using an external network. A gateway is usually connected to the mains power, and has better physical protection mechanisms than sensor nodes have. Gateways have considerable computing and storage resources, and they may also perform aggregation and data storage functions.

A Reader is a portable device such as a PDA or equivalent computing platform. It allows a user to access the WSN directly as it includes a user interface, and is capable to connect to a sensor node. In a way, a Reader can be thought as a portable Control Center, with reduced functionality. A Reader allows a user to approach a WSN or to traverse it, and access information about specific sensors, aggregated data or data stored in the network. It allows also receiving alarms from the WSN. A Reader can be very useful in situations a WSN becomes partitioned. If a Reader needs to access global information from the Control Center it will use an external network (e.g., WiFi, GPRS, UMTS) to establish a data connection and request the required information.

The Control Center runs applications that support the management of the WSN and allow access to data from specific sensors and aggregated data, and also to receive alarms.

Mobility aspects

In the majority of possible set-ups the sensor nodes, gateways and the control center will be stationary whereas readers will be mobile.

In order to support mobility of readers appropriately the WSN needs to know about readers somewhat in advance. As soon as such a device approaches the WSN it will announce itself and the WSN network will reconfigure. If the Reader moves very slowly compared to the time needed for routing algorithms to produce new routes, it can be seen as an additional gateway. However, if the reader moves fast compared to re-routing restrictions may apply to the functionality. For example, it may only access data in its vicinity, which is defined by the time needed to reestablish properly. I.e., the faster the mobile device moves the smaller its vicinity.

The support of mobile devices has significant impact on the realisation of authentication and authorisation mechanisms. I.e., due to the fact that a mobile device may set-up connections to any sensor node, each sensor node needs the capability to deal with these issues. This requires the AA means to be really lightweight with respect to processing effort and memory consumption (See Figure 5).

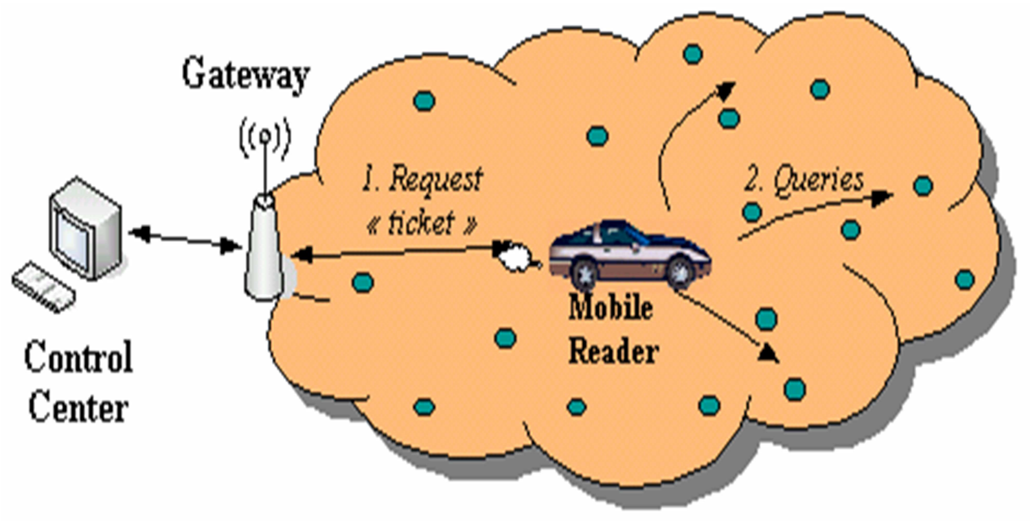

Figure 5: A mobile reader here depicted as a car may move through a WSN while retrieving data from the WSN

4.2.2 The Toolbox Approach

The major advantage of the UbiSec&Sens approach is that it addresses flexiblity or more precise adaptability from the very beginning. The security provided by the UbiSec&Sens toolbox can be tailor made on a per application basis and even be adapted during the life cycle of a certain application. This level of flexibility is achieved by a modular middleware architecture and by introducing the concept of a middleware compiler.

4.2.3 Middleware Compiler

The general task of middleware platforms is to provide a certain level of abstraction that simplifies the development of applications for an application and/or device domain. The UbiSec&Sens project provides suitable building blocks to provide middleware-like functionality by realizing

- High level APIs
- Basic and Complex security services
- Networking support

Since the gravity center of this project is security for WSN as such and not for a specialised application domain or even a single appliction several solutions for each security service are required to be capable to provide security for a wide range of applications.

The set of security modules developed in this project do not constitute a concrete middleware but span a wide range of possible secure middleware configurations, which might be used by other more abstract middleware approaches or directly by certain applications. Thus, UbiSec&Sens security modules are the building blocks for a concrete instantiation of a secure middleware for a certain application.

The selection of the suitable security modules is done by our middleware compiler. In order to generate a suitable set of security modules for a certain application reasonable constraints need to be defined in advance. These constraints are on one hand due to the limitations of wireless sensor nodes and on the other hand imposed by the security that the application under development requires. The hardware driven constraints are for example processing power and available energy to name just a few. Application dependent constraints are lifetime of the overall network, security features like secrecy of measured data or similar. In order to define the relevant constraints an XML based description language for sensor nodes and application requirements is under development. Also the role, e.g., whether or not a sensor node will be an aggregator influences its software set-up. The sensor node description provides information concerning the hardware set-up of a sensor node plus relevant information of its software configuration such as operating system used and already allocated memory.

In addition to the description of the above mentioned constraints the UbiSec&Sens security modules provide a self description. This description provides information concerning the functionality of the module, and the

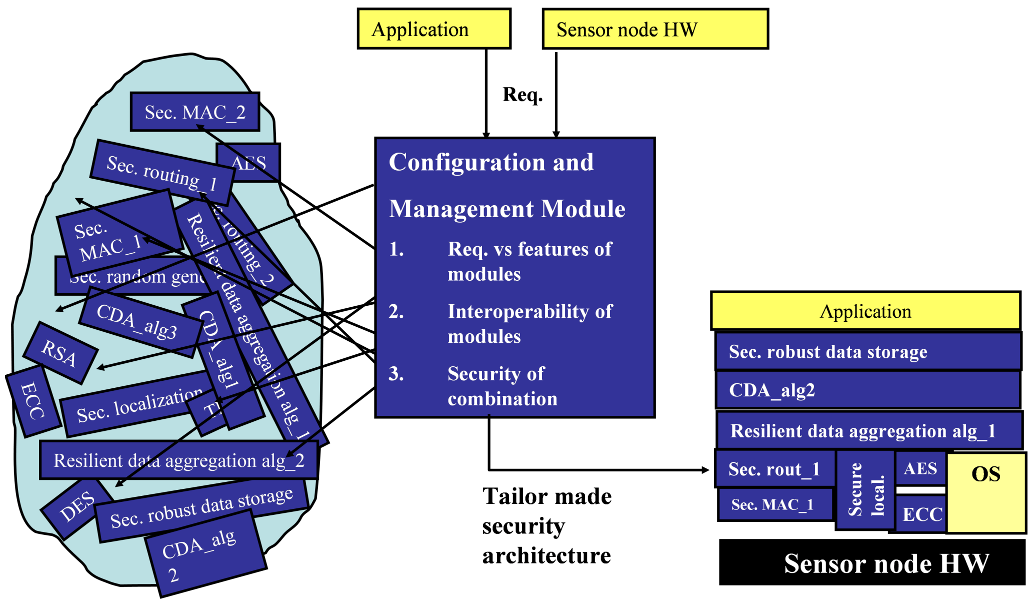

Figure 6: The middleware compiler appropriate UbiSec&Sens modules are selected due to application requirements and sensor node constraints.

