FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Information Model and Security Concept for Distributed Traffic Management Applications based on Wireless Systems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Stefan Szucsich
Matrikelnummer 0728333

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof.Dr. Wolfgang Kastner
Mitwirkung: Univ.-Ass., Dipl. Ing. Lukas Krammer

Wien, 12.12.2012

(Unterschrift Verfasser)

(Unterschrift Betreuung)

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Information Model and Security Concept for Distributed Traffic Management Applications based on Wireless Systems

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

by

Stefan Szucsich
Registration Number 0728333

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof.Dr. Wolfgang Kastner
Assistance: Univ.-Ass., Dipl. Ing. Lukas Krammer

Vienna, 12.12.2012

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Stefan Szucsich
Obere Hauptstraße 57, 7304 Großwarasdorf

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasser)

To my father

Johann Jakob Szucsich

$*$1955 †2010

Abstract

Modern traffic management systems are widely used to control the increasing traffic and to optimize the traffic flow in urban and interurban environments. They can differ in the functionality and number of components involved. For instance, in an interurban environment numerous variable message signs can be controlled based on data coming from radar detectors. In general, modern traffic management systems follow a hierarchical structure. While higher levels collect data from lower levels and provide them to human operators for global traffic monitoring and control, lower levels are responsible for autonomous control as well as traffic and environmental data acquisition. At the lowest level of the hierarchy, i.e., the field level, sensors (e.g. detector loops, temperature sensors) and actuators (e.g. traffic lights, variable message signs) from various vendors are connected to local control units by so called I/O convertors. Since in most cases sensors and actuators use vendor specific protocols for communication, I/O convertors are responsible for translating messages from the standardized higher level communication protocol to a vendor specific protocol and vice versa. However, higher level communication protocols are only regional restricted (de-facto) standards. Thus, extending them onto the field level would not yield a satisfying solution especially for vendors.

Wireless sensor and actuator networks typically consist of many different types of sensors and actuators controlling some physical process. Due to the large number of nodes gathering process data, the nodes of a wireless sensor network are required to be small and low-cost. This in turn leads to limited processing capabilities and the need for low power consumption. However, the need for security contradicts the need for low-cost sensor nodes. The main security challenges in wireless sensor networks are limited resources, large-scale networks, dynamical network topologies and last but not least wireless communication characteristics.

This thesis is twofold. On the one hand it aims at developing a generic information model for sensors and actuators applied in the traffic domain. The information model in combination with wireless data communication allows to replace the centralized approach at the field level of today's traffic management systems with a distributed network of autonomously cooperating sensors and actuators. To evaluate the information model, a proof-of-concept implementation of a traffic jam warning system is presented. The second goal of this thesis is the development of a security concept for distributed traffic management applications based on wireless communication systems. Within the scope of the security concept, a novel broadcast authentication scheme for wireless sensor and actuator networks is proposed and analyzed in detail.

Kurzfassung

Moderne Verkehrsleitsysteme steuern das steigendene Verkehrsaufkommen und optimieren den Verkehrsfluss. Sie unterscheiden sich in ihrer Funktionalität und der Anzahl ihrer Komponenten. So können etwa mehrere Wechselverkehrszeichen mithilfe von Radar-Detektoren gesteuert werden. Moderne Verkehrsleitsysteme besitzen im Allgemeinen eine hierarchische Struktur. Während die höheren Ebenen Daten von darunterliegenden Ebenen sammeln und den Operatoren zur globalen Verkehrsüberwachung und -steuerung zur Verfügung stellen, sind die unteren Ebenen für die autonome Steuerung sowie die Erfassung von Vehrkehrs- und Umweltdaten zuständig. In der untersten Ebene der Hierarchie, der sogenannten Feldebene, finden sich Sensoren (z.B. Induktivschleifendetektoren, Temperatursensoren) und Aktoren (z.B. Ampeln, Wechselverkehrszeichen) von einer Vielzahl an Herstellern. Diese sind über Ein-/Ausgabe-Konzentratoren mit der sogenannten Streckenstation verbunden. In den meisten Fällen verwenden Sensoren und Aktoren herstellerspezifische Kommunikationsprotokolle. Daher müssen E/A-Konzentratoren Nachrichten zwischen herstellerspezifischen Protokollen und standardisierten Protokollen der darüberliegenden Ebenen konvertieren.

Drahtlose Sensor- und Aktor-Netzwerke bestehen häufig aus einer Vielzahl von unterschiedlichsten Sensoren bzw. Aktoren, die einen physikalischen Prozess überwachen bzw. steuern. Aufgrund der großen Anzahl an Sensoren, die oftmals benötigt werden, um Prozessdaten zu sammeln, sollen diese möglichst klein und kostengünstig sein. Diese Anforderungen führen in weiterer Folge dazu, dass die eingesetzten Sensoren typischerweise batteriebetrieben sind und nur über eine stark eingeschränkte Rechenleistung verfügen. Es ist leicht ersichtlich, dass der Bedarf an Sicherheit im Gegensatz zu den zuvor genannten Eigenschaften von drahtlosen Sensor-Netzwerken steht. Die größten Herausforderungen im Bezug auf Sicherheit in drahtlosen Sensor- und Aktor-Netzwerken sind die stark eingeschränkten Ressourcen, die Netzwerkgröße, die Dynamik eines Sensor- und Aktor-Netzwerkes und nicht zuletzt die Charakteristik von drahtloser Datenübertragung selbst.

Die vorliegende Arbeit verfolgt zwei Ziele. Einerseits soll ein generisches Informationsmodell für Sensoren und Aktoren aus dem Verkehrswesen entwickelt werden. Anhand dieses Informationsmodells kann ein Kommunikationsprotokoll für drahtlose Datenübertragung spezifiziert werden, das den zentralistischen Ansatz auf der Feldebene moderner Verkehrsleitsysteme durch ein verteiltes Netzwerk von autonom interagierenden Sensoren und Aktoren ersetzen könnte. Weiters wird zur Evaluierung des Informationsmodells ein Prototyp einer Stauwarnanlage vorgestellt. Das zweite Ziel dieser Arbeit ist die Entwicklung eines Sicherheitskonzepts für verteilte Applikationen in der Verkehrstelematik. Im Zuge dessen wird ein neuartiges Verfahren zur Broadcast-Authentifizierung vorgestellt und im Detail analysiert.

Contents

List of Algorithms

List of Figures

List of Tables

Part I

Introduction and basic concepts

CHAPTER 1

Introduction

1.1 Motivation and problem statement

Traffic Management Systems (TMSs) follow a hierarchical structure [17]. At the lowest level, i.e., the field level, Local Control Units (LCUs) are connected to sensors and actuators collecting traffic and environmental data by wired communication systems. An LCU typically consists of a control module and optional I/O convertors that are responsible for translating messages from the LCU's standardized higher level communication protocol to a vendor specific protocol and vice versa. Due to this centralized approach and the lack of a standardized communication protocol for sensors and actuators applied in today's TMSs, direct communication between sensors and actuators is not possible. Even worse, the failure of an LCU leads to the loss of all its sensors and actuators despite operating properly.

The wiring of sensors and actuators itself has disadvantages, too. It impedes the installation of new infrastructure and hence increases the costs of installation. Furthermore, it lowers the flexibility of TMS facilities. If the infrastructure for TMS facilities can be kept at a minimum, for example by using wireless technologies for data communication and alternative energy supply such as photovoltaic cells, TMS facilities can cost-effectively be employed at arbitrary sites.

To overcome the disadvantages mentioned above, a decentralized network consisting of autonomously cooperating sensors and actuators can replace the centralized approach at the field level of modern TMSs. However, interoperability between sensors and actuators from different vendors must be ensured. To this end, a generic information model needs to be defined building the basis for a standardized communication protocol. Furhermore, wireless technologies allow the use of flexible TMS facilities while reducing installation costs. Nevertheless, due to traffic management being a safety-critical field of application, security aspects, such as message authentication, confidentiality and integrity as well as availability and data freshness, must be considered. Due to the resource limitations of sensors and actuators, especially broadcast authentication is a crucial though not trivial task.

1.2 Aim of this thesis

The main objectives of this thesis are the development of a generic information model for the variety of sensors and actuators that can be found in today's TMSs as well as the development of a security concept for distributed Traffic Management (TM) applications that are based on wireless communication systems. As part of the security concept, a novel broadcast authentication scheme for Wireless Sensor and Actuator Networks (WSANs) is proposed and analyzed. Finally, a proof-of-concept implementation based on ZigBee [40] is presented along with other use cases.

As a result of this thesis, the centralized approach at the field level of modern TMSs could be replaced with a decentralized approach with securely cooperating sensors and actuators. Moreover, by using wireless data communication, far more flexible TMS facilities can be produced reducing installation costs while allowing employment at arbitrary sites.

1.3 Outline

As mentioned in the introduction, the main objectives of this thesis are the development of a generic information model for sensors and actuators in the traffic domain as well as a security concept for distributed TM-applications that are based on wireless communication systems. Thus, Chapter 2 provides an overview of modern TMSs and some of the most popular wireless communication standards for low-cost and low-power devices such as sensors and actuators. It also explains the notion of information modeling and introduces basic security concepts which are needed later on.

Part II addresses the generic information model for sensors and actuators in the traffic domain. Chapter 3 specifies the information model including commands and attributes for general device configuration, traffic control as well as traffic and environmental data acquisition. In addition to the theoretical specification, Chapter 4 presents a proof-of-concept implementation of a traffic jam warning system based on ZigBee and depicts further use cases.

Part III covers the security concept for distributed TM-applications based on wireless communication systems. In Chapter 5, the security concept is specified. It includes the proposal of a novel broadcast authentication scheme for WSANs as well as a precise description of frame protection and key management mechanisms. Chapter 6, evaluates the security concept and provides a detailed analysis of the proposed broadcast authentication scheme.

Finally, Chapter 7 summarizes the main results of this thesis and gives an outlook on further research work.

CHAPTER 2

State of the art

2.1 Traffic management systems

Modern Traffic Management Systems (TMSs) are widely used to control the increasing traffic and to optimize the traffic flow in urban and interurban environments. TMSs can differ in the functionality and number of components involved. For instance, in an interurban environment numerous Variable Message Signs (VMSs) can be controlled based on data coming from radar detectors. In general, TMSs follow a hierarchical structure [17]. The Traffic Management and Information Center (TMIC) operates at the highest level of this hierarchy. It collects data from its underlying Sub-Stations (SSs) and provides them to the users for global traffic monitoring and control. SSs are in turn responsible for autonomous control as well as data collection from their interconnected Local Control Units (LCUs). LCUs operate at the lowest level of the hierarchy, the field level. They are connected to sensors (e.g., detector loops, temperature sensors) and actuators (e.g., traffic lights, VMSs) from various vendors by so called I/O convertors. Communication between devices of different levels is standardized, whilst in most cases sensors and actuators use vendor specific protocols for communication. Note that the number of levels involved may depend on size and complexity of the TMS. The minimum system can be composed of autonomously acting LCUs. While this is sufficient for simple applications, e.g. the visualization of successive speed reduction, more complex applications, e.g. control of VMSs with rerouting instructions, require multiple levels of control and monitoring facilities.