resources—memory footprint and processing power—required by the module. If the module provides a complex functionality its description also contains information on potential dependencies of other UbiSec&Sens security modules. For example cipher mechanisms may require that a secure random number generator is deployed together with the cipher mechanism itself.

Figure 6 illustrates the idea of the middleware compiler. The result of a successful compiler run is an instance of the UbiSec&Sens secure middleware.

If the configuration of sensor nodes is changed during their life time, this is recorded at the WSN configuration map repository (see Figure 7). The repository always reflects the middleware instantiation of all sensor nodes starting from the first set-up. The current set-up of all nodes within a certain part or with a common task is used as an additional constraint whenever a code update is required after deployment. By that interoperability inside the WSN can be guaranteed, e.g., the use of different aggregator node election algorithms can be avoided.

4.2.4 Middleware Architecture

The UbiSec&Sens middleware architecture distinguishes between four types or classes of components, where each component consist of one or more modules. These classes are:

- sensor node abstraction layer
- basic services
- complex services
- middleware core

The sensor node abstraction layer is the only operating system and hardware dependent component. It has to be adapted individually for each OS/hardware combination that shall be supported.

Basic services are modules that do not support several functionality but just one. But they may rely on other basic services such as cipher means do since they require random number generators to be deployed.

Complex services provide a multitude of different functionalities for example aggregation and persistent storage of sensor reading can be provided by the tinyPEDs service. In order to fulfil their tasks complex services may require support from basic services, e.g., to do encryption or decryption. But complex services may be

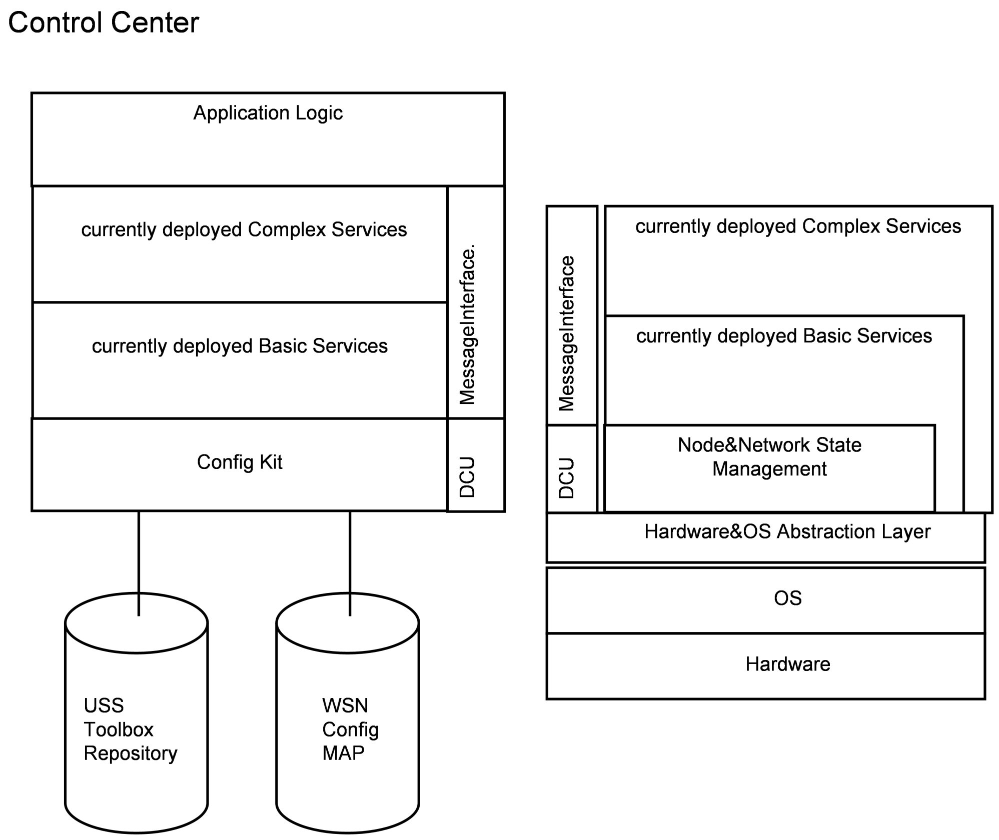
Control Center
Application Logic
currently deployed Complex Services
currently deployed Basic Services
Config Kit
MessageInterface.
DCU
USS Toolbox Repository
WSN Config MAP
MessageInterface
DCU
currently deployed Complex Services
currently deployed Basic Services
Node&Network State Management
Hardware&OS Abstraction Layer
OS
Hardware

Figure 7: The UbiSec&Sens Middleware architecture: control center configuration at the left sensor node configuration at the right hand side

implemented in a monolithic way so that they do not need any basic services to fulfil their tasks. Therefore they may access the abstraction layer directly as basic services do.

The middleware core consists of modules that are necessary to guarantee proper functionality of the other modules and those that are needed on all devices, which set-up an UbiSec&Sens powered system:

- Dynamic code update module: This module is necessary to allow reconfiguration of sensor nodes during their lifetime. Potential triggers can be newly detected vulnerabilities of security modules or a simple reconfiguration due to deployment of new applications.

- Message interpreter: provides local intelligence which is needed to decide for example whether or not the current sensor node is capable to answer a query correctly or whether it has to forward the message. In addition it is a kind of middleware scheduler which passes incoming data to the corresponding middleware modules.

- State management module (SMM): The SMM monitors the sensor node and maintains its state. By that it can trigger a code update for example if the sensor node reaches the management state, which might be caused by expiration of timers or by external triggers such as detection of malicious nodes.

The general middleware architecture is mainly independent of the device type. I.e., the deviations between sensor nodes, readers and control centers are minor. The major differences concern the presence of the sensor node abstraction layer, which is only needed at sensor nodes, and the inclusion of the UbiSec&Sens middleware compiler which is necessary only at the control center. Reader devices may be equipped with their own middleware compiler that may be tailored for specific maintenance tasks if they provide sufficient resources.