As mentioned above, communication between devices of different levels is standardized. Unfortunately, only regional restricted (de-facto) standards exist. Table 2.1 depicts some of the most important communication standards and the communication levels involved. As illustrated in Figure 2.1, levels A, B and C denote data exchange between TMIC and SS, SS and LCU as well as between SS and I/O convertor, respectively [17].

Although Intelligent Traffic Management Systems (ITMSs) have been an active research area in the last decades, human intervention is still necessary especially at the higher levels [14]. ITMSs aim to overcome limitations when facing critical traffic conditions and to support the human operator thus reducing the need for manual intervention. To this end, an ITMS needs

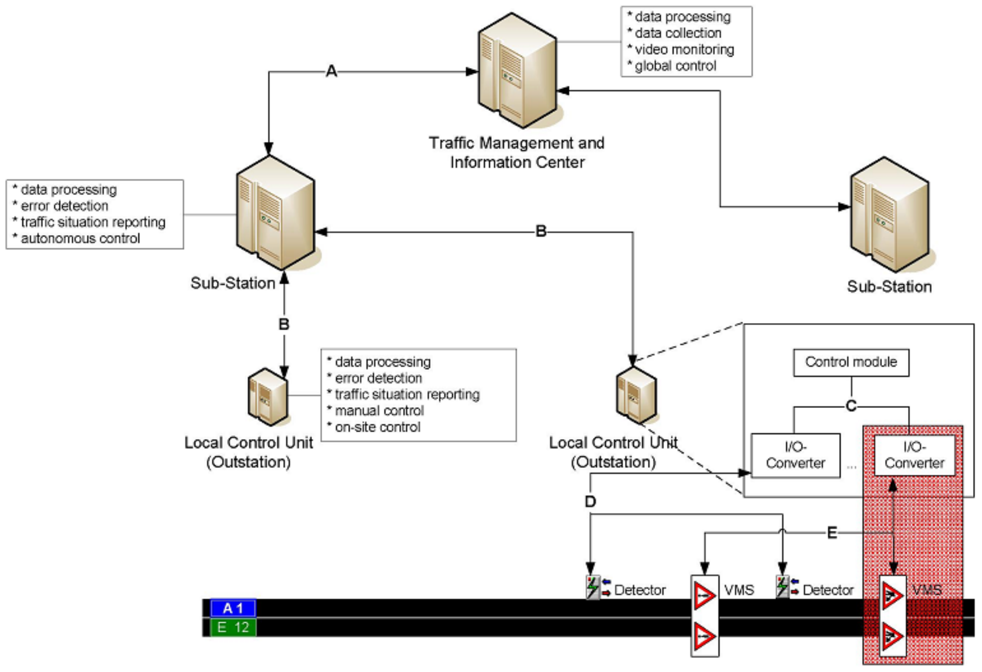


Figure 2.1: Structure of a modern traffic management system (taken from [17])

Communication standard	**Country of origin**	**Levels**		
		A	**B**	**C**
DAP [34]	Netherlands	✓	✓	✓
NTCIP [1]	USA	✓	✓	✓
TLS [4]	Germany		✓	✓
TLS over IP [2]	Austria	✓	✓	

Table 2.1: Communication standards in the traffic domain according to [17]

to comprise a knowledge model of traffic behavior at a strategic level. In [14], the application of advanced knowledge modeling techniques on two ITMSs, namely, KITS and FLUIDS, is presented. Dimitrakopoulos and Demestichas propose an ITMS based on cognitive systems and Wireless Sensor and Actuator Networks (WSANs) in [8]. Further informations on ITMSs, including a historical overview of ITMSs, can be found in [9].

2.2 Wireless standards

WSANs have been an emerging research topic in the recent decade. The following sections provide an overview of some wireless standards.

2.2.1 IEEE 802.15.4

The IEEE 802.15.4 standard [11] specifies Physical Layer (PHY) and Medium Access Control Layer (MAC) for Low-Rate Wireless Personal Area Networkss (LR-WPANs). A LR-WPAN is a low-cost communication network allowing wireless communication for applications with limited power and looser throughput requirements.

The PHY layer defines various frequency bands using different access modes. While the ultrawide-band PHY operates in the sub-gigahertz (249.6 MHz to 749.6 MHz), low band (3.1 GHz to 4.8 GHz) and high band (6 GHz to 10.6 GHz), Direct Sequence Spread Spectrum (DSSS) PHY supports the 2450 MHz, 950 MHz and a 868 MHz bands among others [3]. Across the latter three frequency bands, a total number of 27 channels are available: sixteen in the 2450 MHz band, ten in the 950 MHz band and one in the 868 MHz band. While the 2450 MHz band offers a data rate of 250 kbps, frequency bands 950 MHz and 868 MHz offer data rates of 40 kbps and 20 kbps, respectively. Besides data transmission and reception, the physical layer is also responsible for activation and deactivation of the radio transceiver, channel frequency selection, energy detection within the selected channel, clear channel assessment and link quality estimation.

The MAC layer handles all access to the radio channel. IEEE 802.15.4 specifies two main network topologies for Personal Area Networks (PANs), i.e., the star topology and the peer-to-peer topology (see Figure 2.2). No matter what topology is used, an IEEE 802.15.4 PAN can be *beacon enabled* or *non-beacon enabled*. In a beacon-enabled PAN, the *PAN coordinator* periodically sends beacon messages to synchronize attached devices, identify the PAN and describe the structure of the *superframes*. Superframes denote the interval between two consecutive beacons as illustrated in Figure 2.3. A superframe consists of an *active period* and an optional *inactive period*. The active period is divided into sixteen equally sized slots whereby beacons are always transmitted in the first slot. Remaining slots form the Contention Access Period (CAP) and the optional Contention Free Period (CFP). In CAP, nodes have to compete for channel access using slotted Carrier Sense Multiple Access – Collision Avoidance (CSMA-CA), whereas in CFP Guaranteed Time Slots (GTSs) can be assigned to nodes. Since PAN coordinators are not allowed to interact with their PAN during the inactive period, PAN coordinators and associated devices may enter a low-power mode to safe energy until the next superframe starts. There exist three different data transfer modes: data transfer to a coordinator, data transfer from a coordinator and peer-to-peer data transfers. Whenever a device wants to transfer data to the PAN coordinator, it has to wait for the next beacon and synchronize to the superframe structure. It then sends the data in the next CAP. If the PAN coordinator wants to transfer data to an end device it indicates a pending message in the beacon. After synchronizing to the beacon, the device sends a data request to the PAN coordinator. The coordinator then sends the pending message to the requesting device. Note that both, data request und data transfer, take place in the CAP. The last type of data transfer is peer-to-peer. This data transfer is only possible in peer-to-peer PANs where the device's radio transceivers are always active. Peer-to-peer data transfer takes place in the CAP.

In non-beacon enabled PANs only unslotted CSMA-CA is used for communication. Hence, end devices have to periodically poll the PAN coordinator for pending messages and on the other hand are able to instantly send data to the PAN coordinator.

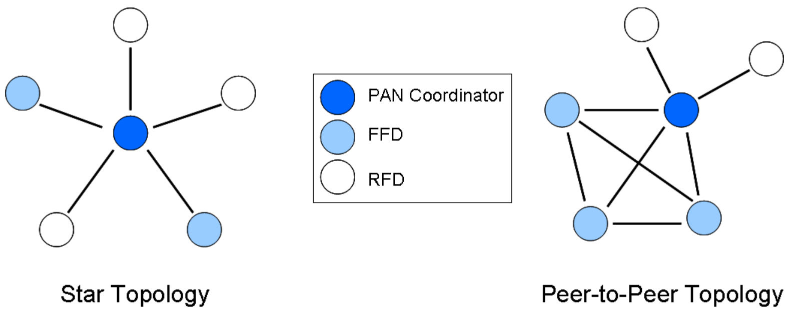


Figure 2.2: IEEE 802.15.4 PAN topologies

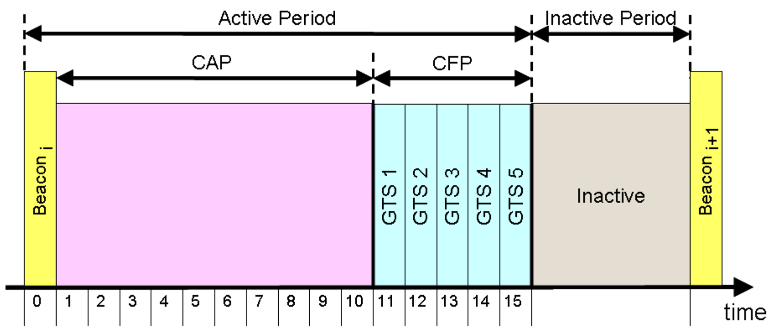


Figure 2.3: IEEE 802.15.4 superframe structure

As depicted in Figure 2.2, the MAC layer defines two types of devices, namely, Full Function Devices (FFDs) and Reduced Function Devices (RFDs). While FFDs implement the full MAC layer functionality, RFDs implement only parts of it. Only FFDs may act as PAN coordinator and furthermore, RFDs can only communicate with FFDs.

Besides (high level) data transfer, beacon synchronization and PAN (dis-)association, the MAC layer is also responsible for frame security. IEEE 802.15.4 specifies eight different security suites that can be classified by the properties they offer [30]: no security, authentication only, encryption only, and encryption and authentication. The latter three are based on the Counter with Cipher Block Chaining Message Authentication Code (CCM*) mode of operation of the Advanced Encryption Standard (AES) block cipher with keys up to 128-bits in length (see Section 2.4.2). However, there are some vulnerabilities and pitfalls especially with respect to key management and integrity protection [30].

2.2.2 ZigBee

ZigBee [40] is a low-cost, low-power, wireless communication standard built upon the IEEE 802.15.4 standard. As depicted in Figure 2.4, only the higher layers, i.e., Network Layer (NWK) and Application Layer (APL), are defined by ZigBee. While the NWK layer provides a routing mechanism for multihop networks as well as services for network management, the APL layer provides a framework for distributed applications and concerns about data modeling.