Figure 7 depicts the UbiSec&Sens middleware architecture for sensor nodes and control centers.

The concrete instantiation of the middleware, i.e., the modules deployed at the sensor nodes, reader devices and at the control center depends on:

- the currently running application

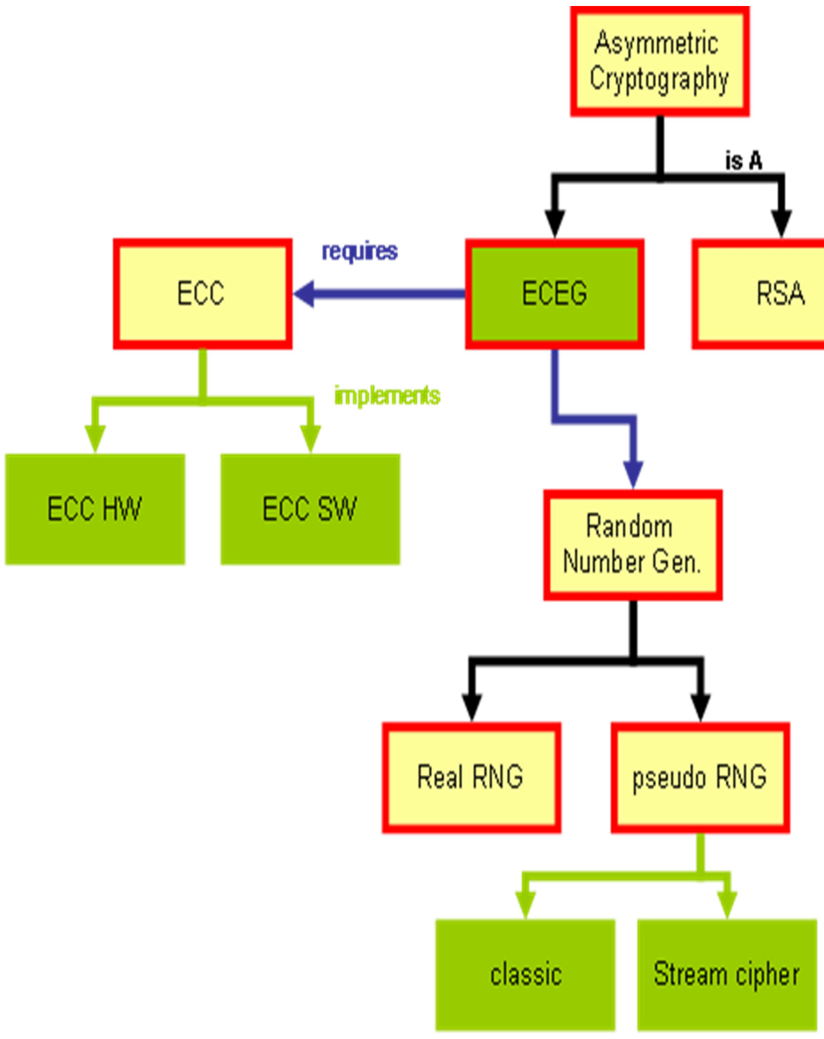

Figure 8: Dependency graph of asymmetric cipher mechanisms as used by the UbiSec&Sens middleware compiler

- the current role of the sensor node
- sensor node capabilities

4.2.4.1 Middleware APIs

Our middleware architecture as outlined so far provides merely a logical view on how complex functionality can be grouped instead of defining abstract or concrete APIs between middleware layers or modules. We clearly point out that such generic APIs are by design not needed. In our approach each module provides a module description that provides a) a method or call signature, b) the resources needed by the module as well as c) dependencies of other modules. These module description are used by our middleware compiler to generate a certain instance of our middleware. Since the compiler is also influenced by the application under development there is no need for a module independent and more general API. If an application programmer wants to bypass some UbiSec&Sens modules she still can do that by using the module description.

Figure 8 shows how a basic service is composed of even more elementary building blocks. The rectangles with the red border describe implementation independent functionality and provide an method/function call interface that can be used by all implementations that are directly linked to this description.

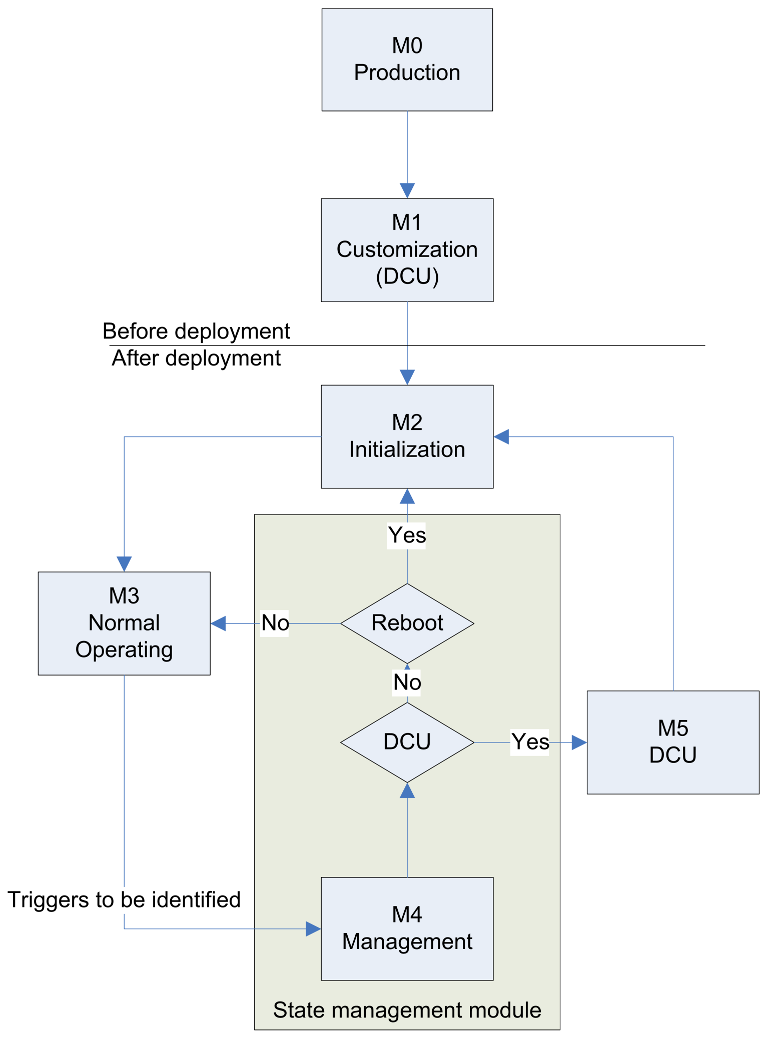

Figure 9: Sensor node states before and after deployment

4.2.4.2 Core Components

State Management Module

Each sensor node can be in one of the following five states:

- M0 production: the node is manufactured and a simple version of DCU is available at the node
- M1 customisation: all modules that are needed for the application are determined by our middleware compiler and loaded onto the sensor node by DCU in a secure environment. After that the node is ready for deployment.
- M2 initialisation: the sensor nodes are deployed and are performing the network set-up, e.g., exploring their neighbourhood and setting up routing information etc.
- M3 normal operating: the sensor node executes its application specific task.
- M4 management: The management state can be separated in at least two parts; the re-configuration mode and the self-repair mode. In the re-configuration mode nodes perform maintenance work, which usually do not require a subsequent initialisation phase but sometimes a dynamic code update. In the self-repair mode more severe errors are treated. This implies usually a subsequent initialisation phase, and most probably a dynamic code update. Transition from state M3 to M5 can be due to an unexpected operation or message received by the sensor node.