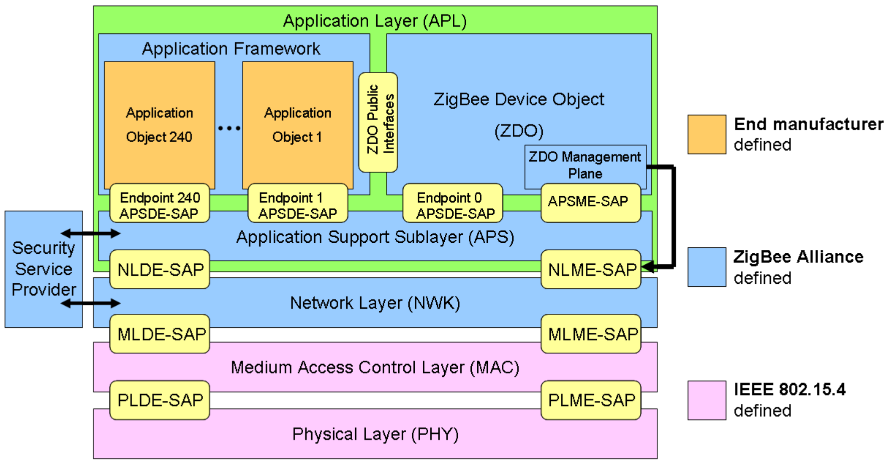


Figure 2.4: ZigBee architecture

Each ZigBee device consists of Application Objects (AOs) defining the device's functionality. An AO is a single part of a distributed application. As shown in Figure 2.4, the ZigBee Device Object (ZDO) is a special AO that is implemented in every device. It implements the basic device functionality and is responsible for the initialization of Application Support Layer (APS), NWK and Security Service Provider (SSP). The ZDO also determines the device's role within the network, i.e., whether the device acts as coordinator, router or end device.

Interoperability between devices from different vendors is ensured through the use of Application Profiles (APs). Currently, application profiles are available for specific application domains (e.g., *ZigBee Home Automation Public Application Profile* [41]). They contain a set of logical *device specifications* that describe dedicated functionalities. Each device specification defines the so called *ZigBee clusters* that have to be implemented (mandatory clusters). ZigBee clusters are collections of *attributes* and *commands*. While a single attribute of a cluster represents a single data entity (e.g., the measured air temperature), commands are used to manipulate these attributes as well as to initiate actions within the device. The exact structure of the clusters (including the specification of the clusters' attributes and commands) is not defined by the core specification – clusters are specified within the so called *ZigBee Cluster Library Specification* [39].

Although being built upon IEEE 802.15.4, ZigBee does not use the security mechanisms defined by IEEE 802.15.4. Instead, ZigBee defines its own security services for NWK and APL layers using AES in combination with the CCM* mode and three different types of keys. *Link keys* are used for all unicast communications between peer entities. They are shared between two ZigBee devices that communicate with each other. *Network keys* are shared amongst all devices within the network and are utilized for broadcast communication. Last but not least, *master keys* are used to distribute other keys.

ZigBee also provides different key management mechanisms. When using *pre-installed keys*, all required keys have to be installed during device configuration. Another possibility is to transport keys over the network (*key transport*). In secure mode, a pre-installed key is used for encryption of the transported key. For the sake of completeness it should be mentioned that keys can also be transported in plain text. Both mechanisms, pre-installed keys and key transport, can be used for all types of keys. Another mechanism is the *key establishment* based on the Symmetric-Key Key Establishment (SKKE) protocol and secured through the use of a master key. However, key establishment is possible for link keys only.

2.2.3 6LoWPAN

IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) has been specified to extend IPv6 based networks over IEEE 802.15.4 based Wireless Personal Area Networks (WPANs) [18]. However, the requirements of IPv6 based networks are in contrast to the limitations of WPANs. The main gap between IPv6 and IEEE 802.15.4 concerns message sizes: While the minimum Maximum Transmission Unit (MTU) size of IPv6 packets is 1280 bytes, the maximum size of IEEE 802.15.4 PHY frames is 127 bytes. Thus, fragmentation of (possibly already fragmented) IPv6 packets may be necessary prior transmission over the WPAN.

The architecture of 6LoWPAN is very simple. It basically consists of an adaption layer that is inserted between the MAC and NWK layers. The 6LoWPAN adaption layer is mainly responsible for encapsulation, fragmentation and reassembly, header compression and address autoconfiguration. Encapsulation may add headers for mesh addressing, broadcast and fragmentation to the IPv6 packet while header compression can be used to shrink original headers of upper layers. As mentioned above, fragmentation may be necessary if the maximum size of a IEEE 802.15.4 PHY frame (reduced by the number of bytes required by the MAC layer header) is exceeded. Address autoconfiguration allows nodes to obtain IPv6 interface identifiers based on their 64-bits IEEE 802.15.4 extended address. However, 16-bits short addresses are also possible for generation of IPv6 interface identifiers. The complete adaption layer functionality is defined in [24].

IPv6 packets are always carried within IEEE 802.15.4 data frames. To support link layer recovery, acknowledgements for data frames carrying IPv6 packets are recommended [24]. Furthermore, note that 6LoWPAN does not specify security services. It solely relies on the security mechanisms defined by IEEE 802.15.4 and higher layers.

2.2.4 WirelessHART

WirelessHART is an open wireless communication standard specifically designed for process control applications in industrial automation [33]. It extends the (wired) HART protocol offering a wireless interface to field devices. The design principles of WirelessHART are focussed on simplicity and robustness [13].

The WirelessHART PHY layer is based on IEEE 802.15.4. However, WirelessHART extends the IEEE 802.15.4 PHY layer with a frequency hopping mechanism to overcome narrow band interference. As opposed to ZigBee, WirelessHART specifies its own MAC layer using Time Division Multiple Access (TDMA) to provide collision free and deterministic communication. Each time slot has a duration of 10 ms which is enough to transmit a data frame and its acknowledgement. A sequence of consecutive time slots is called a *superframe*. Furthermore, WirelessHART defines priority classes of data units which is useful for flow control. The priority classes, from highest to lowest priority, are *control data*, *measurement data*, *normal data* and *alarm and event data*.

While the NWK layer is mainly responsible for routing and link scheduling, the transport layer defines block data transfer mechanisms for TCP-like data transfer with acknowledgements and UDP-like data transfer without acknowledgements. Finally, the WirelessHART APL layer defines commands, responses and data types.

WirelessHART networks consist of a group of network devices, a WirelessHART gateway, a *security manager* and a *network manager* [13]. A network device can be a field device or a handheld device, e.g. from a maintenance worker. The gateway builds the bridge to the process plant and communicates directly with the network manager. It also provides buffering for large data transfers. The network manager is responsible for configuration and maintenance of the WirelessHART network. It propagates keys to the network devices, performs link scheduling and validates devices that want to join the network. Since WirelessHART networks support mesh topology, each network device must be equipped with routing capabilities.

Security in WirelessHART networks is mandatory. To this end, AES with the CCM* mode as defined by IEEE 802.15.4 is used to provide confidentiality and data integrity for end-to-end connections [33]. As mentioned above, the security manager is responsible for key generation while the network manager is responsible for key distribution. WirelessHART defines four types of keys, namely, *public keys*, *session keys*, *join keys* and *network keys*. While public keys are used by joining devices to generate Message Integrity Codes (MICs) on the MAC layer, the network key is used by authenticated network devices for the same purpose. During the joining process, join keys are needed to authenticate a joining device with the network manager. Thus, joining keys have to be unique for each network device. Session keys are unique for each end-to-end connection between two network devices providing end-to-end confidentiality and data integrity.

2.3 Information modeling

Information models comprise concepts, relationships, constraints, rules and operations to specify data semantics for a certain domain of discourse [20]. They provide a sharable and structured view on information and knowledge in the context of the chosen domain. There exist three different methodologies to derive information models, namely the Entity-Relationship (ER) approach, the functional modeling approach and the Object-Oriented (OO) approach.

The ER approach is focused on the concepts of entities and relationships (among entities). Its origin is the graphical notation technique proposed by Chen [5]. However, various extensions of the basic ER modeling technique have been introduced since then.

Functional modeling is based on specification and decomposition of the system's functionality. It considers the flow of information from one system process to another and its basic elements are objects and functions (over objects).

The OO approach identifies objects from the domain of discourse prior to their operations and functions. Its main building blocks are object classes, attributes, operations and associations. One of the most widespread OO modeling language is the Unified Modeling Language (UML) specified by the Object Management Group (OMG) [25]. The main objective of UML is to provide tools for analysis, design and implementation of software-based systems as well as for modeling business. To this end, UML provides a variety of graphical notation techniques such as use case diagrams and class diagrams. In general, UML diagrams can be divided into structural and behavioral diagrams. Structural diagrams describe the structure of the system being modeled whereas behavioral diagrams characterize processes as well as control and data flows.

2.4 Security in wireless sensor networks

A WSAN typically consists of many different types of sensors and actuators controlling some physical process [6]. Due to the large number of nodes gathering process data, the nodes of a WSAN are required to be small and low-cost. This in turn leads to limited processing capabilities and the need for low power consumption. However, most applications require at least some level of security due to the open medium. Obviously, the need for security contradicts the need for low-cost sensor nodes. The main security challenges in WSANs are limited resources, large-scale networks, dynamical network topologies and wireless communication characteristics [6].

Table 2.2 depicts the classification of security attacks according to [32]. Security attacks can be divided into attacks from *outside* and *inside* the WSAN. Outside attacks are performed by unauthorized nodes, i.e., nodes that are not part of the WSAN, whereas inside attacks are made from authorized nodes within the WSAN. Both, outside and inside attacks, can further be divided into *active* and *passive* attacks. Basically, all possible attacks from inside are subsumed by *node compromise*. As soon as a node is compromised, the attacker obtains access to all security materials stored within that node and in further consequence, the node can exhibit arbitrary malicious behavior. While prevention of outside attacks is rather simple, inside attacks are hard to detect. However, intrusion detection systems, such as [12], are able to detect compromised nodes with a high degree of probability.

Classification		**Attack**	**Affects**
inside	active	Node compromise	all
outside	active	Denial of Service (DoS)	Availability
		Message replay	Freshness
		Message modification	Integrity
		Message spoofing	Authentication
	passive	Eavesdropping	Confidentiality

Table 2.2: Classification of security attacks according to [32]

The following sections describe some basic security primitives as well as symmetric- and public-key encryption schemes.

2.4.1 Basic security primitives

2.4.1.1 One-way functions

A function $f : X \to Y$ is called a *one-way function* if $f(x)$ is easy to compute for all $x \in X$ but given $y \in Im(f)$, it is hard to find any $x \in X$ such that $f(x) = y$ [22]. The terms "easy" and "hard" are to be understood with respect to the computational complexity of computing and inverting f, respectively.

For example, *RSA function* and *Rabin function* are candidate[1] one-way functions [22]. RSA function $f_{RSA} : \mathbb{Z}_n \to \mathbb{Z}_n$ is defined as $f_{RSA}(x) = x^e \bmod n$ where $n = pq$ is the product of distinct odd primes p, q and e is an integer such that $\gcd(e, (p-1)(q-1)) = 1$. Rabin function $f_{Rabin} : Q_n \to Q_n$ is defined as $f_{Rabin}(x) = x^2 \bmod n$ where $n = pq$ is the product of distinct primes p, $q \equiv 3 \pmod 4$ and Q_n is the set of all quadratic residues modulo n, i.e., $Q_n = \{a \mid a \in \mathbb{Z}_n^* : \exists x \in \mathbb{Z}_n^* \text{ s.t. } x^2 \equiv a \pmod n\}$.

2.4.1.2 One-way chains

A *one-way chain* C is a sequence $(x_0, x_1, x_2, \cdots x_n)$ defined by $x_i = f(x_{i-1})$, $1 \leq i \leq n$ where f is a one-way function as defined in Section 2.4.1.1. x_0 is said to be the (secret) initial seed of one-way chain C. Figure 2.5 depicts the construction of a one-way chain of length n. Note that the one-way chain elements are disclosed in reverse order of generation. Thus, having x_i, $0 < i \leq n$, x_j, $0 \leq j < i$ can be verified by computing $x_i = f^{i-j}(x_j)$. One-way chains are versatile: Lamport used one-way chains for one-time passwords [19] while Perrig *et al.* proposed a broadcast authentication mechanism based on one-way (key) chains [27].

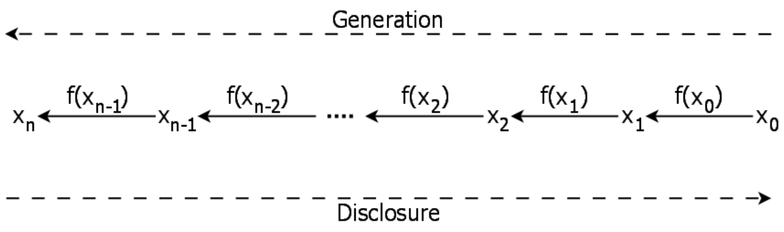


Figure 2.5: Construction of one-way chains according to [27]

2.4.1.3 Hash functions

A *cryptographic hash function* is a function $h : \{0,1\}^m \to \{0,1\}^n$ that maps an input x of arbitrary but finite bitlength m to an output $h(x)$ of fixed bitlength n [22]. Following properties are required by hash functions:

1. **Ease of computation –** Given input x, $h(x)$ is easy to compute.

[1]Note that the existence of one-way functions is not proven thus far. However, it is widely believed that one-way functions do exist.

2. **Preimage resistance –** Given output y, it is computationally infeasible to find any preimage x' such that $h(x') = y$.

3. **Second-preimage resistance –** Given input x, it is computationally infeasible to find any second input $x' \neq x$ such that $h(x) = h(x')$.

4. **Collision resistance –** It is computationally infeasible to find two arbitrary but distinct inputs x and x' such that $h(x) = h(x')$.

Note that in literature alternate terms *weak collision resistance* and *strong collision resistance* may be used for second-preimage resistance and collision resistance, respectively.

Hash functions are commonly used for data integrity and with digital signatures. If data integrity has to be ensured, the hash value of some input is computed and protected using appropriate techniques. To verify integrity of the input data, i.e., that the input data has not been modified, the hash value is recomputed and compared with the original one.

Digital signatures are costly for long messages. To ease the signature generation for long messages, the sender simply signs the message's hash value. Upon receipt, the hash value for the received message is computed and the signature is verified using the computed hash value.

Table 2.3 depicts some cryptographic hash functions and their hash and block sizes.

Algorithm	**Output size [bits]**	**Block size [bits]**
MD5	128	512
RIPEMD-128/256	128/256	512
SHA-1	160	512
WHIRLPOOL	512	512

Table 2.3: Some cryptographic hash functions

2.4.1.4 Hash-based message authentication code (HMAC)

Hash-based Message Authentication Code (HMAC) is a mechanism for message authentication using keys in combination with cryptographic hash functions as defined in Section 2.4.1.3 [16]. Let H denote the chosen hash function with block size B and output size L. Furthermore, let K be a secret key of length $n \leq B$. In [16], the minimal recommended length of s K is L, i.e., $L \leq n \leq B$. The HMAC of message m using key K is defined as HMAC$(K, m) = H((K \oplus \mathit{opad}) \,||\, H((K \oplus \mathit{ipad}) \,||\, m))$ where $\oplus$ denotes bit-wise exclusive-or and $||$ denotes concatenation of bitstrings. Operands *opad* and *ipad* are of length B and have values 0x363636 $\cdots$ 36 and 0x5C5C5C $\cdots$ 5C, respectively.

The specification of a certain hash function is called an *instantiation* of HMAC. For example, HMAC-Message Digest Algorithm Version 5 (MD5) and HMAC-Secure Hash Algorithm Version 1 (SHA-1) are the instantiations using hash functions MD5 and SHA-1, respecively. Note that HMAC instantiations are less affected by collisions than their underlying hash functions alone.

HMACs ensure message authentication and integrity, if the key is known by the sender and recipient of the message, only. Otherwise the HMAC needs to be protected, e.g. using a digital signature.

2.4.2 Symmetric-key cryptography

An encryption scheme, consisting of the sets of encryption and decryption transformations $\{E_e : e \in \mathcal{K}\}$ and $\{D_d : d \in \mathcal{K}\}$ with key space $\mathcal{K}$, is said to be a *symmetric-key* encryption scheme if for each key pair (e, d), d can be easily computed knowing only e and vice versa [22]. Note that in most symmetric-key encryption schemes $e = d$ holds.

Symmetric-key encryption schemes are mainly divided into *block* and *stream ciphers*. A block cipher with blocklength n is a function that maps n-bit blocks of a plaintext message $m = m_1 m_2 \cdots m_l$ to n-bit blocks of the message's ciphertext $c = c_1 c_2 \cdots c_l$, i.e., $E_K(m_j) = c_j$, $1 \leq j \leq l$ where $K \in \mathcal{K}$ is an arbitrary key. The inverse mapping together with K is used to decrypt the ciphertext, i.e., $D_K(c_j) = m_j$, $1 \leq j \leq l$. However, there exist various modes of operations with different properties. For example, the Cipher Block Chaining (CBC) mode of operation encrypts plaintext block m_j using the previously generated ciphertext block c_{j-1} as follows $c_j = E_K(m_j \oplus c_{j-1})$. The first ciphertext block c_1 is generated using some initialization vector IV, i.e., $c_0 = IV$.

In contrast to block ciphers, stream ciphers process plaintext in small blocks using a varying encryption function and a key stream $k_1 k_2 k_3 \cdots k_i \in \mathcal{K}$. However, the distinction between block and stream ciphers is not definite.

Since the computational effort for symmetric-key encryption schemes is far smaller than compared to public-key encryption schemes (see Section 2.4.3), symmetric-key cryptography seems to fit best for the use in WSANs. However, authentication can not be ensured using a symmetric key that is known by more than two entities.

AES [35] is one of the most widespread symmetric-key encryption schemes. It is a block cipher algorithm having a block size of 128 bits which, in the context of WSANs, is equal to its key size. Altough having relative high energy costs per byte, AES is generally regarded as secure choice when selecting ciphers for security schemes [38].

2.4.3 Public-key cryptography

An encryption scheme, consisting of the sets of encryption and decryption transformations $\{E_e : e \in \mathcal{K}\}$ and $\{D_d : d \in \mathcal{K}\}$ with key space $\mathcal{K}$, is said to be a *public-key* encryption scheme if for each key pair (e, d) where e is made publicly available and d is kept secret, it is computationally infeasible to derive d from e [22]. Subsequently, let K_{public} denote the public key e and $K_{private}$ denote the private key d.

To encrypt a message m for recipient A, the public key K^A_{public} of A is used to compute the ciphertext, i.e., $E_{K^A_{public}}(m) = c$. Thus, only A can decrypt the ciphertext using its private key $K^A_{private}$. The original message is obtained by $D_{K^A_{private}}(c) = m$. Besides encryption, public-key encryption schemes can be used for authentication and data integrity within a group of entities, too. If an integrity protected message m' is encrypted using the sender's private key

$K^S_{private}$, all recipients are able to decrypt the message using the sender's public key K^S_{public}. Since the private key $K^S_{private}$, associated to public key K^S_{public}, is only known to its generator, S is the true originator of m'. However, the main disadvantages of public-key encryption schemes are longer keys and higher computational effort compared to symmetric-key encryption schemes.

One of the most popular public-key encryption schemes is RSA which has been developed by Rivest, Shamir and Adleman [29]. The security of RSA relies on the integer factorization problem, i.e., given a positive integer n, find its prime factorization $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ where the p_i are pairwise distinct primes and each $e_i \geq 1$ [22]. However, to obtain a sufficient security level, RSA keys of at least 1024-bits length are recommended.

Another popular public-key encryption scheme is the Elliptic Curve Cryptography (ECC) scheme developed independently by Koblitz [15] and Miller [23]. The main advantage of ECC is that the same security level as RSA can be obtained using smaller keys. For example, a 160-bits ECC key provides the same security level as a 1024-bits RSA key [10].

Part II

Traffic management information model

CHAPTER 3

Information model specification

3.1 Overview

As discussed in Section 2.1, today's Traffic Management Systems (TMSs) follow a hierarchical structure. However, the centralized approach at the field level introduces a single point of failure. If an Local Control Unit (LCU) fails, its sensors and actuators cannot be reached from the higher levels anymore. Additionally, the wiring of sensors and actuators itself has disadvantages, too. It impedes the installation of new infrastructure and hence increases the costs of installation. Furthermore, it decreases the flexibility of TMS facilities. If the infrastructure for TMS facilities can be kept at a minimum, for example by using wireless technologies in combination with alternative energy supply such as photovoltaic cells, TMS facilities can cost-effectively be employed at arbitrary sites.

To counter these disadvantages, a decentralized approach with autonomously cooperating sensors and actuators based on wireless data communication has to be chosen. Interoperability among devices from different vendors is fundamental for such a decentralized system. This leads to the need of a generic information model for sensors and actuators used in TMSs. Figure 3.1 depicts how the field level can look like using the proposed information model below.

Communication between an LCU and devices of higher levels is standardized. However, only regional restricted (de-facto) standards exist. Examples are "Technische Lieferbedingungen für Streckenstationen" (TLS) [4], which is used in Germany, and National Transportation Communications for ITS Protocol (NTCIP) [1] used in the US. As seen in Figure 3.1, communication between sensors and actuators is independent from the protocol utilized for inter-level communication. It is also independent from underlying vendor specific protocols. Tethered sensors/actuators can be attached to gateways allowing them to join the wireless sensor network.

The scenario shown in Figure 3.1 illustrates that sensors and actuators can directly communicate with each other. This allows decentralized control by autonomously cooperating sensors and actuators at the field level.