- M5 DCU: During this state the node performs a dynamic code update. The new code is stored on the node and is verified by appropriate methods such as verification of a digital signature. Subsequently, the new code is installed.

The task of the state management module is to monitor the nodes behaviour and to react appropriate. I.e., as long as the node is not in the state M4 no action has to be taken. If the state M4 is achieved the task of the SMM is to decide about the next steps. Whether a dynamic code update or re-start of the sensor node are necessary depends on the trigger which initiated the transition from M3 to M4. If DCU is executed the control center has to verify whether or not just the requesting node has to be updated or whether other nodes in its vicinity also need to be re-programmed.

Figure 9 shows the sensor node state as well as the SMM with its two major decisions.

Dynamic Code Update (DCU)

The diagram shown in figure 9 refers to DCU in state M1 and state M5. This means that the system requires the capability to change the functionality running on the sensor node right after production (M1) and also during runtime whenever it is needed (M5). In kernel based operating systems like Contiki such dynamic updates are not very challenging. Processes can be added or stopped and executable code can be stored or removed from the node.

In very resource efficient operating systems like TinyOS that merge operating system and application to one image it is not that straightforward to change the functionality. Recently several mechanisms have been developed that provide code update functionality for TinyOS. We are focussing on FlexCup [26] that allows to change methods at runtime.

It is presumed that the control center controls the code management functionality. It knows which node has which code unit in memory. Based on these information the control center sends new code segments to particular nodes or broadcasts code that has to be incorporated into the running systems.

Message Interpreter

The message interpreter consists of two parts. It has a static part that is responsible for dealing with all messages that are directed to the middleware core components DCU and SMM. In addition, its configurable part depends on the services deployed on the node. In order to properly support the configurable part the message interpreter uses a kind of registry which is shared with and maintained by the DCU module. Each time a module is exchanged, deleted or additionally installed the DCU module updates the registry. Thus, the message interpreter always knows which modules are available. Depending on the currently available modules and the node's current role and the message address the message interpreter decides what to do. In principle it is a three stages decision chain.

1. If the node is not the intended recipient the message is forwarded.

2. If the node is the intended recipient the message interpreter checks whether or not the corresponding module is deployed. If *yes*, it is checked whether or not the sensor node runs in the appropriate role. If *yes* the message is delivered, otherwise it forwarded to a more appropriate node.

3. If the node is the intended recipient but the corresponding module is not deployed the SMM is informed about this misalignment. The SMM can then decide what to do. Options are sending a misalignment message back to the control center, requiring a code update or just ignore the misalignment. The reaction of the SMM will depend on the sender of the message, i.e., if the sender is a well known trustworthy party some action will be taken otherwise the misalignment might be ignored.

Abstraction layer

The abstraction layer provides generic interfaces to basic and complex services so they can be developed independent of the underlying operating system. Due to the nature of the UbiSec&Sens modules under development we foresee two interfaces: a storage and a communication interface. The former will provide memory management functionality such as allocation of memory, store and fetch operations of data items used by higher layers.

The communication interface handles incoming and outgoing messages. The latter are passed as payload to the appropriate OS dependent interface. Incoming messages are passed to the message interpreter after removing all protocol headers and trailers if necessary.

5 Initial architecture for UbiSec&Sens demonstrators

Based on the requirements specified by the UbiSec&Sens target usage scenarios and the analysis of available hardware and software, in this section we outline major system components for the set-up of project's demonstrators. The functionality of the software modules developed in the scope of the UbiSec&Sens project will be demonstrated by three scenario specific installations. The description of the major parameters for the installations is given in Section 1. Below we specify the main building blocks of the soft and hardware platforms.

5.1 Hardware platforms for demonstrators

Since the development of own hardware platform is outside the research scope for the UbiSec&Sens project the choice of particular technology is mainly motivated by the degree of popularity of one or another platform and availability of the equipment amongst the partners. We chose popular hardware from Crossbow as the default development platform for the UbiSec&Sens toolbox. Specifically, the functionality of the developed toolbox will be demonstrated on TelosB and MicaZ devices. It is important to note that being developed for such resource constrained devices as aforementioned we foresee easy transition of the UbiSec&Sens modules to more advanced future devices with richer computational, sensing and storing functionality.

5.2 Software platform for demonstrators

As we have shown above all available operating systems have their pros and cons. For example, TinyOS offers a flexible component-based programming model. With respect to a possibility to dynamically update the installed software on sensor nodes at runtime TinyOS offers less efficient mechanisms than for example Contiki. On the other hand, TinyOS offers a variety of the existing software modules and drivers that other operating systems cannot offer. In addition TinyOS is the most popular development platform in the domain of academic sensor networking research. Since own low-level development and maintenance of the operating system functionality is outside the research scope for the UbiSec&Sens project we based our choice of the operating system on the degree of popularity of a particular platform, the available expertise among partners and availability of the existing software modules. After the analysis we chose TinyOS version 2.x as the default platform for the UbiSec&Sens software.

Having several years experience of joint development of software modules in projects involving multiple independent partners we foresee integration of future software modules from the beginning of the toolbox development. Figure 10 shows a preliminary stub architecture for the UbiSec&Sens software toolbox. In the description to follow we adopt the TinyOS *component* notation when referring to specific pieces of software. A set of specific UbiSec&Sens software components that will be implemented in the scope of the stub architecture and adopted for the project's demonstrators is described in Section 5.3.