The information model presented below is based on both, TLS [4] and NTCIP [1]. As

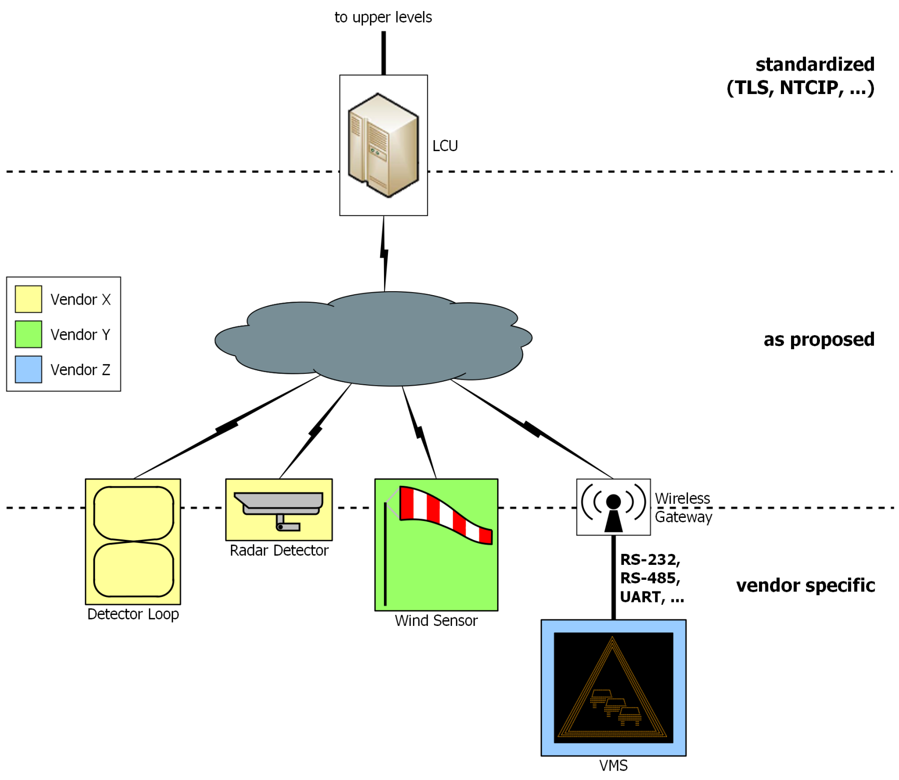


Figure 3.1: Proposed TMS field level structure

illustrated in Figure 3.2, it is structured on the basis of Functional Groups (FGs)[1]. Each FG represents a specific field of TMSs, such as collection of environmental data or Variable Message Sign (VMS) control. FGs are further divided into *clusters*. Clusters are identified by a 16-bit unsigned integer which has to be unique. While the higher byte encodes the FG the cluster belongs to, the lower byte identifies the cluster within the FG. The basic idea of using clusters is to form groups of attributes and commands that belong together semantically. Furthermore, communication between devices is ensured through so called *cluster bindings*. A cluster binding between two devices exists, if *both* of them support some cluster *X*. More precisely, one device has to support the *server side* of cluster *X* while the other one has to support the *client side*. The server side of a cluster retains attributes physically, e.g, parameters or measurement values, while the client side manipulates or retrieves them. Commands received at the server side have to be generated on the client side and vice versa.

Subsequent sections specify a generic information model for sensors and actuators used in TMSs. Each FG is described in detail, including their cluster definitions[2].

[1]Note that the classification and numbering of functional groups is not the same as defined in [4]
[2]Note that for the sake of clarity, the server side of clusters is depicted only

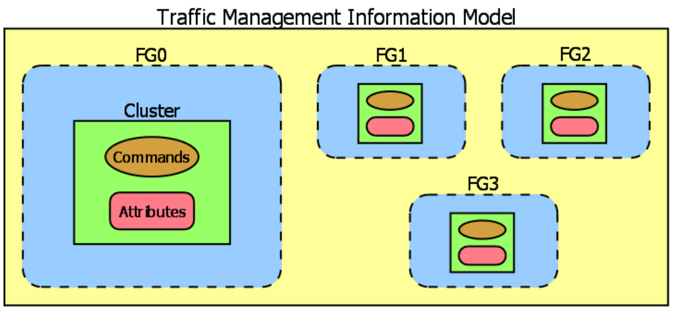


Figure 3.2: Structure of the traffic management information model

3.2 FG0 – General device configuration

FG0 comprises attributes and commands which are used for general configuration of devices. General configuration means that this group's clusters are applicable (and even mandatory) for every device, whether it is a simple temperature sensor or a more complex LCU. The clusters of FG0 are depicted in the following sections.

3.2.1 *General Device Information* cluster

The *General Device Information* cluster is mandatory for each device and keeps information regarding the whole device. It comprises both, management and process data. Since device configuration usually takes place at deployment, attributes representing management data are read-only. Subsequent modifications have to be done using appropriate *SetX* commands (see Section 3.2.1.2).

3.2.1.1 Attributes

The attributes of the *General Device Information* cluster are summarized in Table 3.1.

3.2.1.1.1 *Name* attribute The *Name* attribute is of type `Character String` and stores the device's name. Since it is necessary for identifying devices within the TMS, device names have to be unique.

3.2.1.1.2 *System Status* attribute The device's status is reflected by the `8-bit Enumeration` typed *System Status* attribute. Table 3.2 depicts the possible values of the system status enumeration.

3.2.1.1.3 *Power Source* attribute The *Power Source* attribute states how the device is provided with power. Its possible values are listed in Table 3.3.

Name	Type	Subtype	Access
Location	Struct		read
Latitude	Int32		
Longitude	Int32		
Reference Height	Int16		read
Sensors/Actuators	Array	Struct	read
Functional Group	UnsignedInt8		
Subgroup	UnsignedInt8		
Identifier	UnsignedInt16		
Sensor Height	Int16		
Status	Enum8		
System Status	Enum8		read
Power Source	Enum8		read
Battery	UnsignedInt8		read
Name	CharacterString		read
Subscriptions	Array	Struct	read
Cluster ID	UnsignedInt16		
Attributes	Array	Struct	
Attribute ID	UnsignedInt16		
Devices	Array	DeviceAddress	

Table 3.1: Attributes of the attribute *General Device Information* cluster

Value	System Status
0	Operational
1	Non-operational

Table 3.2: Values of the *System Status* enumeration

Value	Power Source
0	Grid
1	Battery-powered

Table 3.3: Values of the *Power Source* enumeration

3.2.1.1.4 ***Battery*** **attribute** For battery-powered devices, the *Battery* attribute indicates the state of charge in *percent*. It is of type `Unsigned 8-bit Integer` and ranges from 0 % to 100 %.

3.2.1.1.5 ***Location*** **attribute** The *Location* attribute stores the device's location. It is of type `Struct` and comprises two members, i.e. *Latitude* and *Longitude*. Both members are of type `Signed 32-bit Integer` and their unit is *degrees* (°). For the sake of precision, a resolution of 10^{-6} ° is used. This means that the stored value has to be multiplied by 10^{-6} to retrieve the correct value in degrees. *Latitude* ranges from $-90\,^\circ$ to $+90\,^\circ$, while *Longitude* ranges from $-180\,^\circ$ to $+180\,^\circ$.

3.2.1.1.6 *Reference Height* attribute In some cases it might be necessary to know the exact height of a sensor or actuator. This height can be calculated using the *Reference Height* attribute. The *Reference Height* is of type `Signed 16-bit Integer` and contains the height of the device's reference point with respect to the mean sea level in *m*.

3.2.1.1.7 *Sensors/Actuators* attribute The *Sensors/Actuators* attribute stores an array of all sensors and actuators of the device. Its elements are of type `Struct` and comprise five members describing a single sensor or actuator. Each sensor/actuator is identified by the *Functional Group*, *Subgroup* and *Identifier*. While *Functional Group* and *Subgroup* are of type `Unsigned 8-bit Integer`, *Identifier* is a `Unsigned 16-bit Integer`. The *Identifier* has to be unique within the given *Subgroup*. *Sensor Height* contains the height of the sensor/actuator relative to the device's reference point in *cm*. Finally, the status of the sensor/actuator can be retrieved from the `8-bit Enumeration` *Status*. Possible sensor statuses are shown in Table 3.4.

Value	Status
0	Operational
1	Erroneous

Table 3.4: Values of the *Status* enumeration

3.2.1.1.8 *Subscriptions* attribute The *Subscriptions* array maintains a list of all active reporting subscriptions. Each element represents a single cluster identified by the `Unsigned 16-bit Integer` *Cluster ID*. The records of the *Attributes* array comprise of the `Unsigned 16-bit Integer` typed *Attribute ID* identifying the attribute and a list of device addresses (*Devices*).

3.2.1.2 Commands

Table 3.5 depicts the commands of the *General Device Information* cluster. The first column states whether the command is received or generated.

3.2.1.2.1 *ReadAttributes* command The *ReadAttributes* command allows a client to retrieve the current values of attributes. Parameter *RA List* contains records of type `Struct`, each specifying a certain *Cluster ID* and a list of attribute identifiers, i.e., the *Attribute List* , that have to be read. If a device receives a *ReadAttributes* command, it generates a *ReadAttributesResponse* containing the current attribute values if present.

R/G	Name	Parameter	Type	Subtype
R	ReadAttributes	RA List	Array	Struct
		Cluster ID	UnsignedInt16	
		Attribute List	Array	UnsignedInt16
G	ReadAttributesResponse	RAR List	Array	Struct
		Cluster ID	UnsignedInt16	
		Data	Array	Struct
		Attribute ID	UnsignedInt16	
		Attribute Value	OctetString	
R	WriteAttributes	WA List	Array	Struct
		Cluster ID	UnsignedInt16	
		Data	Array	Struct
		Attribute ID	UnsignedInt16	
		Attribute Value	OctetString	
G	WriteAttributesResponse	WAR List	Array	Struct
		Cluster ID	UnsignedInt16	
		Data	Array	Struct
		Attribute ID	UnsignedInt16	
		Status	Enum8	
R	SetName	newName	CharacterString	
R	SetLocation	newLatitude	Int32	
		newLongitude	Int32	
R	SetReferenceHeight	newReferenceHeight	Int16	
R	GetSensorStatus	Functional Group	UnsignedInt8	
		Subgroup	UnsignedInt8	
		Identifier	UnsignedInt16	
G	GetSensorStatusResponse	Status	Enum8	
R	SubscribeReporting	Reporting List	Array	Struct
		Cluster ID	UnsignedInt16	
		Attribute List	Array	UnsignedInt16
R	UnsubscribeReporting	Cluster List	Array	UnsignedInt16

Table 3.5: Commands of the *General Device Information* cluster

3.2.1.2.2 ***ReadAttributesResponse* command** The *ReadAttributesResponse* is generated after receipt of a *ReadAttributes* command. It returns the requested attribute values to the client. Parameter *RAR List* is an array of type `Struct`. Each record of the *RAR List* contains a *Cluster ID* and a *Data* array which in turn comprises pairs of *Attribute ID* and *Attribute Value*. The format of the *Attribute Value* element depends on the attribute and thus is determined by the *Attribute ID*.

3.2.1.2.3 *WriteAttributes* command The *WriteAttributes* command allows a client to write attribute values. Each record of the *WA List* parameter contains a *Cluster ID* and a *Data* array specifying the new attribute values that have to be written. If a device receives a *WriteAttributes* command, it generates a *WriteAttributesResponse* containing the status for each written attribute.

3.2.1.2.4 *WriteAttributesResponse* command The *WriteAttributesResponse* is generated after receipt of a *WriteAttributes* command. It returns the status for each written attribute value to the client. Parameter *WAR List* is an array of type `Struct`. Each record of the *WAR List* contains a *Cluster ID* and a *Data* array which in turn comprises pairs of *Attribute ID* and *Status*. The *Status* element is of type `8-bit Enumeration` and indicates whether or not the attribute has been written successfully.

3.2.1.2.5 *SetName* command The *SetName* command sets the device's *Name* to *newName*.

3.2.1.2.6 *SetLocation* command The device's *Location* can be modified using the *SetLocation* command. It has two parameters of type `Signed 32-bit Integer`, i.e. *newLatitude* and *newLongitude*. Their format is described in Section 3.2.1.1.5.

3.2.1.2.7 *SetReferenceHeight* command To alter the device's *ReferenceHeight*, the *SetReferenceHeight* command can be used. Its only parameter is the `Signed 16-bit Integer` typed *newReferenceHeight*. It contains the new height of the device's reference point with respect to the mean sea level in *m*.

3.2.1.2.8 *GetSensorStatus* command The *GetSensorStatus* command allows a client to retrieve the status of a certain sensor or actuator. Parameters *Functional Group*, *Subgroup* and *Identifier* specify the sensor or actuator whose status has to be returned. If a device receives a *GetSensorStatus* command, it generates a *GetSensorStatusResponse* containing the status of the sensor or actuator.

3.2.1.2.9 *GetSensorStatusResponse* command The *GetSensorStatusResponse* is generated after receipt of a *GetSensorStatus* command. It returns the *Status* of a sensor or actuator to a requesting client. The parameter *Status* is of type `8-bit Enumeration`. Its possible values are depicted in Table 3.4.

3.2.1.2.10 *SubscribeReporting* command Clients can subscribe to reporting of measurement values using the *SubscribeReporting* command. The *Reporting List* parameter contains records of type `Struct`, each specifying a cluster and a list of attribute identifiers (i.e., the *Attribute List*) that have to be reported. Valid subscriptions are appended to the *Subscriptions* array. A detailed description of the reporting mechanism can be found in Section 3.3.1.1.3.

3.2.1.2.11 *UnsubscribeReporting* command The *UnsubscribeReporting* command can be used to unsubscribe from reporting of measurement values. Unsubscribing is only supported for whole clusters. Parameter *Cluster List* contains a list of *Unsigned 16-bit Integer* cluster identifiers. The client's corresponding entries have to be removed from the *Subscriptions* array.

3.3 FG1 – Traffic data acquisition

FG1 covers all attributes and commands which are necessary for acquisition of traffic data of all kind. The three main areas of traffic data are load statistics, traffic statistics and speeding statistics. Table 3.6 depicts the clusters of FG1 which are described in detail in the following sections.

Cluster name
General Configuration – Traffic Data Acquisition
Vehicle Measurement – Configuration
Vehicle Measurement
Load Statistics – Configuration
Load Statistics – Short-Term Measurement
Load Statistics – Long-Term Measurement
Traffic Statistics – Configuration
Traffic Statistics – Short-Term Measurement
Traffic Statistics – Long-Term Measurement
Speeding Statistics – Configuration
Speeding Statistics – Short-Term Measurement
Speeding Statistics – Long-Term Measurement

Table 3.6: Clusters of FG1

3.3.1 *General Configuration – Traffic Data Acquisition* cluster

The *General Configuration – Traffic Data Acquisition* cluster is mandatory for every device that has to gather traffic data.

3.3.1.1 Attributes

Table 3.7 shows the attributes of the *General Configuration – Traffic Data Acquisition* cluster.

3.3.1.1.1 *Measurement Period – Short-Term Data* attribute The *Measurement Period – Short-Term Data* gives the measurement period for short-term traffic data acquisition in *seconds*. It is of type `Unsigned 8-bit Integer` and has a resolution of 30 s. With that, values from 30 s to 7200 s are possible.

Name	Type	Access
Measurement Period – Short-Term Data	UnsignedInt8	read/write
Measurement Period – Long-Term Data	UnsignedInt16	read/write
Operating Mode	Enum8	read/write

Table 3.7: Attributes of the *General Configuration – Traffic Data Acquisition* cluster

3.3.1.1.2 ***Measurement Period – Long-Term Data*** **attribute** As for short-term traffic data, the measurement period for long-term traffic data acquisition can be configured. *Measurement Period – Long-Term Data* is of type `Unsigned 16-bit Integer` and states the measurement period in *hours*. It has a resolution of 1 h, ranging from 2 h to 672 h.

3.3.1.1.3 ***Operating Mode*** **attribute** This attribute specifies the operating mode for sensors acquiring traffic data. Table 3.8 depicts the possible operating modes. In *Reporting* mode, measured values are reported to subscribed clients automatically at the end of each measurement period. If *Polling* mode is chosen, clients have to request measured values by sending appropriate commands.

Value	Operating Mode
0	Reporting
1	Polling

Table 3.8: Values of the *Operating Mode* enumeration for sensors

3.3.2 *Vehicle Measurement – Configuration* cluster

The *Vehicle Measurement – Configuration* cluster defines parameters that are necessary to properly detect and classify vehicles.

3.3.2.1 Attributes

The attributes of the *Vehicle Measurement – Configuration* cluster are depicted in Table 3.9.

3.3.2.1.1 ***Maximum Axle Distance – Twin Axle*** **attribute** For the detection of twin axles, the *Maximum Axle Distance – Twin Axle* attribute defines the maximum allowed distance between the two axes of a twin axle in *cm*. If the maximum allowed distance between two subsequent axles is exceeded, two single axles should be detected rather than a twin axle. The *Maximum Axle Distance – Twin Axle* attribute is of type `Unsigned 8-bit Integer` and ranges from 20 cm to 200 cm.

3.3.2.1.2 ***Maximum Axle Distance – Triple Axle*** **attribute** Similar to the *Maximum Axle Distance – Twin Axle* attribute, the *Maximum Axle Distance – Triple Axle* attribute specifies the

Name	**Type**	**Subtype**	**Access**
Maximum Axle Distance – Twin Axle	UnsignedInt8		read/write
Maximum Axle Distance – Triple Axle	UnsignedInt8		read/write
Maximum Loads – By Vehicle Class	Array	Struct	read/write
Vehicle Class Code	Enum8		
Maximum Total Load	UnsignedInt16		
Maximum Axle Loads – By Axle Class	Array	Struct	
Axle Class Code	Enum8		
Maximum Axle Load	UnsignedInt16		

Table 3.9: Attributes of the *Vehicle Measurement – Configuration* cluster

maximum allowed distance between each two subsequent axles of a triple axle in *cm*. It is of type `Unsigned 8-bit Integer` and ranges from 20 cm to 150 cm.

3.3.2.1.3 ***Maximum Loads – By Vehicle Class*** **attribute** The *Maximum Loads – By Vehicle Class* array stores the maximum total load as well as the maximum axle loads for each vehicle class. *Maximum Total Load* is of type `Unsigned 16-bit Integer` and its unit is *kg*. The *Maximum Axle Loads – By Axle Class* array defines the *Maximum Axle Load* for each type of axle. Maximum axle loads are given in *kg*.

3.3.3 *Vehicle Measurement* cluster

The *Vehicle Measurement* cluster is mandatory for each device that collects traffic data. However, depending on the data to be gathered, not all attributes need to be present.

3.3.3.1 Attributes

Table 3.10 summarizes the attributes of the *Vehicle Measurement* cluster.

3.3.3.1.1 ***Vehicle Class Code*** **attribute** The `8-bit Enumeration` typed *Vehicle Class Code* attribute indicates the vehicle class of the detected vehicle. Its possible values are depicted in Table 3.11. Vehicle class *Other* has to be applied for vehicles that could not be classified uniquely.

3.3.3.1.2 ***Vehicle Length*** **attribute** The *Vehicle Length* attribute stores the length of the detected vehicle in *cm*. It is of type `Unsigned 16-bit Integer`.

3.3.3.1.3 ***Vehicle Height*** **attribute** *Vehicle Height* is of type `Unsigned 16-bit Integer` and records the vehicle's height in *cm*.

3.3.3.1.4 ***Vehicle Width*** **attribute** The width of the vehicle is also measured in *cm*. It is stored in the *Vehicle Width* attribute, which is of type `Unsigned 16-bit Integer`.

Name	Type	Subtype	Access
Vehicle Class Code	Enum8		read
Vehicle Length	UnsignedInt16		read
Vehicle Height	UnsignedInt16		read
Vehicle Width	UnsignedInt16		read
Vehicle Velocity	UnsignedInt16		read
Vehicle Distance	UnsignedInt16		read
Vehicle Total Load	UnsignedInt16		read
Vehicle Excess Load	Boolean		read
Vehicle Axle Load Data	Array	Struct	read
Axle Class Code	Enum8		
Axle Load	UnsignedInt16		
Excess Load	Boolean		
Axle Distance	UnsignedInt16		

Table 3.10: Attributes of the *Vehicle Measurement* cluster

Value	Vehicle Class
0	Other
1	Motorcycle
2	Motor car
3	Van
4	Motor car with trailer
5	Truck
6	Truck with trailer
7	Articulated truck
8	Bus

Table 3.11: Values of the *Vehicle Class* enumeration according to [4]

3.3.3.1.5 *Vehicle Velocity* attribute *Vehicle Velocity* states the vehicle's velocity. This attribute is of type `Unsigned 16-bit Integer` and has a resolution of 0.01 km/h.

3.3.3.1.6 *Vehicle Distance* attribute The *Vehicle Distance* attribute states the distance of the vehicle's first detected axle to the last detected axle of the previous vehicle in *dm*. It is of type `Unsigned 16-bit Integer`. A value of 0 indicates that either no preceding vehicle has been detected or the distance to the previously detected vehicle exceeds the maximum value.