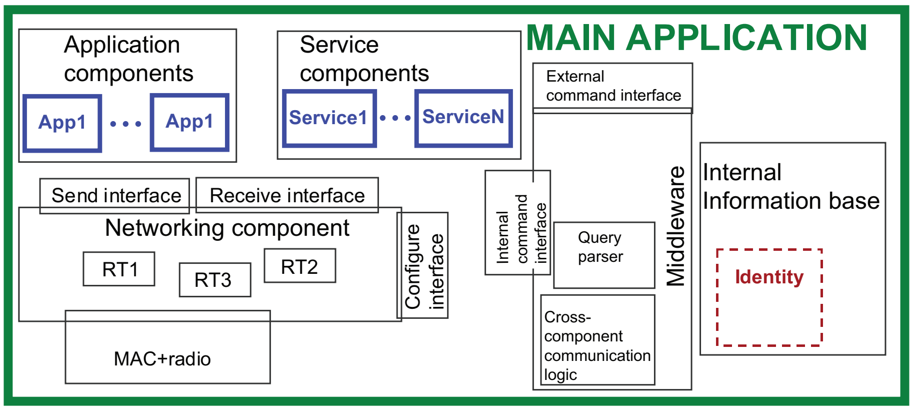

Figure 10: Software architecture for the demonstrators setup

5.2.1 Application components

As application components we regard pieces of software that assure end-user services described in the scenario specification section. It is a wrapper piece of code that includes wiring of the interfaces from other software parts of the architecture and invoking lower-level services needed for correct functioning of the application.

5.2.2 Services components

As service components we define software pieces that provide helper functionalities on different layers to other applications and services. Examples of service components are random number generators, encryption / decryption techniques, in-network processing functionality, distributed data storage etc.

5.2.3 Networking component

Networking component contains a set of functionalities associated with data transmission over network interfaces. We foresee a single instance of the networking component. In order to clarify the reason for this decision, consider routing protocols as an example. As we have shown in Section 3.1 implementation of a single routing protocol is heavy in terms of memory consumption. At the same time, parts of different routing schemes are functionally similar. As an example consider two different reactive routing schemes: one scheme is capable of setting up a multicast tree but not a unicast path, another functions in an opposite way. Due to the reactive nature of these routing schemes the functionally common part is the mechanism for propagation of route request and route reply messages.

In the scope of workpackage WP1 we attempt to minimize the implementation complexity of several network protocols needed for a particular application by extracting functionally common parts and wrapping them around the protocol specific functionalities. The networking component will provide well defined interfaces toward application and service level programmers that allow to customize the network layer functionality in a flexible way.

5.2.4 Middleware component

The functionality of the *middleware component* is described above in Section 4.2.4. In the demonstrator the middleware component should show interfaces toward external user commands and the commands issued by different components internally.

5.2.5 Internal Information Base

This component is responsible for configuring, storing and providing the information needed by other system components. The three types of functionality that will be represented in the demonstrators by this component are:

1. Meta description of services needed for the middleware component.
2. Data storage /access related functionality.
3. Identity Module.

Identity module

Identity module is the essential part of the software architecture. The term *identity* in this context includes:

- Node's low-level addresses (i.e., MAC addresses, unicast/multicast network addresses).
- Application level names associated with the node.
- Location information, including geographical coordinates, coordinates in a local coordinate system.
- Functional roles that a particular nodes plays at the moment (i.e., cluster head, aggregator, sink etc.).

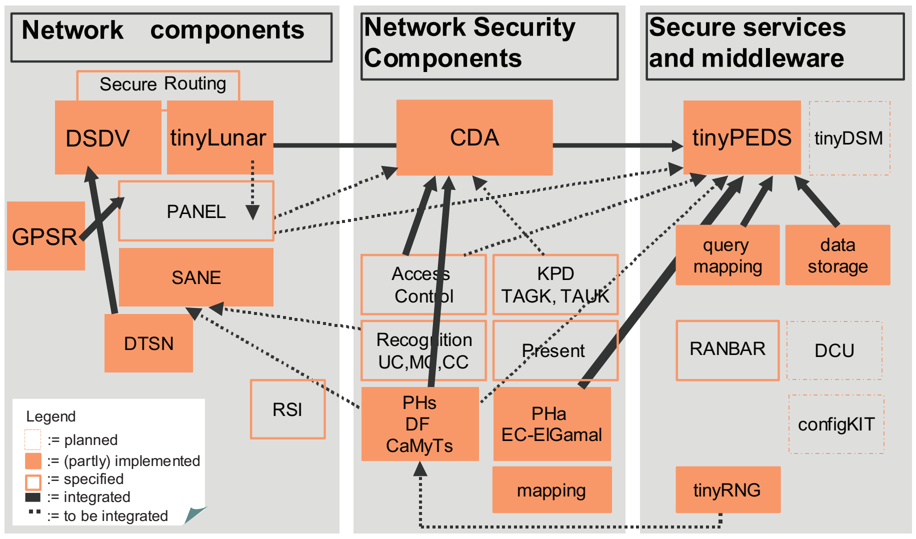

Figure 11: Specific software modules included in the UbiSec&Sens security toolbox

Some identity information is due to its nature configured statically at the deployment time and not altered afterwards. To this class of identity information we attribute GPS coordinates for static WSNs, hardware specific addresses. Another type of identity information such as roles, locality-constrained addresses and similar are auto-configured during the WSN bootstrap phase and maintained during the main operation phase. It is important that the design of this block should include protection from concurrent write access and the information consistency check.

5.3 UbiSec&Sens software modules for planed demonstrators

In this section we describe specific software modules that will be included in the UbiSec&Sens security toolbox. Figure 11 shows planned, specified and implemented pieces of software in the UbiSec&Sens project. The modules are grouped into three categories: Networking components, security components, and secure services and middleware components. The modules for which exist at least a preliminary implementation for TinyOS are shown by the shaded rectangles in the figure. The specified and implemented in a network simulator but not implemented on real hardware modules are shown by empty rectangles with the text. The planned modules are shown by dashed rectangles with the text. The figure also indicates currently integrated modules, this is shown by the bold arrows. The dashed arrows show the short term plan for modules integration. Below we summarize the content of each module.

5.3.1 Networking components

Note that the detailed specification of the modules outlined below falls beyond the scope for this deliverable and will be reported in other documents (i.e., D1.1).

5.3.1.1 TinyLUNAR

TinyLUNAR is a reactive end-to-end connection oriented routing protocol. The protocol is an adopted to the specifics of sensor networks routing scheme originally developed for mobile wireless ad-hoc networks. The Lightweight UNderlay Adhoc Routing (LUNAR) is a layer 2 protocol that utilizes an extended label-switching forwarding technology. The major property of LUNAR is a simplicity of implementation in comparison to other protocols developed for MANETs. This is achieved by reducing the route maintenance phase of the protocol to the minimum: In LUNAR all established paths automatically and periodically expire and rebuild again upon demand from the application. TinyLUNAR inherits the simplicity of its predecessor. In addition, it offers a flexible interface to the application level programmer to specify the destination node. The current implementation of tinyLUNAR in TinyOS for MICA and Telos motes offers a competitive performance and stability compared to its counterparts, e.g., tinyAODV. The goal of future development of the protocol is to include a native support for data aggregation and in-network processing. This will be done by implementing support for multicast and convergecast flows.