3.3.3.1.7 *Vehicle Total Load* attribute The *Vehicle Total Load* attribute stores the vehicle's total load in *kg*. It is of type `Unsigned 16-bit Integer`

3.3.3.1.8 *Vehicle Excess Load* attribute

The `Boolean` attribute *Vehicle Excess Load* indicates whether or not the vehicle's total load exceeds the maximum total load for the detected vehicle class.

3.3.3.1.9 *Vehicle Axle Load Data* attribute

The *Vehicle Axle Load Data* array contains one record for each detected axle of the vehicle. *Axle Class Code* states the type of axle. Tabe 3.12 shows the different types of axles defined. While *Axle Load* stores the axle's load in *kg*, *Excess Load* indicates whether or not the axle's load exceeds the maximum axle load for the given type of axle and the detected vehicle class. *Axle Distance* states the distance to the previously detected axle in *cm*. A value of 0 indicates that either no preceding axle has been detected or the distance to the previously detected axle exceeds the maximum value.

Value	Axle Class
0	Other
1	Single axle
2	Twin axle
3	Triple axle

Table 3.12: Values of the *Axle Class* enumeration according to [4]

3.3.3.2 Commands

The commands of the *Vehicle Measurement* cluster are shown in Table 3.13.

R/G	Name	Parameter	Type	Subtype
G	ReportVehicle	Data	Array	Struct
		Attribute ID	UnsignedInt16	
		Attribute Value	OctetString	

Table 3.13: Commands of the *Vehicle Measurement* cluster

3.3.3.2.1 *ReportVehicle* command

The *ReportVehicle* command is used to notify subscribed devices about a detected vehicle. Its only parameter is the *Data* array of type `Struct`. Each record of the *Data* array is a pair of *Attribute ID* and *Attribute Value*. The format of the *Attribute Value* element depends on the attribute and thus is determined by the *Attribute ID*.

3.3.4 *Load Statistics – Configuration* cluster

The *Load Statistics – Configuration* cluster comprises parameters that are necessary for the collection of short- or long-term load statistics.

3.3.4.1 Attributes

The attributes of the *Load Statistics – Configuration* cluster are depicted in Table 3.14.

Name	Type	Subtype	Access
Total Load Classes	Array	Struct	read/write
Vehicle Class Code	Enum8		
Load Boundaries	Array	UnsignedInt16	
Axle Load Classes	Array	Struct	read/write
Axle Class Code	Enum8		
Load Boundaries	Array	UnsignedInt16	

Table 3.14: Attributes of the *Load Statistics – Configuration* cluster

3.3.4.1.1 ***Total Load Classes*** **attribute** This attribute defines total load classes for each vehicle class. Total load classes are left-open intervals specified by a list of boundaries, i.e. the *Load Boundaries* array. Load boundaries are of type `Unsigned 16-bit Integer` and are given in *kg*. To form proper load classes, the values of the *Load Boundaries* array must be strictly increasing. Since the value of 0 is used as left bound of the first load class, the *Load Boundaries* array must not contain the value 0. The last load class is built using ∞ as right bound. For example: the *Load Boundaries* array $[3500]$ leads to two total load classes, i.e. $C_1 = (0, 3500]$ and $C_2 = (3500, \infty]$.

3.3.4.1.2 ***Axle Load Classes*** **attribute** As for total load classes, the *Axle Load Classes* array specifies axle load classes for each axle class. Axle load classes are formed by the *Load Boundaries* array. Load boundaries are of type `Unsigned 16-bit Integer` and are stated in *kg*.

3.3.5 *Load Statistics – Short-Term Measurement* cluster

The *Load Statistics – Short-Term Measurement* cluster provides short-term load statistics.

3.3.5.1 Attributes

Table 3.15 depicts the attributes of the *Load Statistics – Short-Term Measurement* cluster.

3.3.5.1.1 ***Total Load Count – By Vehicle Class*** **attribute** The *Total Load Count – By Vehicle Class* array keeps an excess loads counter and counters for each total load class for every vehicle class. *Vehicle Class Excess Loads* is of type `Unsigned 8-bit Integer`, counting the number of excess loads belonging to the given vehicle class. The *Total Load Class Count* array contains counters of type `Unsigned 16-bit Integer` for each total load class of the given vehicle class. For the definition of total load classes refer to Section 3.3.4.1.1.

Name	Type	Subtype	Access
Total Load Count – By Vehicle Class	Array	Struct	read
Vehicle Class Code	Enum8		
Vehicle Class Excess Loads	UnsignedInt8		
Total Load Class Count	Array	UnsignedInt16	
Axle Load Count – By Axle Class	Array	Struct	read
Axle Class Code	Enum8		
Axle Class Excess Loads	UnsignedInt8		
Axle Load Class Count	Array	UnsignedInt16	

Table 3.15: Attributes of the *Load Statistics – Short-Term Measurement* cluster

3.3.5.1.2 *Axle Load Count – By Axle Class* attribute Similar to the *Total Load Count – By Vehicle Class* array, the *Axle Load Count – By Axle Class* array keeps an excess loads counter and counters for each axle load class for every type of axle. *Axle Class Excess Loads* is of type `Unsigned 8-bit Integer`, counting the number of excess loads belonging to the given axle class. The *Axle Load Class Count* array contains counters of type `Unsigned 16-bit Integer` for each axle load class of the given axle class. For the definition of axle load classes refer to Section 3.3.4.1.2.

3.3.5.2 Commands

The commands of the *Load Statistics – Short-Term Measurement* cluster are shown Table 3.16.

R/G	Name	Parameter	Type	Subtype
R	GetLoadStatistics	Attribute List	Array	UnsignedInt16
G	ReportLoadStatistics	Data	Array	Struct
		Attribute ID	UnsignedInt16	
		Attribute Value	OctetString	

Table 3.16: Commands of the *Load Statistics – Short-Term Measurement* cluster

3.3.5.2.1 *GetLoadStatistics* command The *GetLoadStatistics* command can be used to retrieve short-term load statistics. Parameter *Attribute List* allows a client to query attribute values selectively. If a device receives a *GetLoadStatistics* command, it generates a *ReportLoadStatistics* command including the attributes specified by the *Attribute List* parameter.

3.3.5.2.2 *ReportLoadStatistics* command Short-term load statistics are transferred using the *ReportLoadStatistics* command. This command is generated either after receipt of a *GetLoadStatistics* command or in consequence of a expired measurement period. The format of the *Data* parameter is the same as described in Section 3.3.3.2.1.

3.3.6 *Load Statistics – Long-Term Measurement* cluster

The *Load Statistics – Long-Term Measurement* cluster provides long-term load statistics.

3.3.6.1 Attributes

Table 3.17 summarizes the attributes of the *Load Statistics – Long-Term Measurement* cluster.

Name	Type	Subtype	Access
Total Load Count – By Vehicle Class	Array	Struct	read
Vehicle Class Code	Enum8		
Vehicle Class Excess Loads	UnsignedInt16		
Total Load Class Count	Array	UnsignedInt32	
Axle Load Count – By Axle Class	Array	Struct	read
Axle Class Code	Enum8		
Axle Class Excess Loads	UnsignedInt16		
Axle Load Class Count	Array	UnsignedInt32	

Table 3.17: Attributes of the *Load Statistics – Long-Term Measurement* cluster

3.3.6.1.1 *Total Load Count – By Vehicle Class* attribute The *Total Load Count – By Vehicle Class* array keeps an excess loads counter and counters for each total load class for every vehicle class. *Vehicle Class Excess Loads* is of type `Unsigned 16-bit Integer`, counting the number of excess loads belonging to the given vehicle class. The *Total Load Class Count* array contains counters of type `Unsigned 32-bit Integer` for each total load class of the given vehicle class. For the definition of total load classes refer to Section 3.3.4.1.1.

3.3.6.1.2 *Axle Load Count – By Axle Class* attribute Similar to the *Total Load Count – By Vehicle Class* array, the *Axle Load Count – By Axle Class* array keeps an excess loads counter and counters for each axle load class for every type of axle. *Axle Class Excess Loads* is of type `Unsigned 16-bit Integer`, counting the number of excess loads belonging to the given axle class. The *Axle Load Class Count* array contains counters of type `Unsigned 32-bit Integer` for each axle load class of the given axle class. For the definition of axle load classes refer to Section 3.3.4.1.2.

3.3.6.2 Commands

The commands of the *Load Statistics – Long-Term Measurement* cluster are shown in Table 3.18.

3.3.6.2.1 *GetLoadStatistics* command Long-term load statistics can be queried using the *GetLoadStatistics* command. The *Attribute List* parameter specifies which attribute values should be retrieved.

R/G	Name	Parameter	Type	Subtype
R	GetLoadStatistics	Attribute List	Array	UnsignedInt16
G	ReportLoadStatistics	Data	Array	Struct
		Attribute ID	UnsignedInt16	
		Attribute Value	OctetString	

Table 3.18: Commands of the *Load Statistics – Long-Term Measurement* cluster

3.3.6.2.2 ***ReportLoadStatistics* command** The *ReportLoadStatistics* command is used to report long-term load statistics. It is generated either after receipt of a *GetLoadStatistics* command or in consequence of a expired measurement period. The format of the *Data* parameter is the same as described in Section 3.3.3.2.1.

3.3.7 *Traffic Statistics – Configuration* cluster

The *Traffic Statistics – Configuration* cluster defines parameters for gathering short- and long-term traffic statistics.

3.3.7.1 Attributes

The attributes of the *Traffic Statistics – Configuration* cluster are summarized in Table 3.19.

Name	Type	Subtype	Access
Averaging Mode	Enum8		read/write
Speed Classes	Array	Struct	read/write
Vehicle Class Code	Enum8		
Speed Boundaries	Array	UnsignedInt8	

Table 3.19: Attributes of the *Traffic Statistics – Configuration* cluster

3.3.7.1.1 ***Averaging Mode* attribute** The *Averaging Mode* defines how the average velocity is computed. Possible values are depicted in Table 3.20.

Value	Averaging mode
0	Arithmetic mean
1	Median

Table 3.20: Values of the *Averaging Mode* enumeration

3.3.7.1.2 ***Speed Classes* attribute** The *Speed Classes* array defines speed classes used for short- and long-term traffic data statistics. Speed classes are left-open intervals specified by a list of boundaries, i.e. the *Speed Boundaries* array. Speed boundaries are of type `Unsigned`

`8-bit Integer` and are given in *km/h*. To form proper speed classes, the values of the *Speed Boundaries* array must be strictly increasing. Since the value of 0 is used as left bound of the first speed class, the *Speed Boundaries* array must not contain the value 0. The last speed class is built using ∞ as right bound. For example: the *Speed Boundaries* array $[50, 100]$ leads to three speed classes, i.e. $C_1 = (0, 50]$, $C_2 = (50, 100]$, $C_3 = (100, \infty]$.