5.3.1.2 DSDV

The Destination-Sequenced Distance-Vector Routing (DSDV) implementation developed for UbiSec&Sens is a simplified implementation of the original MANET routing protocol, adapted to the resource restrictions that apply in WSNs. DSDV is a table-driven routing scheme for ad hoc mobile networks based on the classic Bellman-Ford algorithm. It was developed by C. Perkins in 1994. The main contribution of the algorithm was to solve the routing loop problem. Each entry in the routing table contains a sequence number, the sequence numbers are even if a link is present, otherwise an odd number is used. The numbers are generated by the destination, and the emitter needs to send out the next update with this number. In the original version, routing information was exchanged between nodes by sending full dumps infrequently and smaller incremental updates more frequently. In this implementation the routing information is disseminated as small periodic updates only, with no full dumps being generated.

5.3.1.3 DTSN

Distributed Transport for Sensor Networks (DTSN) is a novel reliable transport protocol for convergecast and unicast communications in Wireless Sensor Networks (WSNs). In DTSN, the source completely controls the loss recovery process in order to minimize the overhead associated with control and data packets. The basic loss recovery algorithm is based on Selective Repeat ARQ, employing both positive and negative acknowledgments. DTSN is able to detect when all packets of a session are lost, besides scattered gaps in the packet sequence. Caching at intermediate nodes is used to avoid the inefficiency of the strictly end-to-end transport reliability TCP-like model, commonly employed in broadband networks. Reliability differentiation is achieved by means of the smart integration of partial buffering at the source, integrated with erasure coding and caching at intermediate nodes. The implementation of DTSN in UbiSec&Sens corresponds to the basic service, which considers only total reliability and no reliability differentiation.

5.3.1.4 Secure Aggregator Node Election (SANE)

Secure aggregator node election is a protocol which ensures an non-manipulable election of a cluster-head from a fixed set of nodes. The protocol is based on random contributions of each member of the group which all together provide a value indicating which node is elected as the new aggregator. Security is based on commitment based security protocols.

5.3.1.5 PANEL

PANEL is a Position-based Aggregator Node ELection protocol, meaning that the protocol uses the geographical position information of the sensor nodes to determine the set of aggregator nodes in each epoch during the lifetime of the network. PANEL assumes that the sensor field is divided into geographical clusters. In each epoch, a reference point is computed by every node in each cluster in a distributed manner based on the epoch number. The aggregator role in each cluster is then taken by the node that is the closest to the cluster's reference point. The messages used in the aggregator node election algorithm are also used to establish routing tables in the sensor nodes that belong to the cluster. These tables are then used to send the data to be aggregated to the aggregator nodes. The reference points can also be used to route messages toward the aggregator of a distant cluster, which is useful in case of query processing and in case of backing up data collected by an aggregator at another aggregator. This latter functionality is designed into PANEL in order to support TinyPEDS.

5.3.1.6 GPSR

GPSR stands for Greedy Perimeter Stateless Routing. It is a position based routing protocol that was not developed within the project, but that can be re-used here for inter-cluster routing in combination with PANEL. In particular, GPSR can be used to route backup data and queries toward the reference point of a distant cluster. Once the message reaches the cluster, the routing algorithm switches to the intra-cluster routing protocol which uses the tables established during the aggregator node election phase. GPSR uses greedy forwarding as long as possible, and it switches to face routing when a void is encountered and the message cannot be forwarded towards the destination in a greedy manner.

5.3.1.7 RSI

The Robust Self-Initialization (RSI) protocol is a distributed self-stabilizing address assignment protocol for WSNs, which is: i) more energy efficient than previous address assignment protocols (most were developed for MANETS); ii) robust against malicious behaviour of a limited amount of nodes; iii) able to join network partitions.

The protocol was proven to be a probabilistically self-stabilized protocol, even though it uses a limited number of messages. The energy efficiency is obtained by eliminating ACK messages, and by using the reception strength of each message to optimize the number of messages required. The robustness is achieved by entering in *whispering* mode whenever a sensor node detects an attack. Finally, the protocol uses a *private nicknames* technique in order to solve the address collisions when previously disjoint networks connect.

5.3.2 Network Security Components

5.3.2.1 Concealed Data Aggregation (CDA)

Concealed data aggregation ensures end-to-end encryption of convergecast traffic with in-network processing. Aggregation functions can be average, movement detection, variance etc. The applied encryption scheme for CDA is a symmetric homomorphic encryption scheme which can be the Domingo-Ferrer encryption transformation or the stream-cipher based scheme from Castelluccia, Mykletun and Tsudik.

5.3.2.2 WSN Access Control

The WSN access control mechanism at the sink node and at each sensor node within the WSN ensures that only authorised reader devices can request data from the WSN. WSN access control is linked to the query mapping which is described below.

5.3.2.3 RU: Recognition Unicast

This module contains the implementation of a protocol which provides authentication between two communication participants in applications of WSNs. These participants can be either two sensor nodes or a sensor node and the base station. The protocol enjoys a modular construction and is based on a cryptographic pseudorandom function which can be instantiated with a message authentication code or a symmetric encryption scheme. Parts of this protocol can also be used for the authentication of messages exchanged between participants during the execution of some higher level protocol.

5.3.2.4 RM: Recognition Multicast

This module provides implementation of the authentication scheme for the multicast communication in which the base station broadcasts a message to all sensor nodes in the network. The scheme is based on a random key pre-distribution and one-bit MACs. This scheme is efficient in terms of energy and remains secure against node captures, which is extremely relevant for WSNs. Additionally, the module does not depend on the sensor network topology, so it can be applied in most scenarios and applications, i.e., the scheme is relevant for mobile readers and fixed sinks, as well for mesh or hierarchical networks.

5.3.2.5 RC: Recognition Convergecast

This module provides implementation of the framework for the in-network aggregation which focuses on the scenario with a single aggregator node. It is a novel solution which enjoys modularity and provable security and is based solely on efficient symmetric cryptographic primitives such as hash functions and message authentication codes. The framework makes use of the authenticated broadcast channel between the base station and sensor nodes as well as individual secret keys shared between each node and the base station. Additionally, it makes use of an underlying aggregator election protocol. Together with these building blocks the framework can be used as a stand alone aggregation application or can be part of some higher level applications.

5.3.2.6 PRESENT

This module provides encryption and decryption functionalities of the block cipher PRESENT. PRESENT is a lightweight symmetric block cipher which was developed within the UbiSec&Sens project. It operates on 64-bit messages either with 80-bit keys or 128-bit keys and is a substitution permutation network (SPN) with 32 rounds.

5.3.2.7 Topology Aware Group Keying (TAGK)

The topology aware group keying is a key pre-distribution approach which is shaped for CDA. TAGK is needed in cases when applying a symmetric homomorphic scheme based on group keys. The encryption scheme from Domingo-Ferrer is such a scheme. TAGK distributes per "routable region" a subset of keys from a key pool. Each node with the same key is in the role of a sensing node whereas the remaining nodes act as aggregator nodes, forwarding nodes or are in idle mode.

5.3.2.8 Topology Aware Unique Keying (TAUK)

The topology aware unique keying is a key pre-distribution scheme which provides key pre-distribution in the bootstrapping phase of the WSN for CDA based on a symmetric homomorphic encryption with single pairwise keys between the sink and each sensor node. In addition the TAUK provides key refreshment. This KPD can be applied to the homomorphic scheme from Castelluccia, Mykletun and Tsudik and therefore helps reducing data overhead due to ID transmission during the transmission phase.

5.3.3 Secure services and middleware

5.3.3.1 TinyRNG

TinyRNG is a cryptographic random number generator for wireless sensors network nodes. It uses transmission bit errors on a wireless sensor network, which are a very good source of randomness. We demonstrated that these errors are randomly distributed and uncorrelated from one sensor to another. Furthermore, these errors are difficult to observe and manipulate by an attacker. TinyRNG was designed and implemented for sensor networks, leveraging these results. The design was conducted with the aim of security, efficiency and low memory footprint in mind. It provides backward and forward security as well as production of random numbers with cryptographic quality, and resistance to reboot attacks. Moreover, it can be used trough the standard TinyOS interface for random numbers, therefore reducing application porting effort to the minimum.

5.3.3.2 Tiny Persistent Encrypted Data Storage (tinyPEDS)

The tiny persistent encrypted data storage is a middleware solution which provides distributed encrypted data storage within the WSN. Data are encrypted in a homomorphic way in a nested arrangement by applying symmetric as well as asymmetric homomorphic encryption. The sensed data can be aggregated over the time and over the region. Data replica are stored to handle exhausting nodes. They are transmitted to the actual cluster head in the right hand neighbourhood.

5.3.3.3 Tiny Distributed Shared Memory (tinyDSM)

The basic idea of the tinyDSM is to provide means that allow sensor nodes to share their sensor readings in an application dependent way. By that any of these sensor nodes can answer queries for which it has the appropriate data stored. So, this information is available also when some nodes are exhausted or in a sleep mode. In addition, tinyDSM supports an events mechanism, which allows to specify a threshold and messages that have to be sent and/or actions that have to be triggered in case the threshold is passed. tinyDSM is configured via policies. The policy file specifies for instance the replication strategy i.e., how many replica of a certain data item are required, etc. It also provides the event definitions as well as a specification of how much of the memory of the sensor node has to be reserved for a certain data item.

The tinyDSM distinguishes three priorities of its messages. Event triggered messages have the highest priority since they might be alarm messages that have to be delivered as fast as possible. Update messages have the second highest priority due to the fact that they are needed to update replicas and by that ensure consistency of the stored data. The least priority is given to queries since they do not change the systems state.

5.3.3.4 Query mapping

The query mapping module is a software module which maps “user-friendly” SQL like queries from a reader device to the sink node. Query mapping at the sink node translates user-friendly queries to controlled flooding messages. Whereas the user-friendly queries may be the same for all provided middleware approaches like tinyPEDS, tinyDSM or CDA, the controlled flooding messages for the wireless part are already adapted to the specific middleware solution.

5.3.3.5 RANBAR

RANBAR is an outlier elimination technique designed for sensor networks. RANBAR is based on the RANSAC (RANdom SAmple Consensus) paradigm, which gives us a hint on how to instantiate a model if there are a lot of compromised data elements. However, the paradigm does not specify an algorithm and it uses a guess for the number of compromised elements, which is usually not known in real life environments. The RANBAR algorithm eliminates the need for the guess, therefore it is capable to handle a high percentage of outlier measurement data by leaning on only one assumption, namely that the sample is i.i.d. in the unattacked case.

5.3.3.6 DCU

The dynamic code update (DCU) module provides means to exchange code between sensor nodes and the control center in a secure way. A more detailed description is given in section 4.2.4.

5.3.3.7 configKIT

The selection of UbiSec&Sens modules according to the requirements of a certain application and to the set-up of a certain sensor node is computed by the configKIT. It will be installed only on the control center, see 4.2.3.

References

[1] Krishnakumar Balasubramanian Arvind. Applying model-driven development to distributed real-time and embedded avionics systems. *International Journal of Embedded Systems*, Special issue on Design and Verification of Real-Time Embedded Software, April 2005.

[2] Shah Bhatti, James Carlson, Hui Dai, Jing Deng, Jeff Rose, Anmol Sheth, Brian Shucker, Charles Gruenwald, Adam Torgerson, and Richard Han. Mantis os: an embedded multithreaded operating system for wireless micro sensor platforms. *Mob. Netw. Appl.*, 10(4):563–579, 2005.

[3] A. Brown, J. Conallen, and D. Tropeano. *Models, Modeling, and Model-Driven Architecture (MDA)*, chapter Introduction:. Springer Verlag, 2005.

[4] ATMEL Corporation. *ATmega128(L) - 8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash*, 2006. Available at: http://www.atmel.com /dyn/resources/prod_documents/doc2467.pdf.

[5] ATMEL Corporation. *ATmega1281 - 8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash, preliminary Datasheet*, 2007. Available at: http://www.atmel.com/dyn/resources/prod_documents/doc2549.pdf.

[6] Moteiv Corporation. *Tmote sky - ultra low power IEEE 802.15.4 compliant wireless sensor module*, 2006. Available at: http://www.moteiv.com/products/docs/tmote-sky-datasheet.pdf.

[7] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *In First IEEE Workshop on Embedded Networked Sensors*, 2004.

[8] Matthew Eby, Jan Werner, Gabor Karsai, and Akos Ledeczi. Integrating security modeling into embedded system design. In *International Conference and Workshop on the Engineering of Computer Based Systems*. IEEE, 2007.

[9] Elliptic semiconductor. *CLP-22 Elliptic Curve Point Multiplier Core*, 2006. Available from http://www.ellipticsemi.com/CLP-22_60102.pdf.

[10] Aurelien Francillon and Claude Castelluccia. TinyRNG: A cryptographic random number generator for wireless sensors network nodes. In *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks and Workshops, 2007. WiOpt 2007. 5th International Symposium on*, pages 1–7, Limassol, Cyprus, 2007.

[11] Vipul Gupta, Matthew Millard, Stephen Fung, Yu Zhu, Nils Gura, Hans Eberle, and Sheueling Chang Shantz. Sizzle: A standards-based end-to-end security architecture for the embedded internet (best paper). In *PERCOM '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, pages 247–256, Washington, DC, USA, 2005. IEEE Computer Society.

[12] N. Gura, S. Shantz, H. Eberle, D. Finchelstein, S. Gupta, V. Gupta, and D. Stebila. An end-to-end systems approach to elliptic curve cryptography. In *CHES '02: Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*, 2002.

[13] J. Hatcliff, W. Deng, M. Dwyer, G. Jung, and V. Prasad. Cadena: An integrated development, analysis, and verification environment for component-based systems. In *Proceedings of the 25th International Conference on Software Engineering*, 2003.

[14] J. Hill, P. Levis, S. Madden, A. Woo, J. Polastre, C. Whitehouse, R. Szewczyk, C. Sharp, D. Gay, M. Welsh, D. Culler, and E. Brewer. TinyOS: http://www.tinyos.net, December 2005.

[15] CrossBow Technology Inc. *MTS / MDA, Sensors, Data Aquisition Boards*. Available at: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MTS_MDA_Datasheet.pdf.

[16] CrossBow Technology Inc. *TelosB Mote Platform Datasheet*. Available at: http://www.xbow.com/Products /Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf.

[17] CrossBow Technology Inc. *MPR / MIB User's Manual*, 2005. Available at: http://www.xbow.com /Support/Support_pdf_files/MPR-MIB_Series_Users_Manual.pdf.

[18] CrossBow Technology Inc. *Imote2 Datasheet*, 2007. Available at: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/Imote2_Datasheet.pdf.

[19] Microchip Technology Inc. *25AA1024, 25LC1024, 1 Mbit serial EEPROM memory, preliminary Datasheet*, 2006. Available at: http://ww1.microchip.com/downloads/en/DeviceDoc/21836B.pdf.

[20] Texas Instruments Inc. *Single-Chip 2.4 GHz IEEE 802.15.4 Compliant and ZigBee(TM) Ready RF Transceiver*. Available at: http://www-s.ti.com/sc/ds/cc2420.pdf.

[21] Texas Instruments Inc. *Single-Chip Very Low Power RF Transceiver*. Available at: http://www-s.ti.com/sc/ds/cc1000.pdf.

[22] Texas Instruments Inc. *MSP430 Family of Ultra-lowpower 16-bit RISC Processors*, 2005. Available at: http://www-s.ti.com/sc/ds/msp430f1611.pdf.

[23] Manish Kochhal, Loren Schwiebert, and Sandeep Gupta. Role-based hierarchical self organization for wireless ad hoc sensor networks. In *WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pages 98–107, New York, NY, USA, 2003. ACM Press.

[24] P. Levis and D. Culler. Mate: A tiny virtual machine for sensor networks. In *International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, USA*, Oct. 2002. To appear.

[25] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[26] 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 Kay Römer, Holger Karl, and Friedemann Mattern, editors, *Wireless Sensor Networks: Third European Workshop, EWSN 2006, Zurich, Switzerland, February 13-15, 2006. Proceedings*, Lecture Notes in Computer Science; 3868, pages 212–227. Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Springer-Verlag, Januar 2006. ISBN 3-540-32158-6.

[27] Gerardo Orlando and Christof Paar. A high performance reconfigurable elliptic curve processor for $GF(2^m)$. In *CHES '00: Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*, pages 41–56, London, UK, 2000. Springer-Verlag.

[28] Steffen Peter, Peter Langendoerfer, and Krzysztof Piotrowski. Public Key Cryptography Smart Dust is Affordabe. In *International Journal of Sensor Networks (IJSNet)*. Inderscience Publishers, to be published 2007.

[29] Krzysztof Piotrowski, Peter Langendrfer, and Steffen Peter. How public key cryptography influences wireless sensor node lifetime. In *Proceedings of the Fourth ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN)*, 2006.

[30] Kay Römer and Friedemann Mattern. The design space of wireless sensor networks. *IEEE Wireless Communications*, 11(6):54–61, December 2004.

[31] Nazar A. Saqib, Francisco Rodríguez-Henríquez, and Arturo Díaz-Pérez. A parallel architecture for fast computation of elliptic curve scalar multiplication over $GF(2^m)$. In *IPDPS*, 2004.

[32] Akashi Satoh and Kohji Takano. A scalable dual-field elliptic curve cryptographic processor. *IEEE Trans. Comput.*, 52(4):449–460, 2003.

[33] NORDIC Semiconductors. *nRF24L01 - Single chip 2.4 GHz Transceiver*, 2006. Available at: http://www.nordicsemi.no/files/Product/data_sheet/Product_Specification_nRF24L01_1.0.pdf.

[34] Sensirion. *SHT1x / SHT7x, Humidity & Temperature Sensor*. Available at: http://www.sensirion.com/en/pdf/product_information/Data_Sheet_humidity_sensor_SHT1x_SHT7x_E.pdf.

[35] Chang Shu, Kris Gaj, and Tarek A. El-Ghazawi. Low latency elliptic curve cryptography accelerators for nist curves over binary fields. In *Proceedings of the 2005 IEEE International Conference on Field-Programmable Technology (FPT)*, 2005.

[36] J. Stankovic, R. Zhu, R. Poornalingam, C. Lu, Z. Yu, M. Humphrey, and B. Ellis. Vest: An aspect-based composition tool for real-time systems. In *Proceedings of the IEEE Real-time Applications Symposium*, 2003.

[37] Jan Steffan, Ludger Fiege, Mariano Cilia, and Alejandro Buchmann. Towards multi-purpose wireless sensor networks. In *ICW '05: Proceedings of the 2005 Systems Communications (ICW'05, ICHSN'05, ICMCS'05, SENET'05)*, pages 336–341, Washington, DC, USA, 2005. IEEE Computer Society.

[38] STMicroelectronics. *STR71xF, Datasheet*, 2006. Available at: http://www.st.com/stonline/books/pdf/docs/10350.pdf.

[39] Website. Freertos homepage.

[40] Website. Project sunspot web portal.

[41] Website. Sos 2.x home page.

[42] Kamin Whitehouse, Cory Sharp, Eric Brewer, and David Culler. Hood: a neighborhood abstraction for sensor networks. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 99–110, New York, NY, USA, 2004. ACM Press.

[43] Johannes Wolkerstorfer. Is elliptic-curve cryptography suitable to secure rfid tags? In *Workshop on RFID and Light-Weight Crypto*, 7 2005.

[44] Yong Yao and J. E. Gehrke. The cougar approach to in-network query processing in sensor networks. *ACM SIGMOD Record*, 31(2):9–18, September 2002.

[45] Stephen S. Yau, Fariaz Karim, Yu Wang, Bin Wang, and Sandeep K. S. Gupta. Reconfigurable context-sensitive middleware for pervasive computing. *IEEE Pervasive Computing*, 1(3):33–40, 2002.