3.3.8 *Traffic Statistics – Short-Term Measurement* cluster

The *Traffic Statistics – Short-Term Measurement* cluster provides attributes for short-term traffic statistics.

3.3.8.1 Attributes

Table 3.21 shows the attributes of the *Traffic Statistics – Short-Term Measurement* cluster.

Name	Type	Subtype	Access
Traffic Intensity – Overall	UnsignedInt16		read
Traffic Intensity – By Vehicle Class	Array	Struct	read
Vehicle Class Code	Enum8		
Vehicle Class Intensity	UnsignedInt16		
Speed Class Intensity	Array	UnsignedInt16	
Average Velocity – Overall	UnsignedInt16		read
Average Velocity – By Vehicle Class	Array	Struct	read
Vehicle Class Code	Enum8		
Vehicle Class Average Velocity	UnsignedInt16		

Table 3.21: Attributes of the *Traffic Statistics – Short-Term Measurement* cluster

3.3.8.1.1 ***Traffic Intensity – Overall* attribute** The *Traffic Intensity – Overall* attribute is a counter of type `Unsigned 16-bit Integer`. It stores the number of vehicles, passing the controlled section, per time interval (see Section 3.3.1.1.1).

3.3.8.1.2 ***Traffic Intensity – By Vehicle Class* attribute** For more detailed statistics, the *Traffic Intensity – By Vehicle Class* attribute keeps a counter for each vehicle class. The *Vehicle Class Intensity* is of type `Unsigned 16-bit Integer`, counting the number of vehicles belonging to a certain vehicle class, only. Furthermore, counters for each speed class of the given vehicle class exist. For the definition of speed classes refer to Section 3.3.7.1.2.

3.3.8.1.3 ***Average Velocity – Overall* attribute** The *Average Velocity – Overall* attribute stores the average velocity of all vehicles over the measurement period (see Section 3.3.1.1.1). It is of type `Unsigned 16-bit Integer` and has a resolution of $0.01\,\mathrm{km/h}$. Computation of the average velocity depends on the value of the *Averaging Mode* attribute (see Section 3.3.7.1.1).

3.3.8.1.4 ***Average Velocity – By Vehicle Class*** **attribute** As for traffic intensity counters, the average velocity can also be retrieved for each vehicle class itself. The *Vehicle Class Average Velocity* records the average velocity of the given vehicle class. It is of type `Unsigned 16-bit Integer` and has a resolution of $0.01\,\mathrm{km/h}$.

3.3.8.2 Commands

In Table 3.22, the commands of the *Traffic Statistics – Short-Term Measurement* cluster are depicted.

R/G	Name	Parameter	Type	Subtype
R	GetTrafficStatistics	Attribute List	Array	UnsignedInt16
G	ReportTrafficStatistics	Data	Array	Struct
		Attribute ID	UnsignedInt16	
		Attribute Value	OctetString	

Table 3.22: Commands of the *Traffic Statistics – Short-Term Measurement* cluster

3.3.8.2.1 ***GetTrafficStatistics*** **command** The *GetTrafficStatistics* command can be used to retrieve short-term traffic statistics. The parameter *Attribute List* allows a client to query attribute values selectively.

3.3.8.2.2 ***ReportTrafficStatistics*** **command** Short-term traffic statistics are transferred using the *ReportTrafficStatistics* command. This command is generated either after receipt of a *GetTrafficStatistics* command or in consequence of a expired measurement period. The format of the *Data* parameter is the same as described in Section 3.3.3.2.1.

3.3.9 *Traffic Statistics – Long-Term Measurement* cluster

The *Traffic Statistics – Long-Term Measurement* cluster provides long-term traffic statistics.

3.3.9.1 Attributes

Table 3.23 summarizes the attributes of the *Traffic Statistics – Long-Term Measurement* cluster.

3.3.9.1.1 ***Traffic Intensity – Overall*** **attribute** In contrast to the *Traffic Intensity – Overall* attribute for short-term traffic statistics (see Section 3.3.8.1.1, the *Traffic Intensity – Overall* attribute for long-term traffic statistics is a counter of type `Unsigned 32-bit Integer`. It stores the number of vehicles, passing the controlled section, per time interval (see Section 3.3.1.1.2).

Name	Type	Subtype	Access
Traffic Intensity – Overall	UnsignedInt32		read
Traffic Intensity – By Vehicle Class	Array	Struct	read
Vehicle Class Code	Enum8		
Vehicle Class Intensity	UnsignedInt32		
Speed Class Intensity	Array	UnsignedInt32	
Average Velocity – Overall	UnsignedInt16		read
Average Velocity – By Vehicle Class	Array	Struct	read
Vehicle Class Code	Enum8		
Vehicle Class Average Velocity	UnsignedInt16		

Table 3.23: Attributes of the *Traffic Statistics – Long-Term Measurement* cluster

3.3.9.1.2 ***Traffic Intensity – By Vehicle Class* attribute** The *Traffic Intensity – By Vehicle Class* attribute for long-term traffic statistics is similar to the one for short-term traffic statistics (see Section 3.3.8.1.2). They only differentiate by their counter size. The long-term version uses counters of type `Unsigned 32-bit Integer` for *Vehicle Class Intensity* and each single speed class counter.

3.3.9.1.3 ***Average Velocity – Overall* attribute** The *Average Velocity – Overall* attribute stores the average velocity of all vehicles over the measurement period (see Section 3.3.1.1.1). It is of type `Unsigned 16-bit Integer` and has a resolution of 0.01 km/h. Computation of the average velocity depends on the value of the *Averaging Mode* attribute (see Section 3.3.7.1.1).

3.3.9.1.4 ***Average Velocity – By Vehicle Class* attribute** As for traffic intensity counters, the average velocity can also be retrieved for each vehicle class itself. The *Vehicle Class Average Velocity* records the average velocity of the given vehicle class. It is of type `Unsigned 16-bit Integer` and has a resolution of 0.01 km/h.

3.3.9.2 Commands

The commands of the *Traffic Statistics – Long-Term Measurement* cluster are shown in Table 3.24.

R/G	Name	Parameter	Type	Subtype
R	GetTrafficStatistics	Attribute List	Array	UnsignedInt16
G	ReportTrafficStatistics	Data	Array	Struct
		Attribute ID	UnsignedInt16	
		Attribute Value	OctetString	

Table 3.24: Commands of the *Traffic Statistics – Long-Term Measurement* cluster

3.3.9.2.1 ***GetTrafficStatistics* command** Long-term load statistics can be queried using the *GetTrafficStatistics* command. The *Attribute List* parameter specifies which attribute values should be retrieved.

3.3.9.2.2 ***ReportTrafficStatistics* command** The *ReportTrafficStatistics* command is used to transfer long-term traffic statistics. It is generated either after receipt of a *GetTrafficStatistics* command or in consequence of a expired measurement period. The format of the *Data* parameter is the same as described in Section 3.3.3.2.1.

3.3.10 *Speeding Statistics – Configuration* cluster

The *Speeding Statistics – Configuration* cluster specifies parameters for the collection of short- and long-term speeding statistics.

3.3.10.1 Attributes

In Table 3.25, the attributes of the *Speeding Statistics – Configuration* cluster are summarized.

Name	Type	Subtype	Access
Settings – Front Camera	UnsignedInt8		read/write
Settings – Rear Camera	UnsignedInt8		read/write
Observe Restriction On Passing For Trucks	Boolean		read/write
VMS Speed Limit Active	Boolean		read/write
VMS Speed Limit	UnsignedInt8		read/write
Speed Limits – By Vehicle Class	Array	Struct	read/write
Vehicle Class Code	Enum8		
Maximum Allowed Velocity	UnsignedInt8		
Triggering Velocity	UnsignedInt8		
Speed Limits – By Variable Message Sign	Array	Struct	read/write
Maximum Allowed Velocity	UnsignedInt8		
Triggering Velocity	UnsignedInt8		

Table 3.25: Attributes of the *Speeding Statistics – Configuration* cluster

3.3.10.1.1 ***Settings – Front Camera* attribute** The *Settings – Front Camera* attribute is a bit field of type `Unsigned 8-bit Integer`. It allows to specify the triggering of the front camera, i.e., the camera which is directed against the driving direction. Table 3.26 lists the flags that are available for configuration of the camera. If the *Store pictures* flag is set, pictures have to be stored. Flags *Main flash* and *Additional Flash* can be used to activate the corresponding flash. All other flags are reserved for future use and should be set to zero.

Bit	Flag
0	Store pictures
1	Main flash
2	Additional flash
3 . . . 7	*reserved*

Table 3.26: Flags for the configuration of cameras according to [4]

3.3.10.1.2 ***Settings – Rear Camera* attribute** As for the front camera, the *Settings – Rear Camera* attribute allows to specify the triggering of rear camera. The rear camera is mounted in driving direction. For the functional description of this attribute refer to Section 3.3.10.1.1.

3.3.10.1.3 ***Observe Restrictions On Passing For Trucks* attribute** This attribute indicates whether or not restrictions on passing for trucks should be observed. If this attribute is set to *true*, the cameras have to be triggered according to their mode.

3.3.10.1.4 ***VMS Speed Limit Active* attribute** The *VMS Speed Limit Active* attribute indicates whether or not a certain speed limit signaled by a VMS is active. If this attribute is set to *true*, a general speed limit is specified by the *VMS Speed Limit* attribute. Otherwise, the vehicle class specific speed limits as defined by the *Speed Limits – By Vehicle Class* attribute are valid.

3.3.10.1.5 ***VMS Speed Limit* attribute** The *VMS Speed Limit* attribute defines a general speed limit signaled by a VMS. It is of type `Unsigned 8-bit Integer` and has a resolution of $1\,\mathrm{km/h}$.

3.3.10.1.6 ***Speed Limits – By Vehicle Class* attribute** Vehicle class specific speed limits are defined by the *Speed Limits – By Vehicle Class* array. Each record consists of the *Vehicle Class Code* (see Table 3.11), the *Maximum Allowed Velocity* and the *Triggering Velocity*. The latter two elements are of type `Unsigned 8-bit Integer`. They are given in *km/h* with a resolution of $1\,\mathrm{km/h}$. The triggering velocity defines the minimum velocity at which the cameras are triggered.

3.3.10.1.7 ***Speed Limits – By VMS* attribute** If *VMS Speed Limit Active* is set to *true*, vehicle class specific speed limits are overruled by a general speed limit as specified by the *VMS Speed Limit* attribute. The *Speed Limits – By VMS* array stores possible speed limits and their triggering velocities. Both, *Maximum Allowed Velocity* and *Triggering Velocity* are given in *km/h* with a resolution of $1\,\mathrm{km/h}$.

3.3.11 *Speeding Statistics – Short-Term Measurement* cluster

The *Speeding Statistics – Short-Term Measurement* cluster provides short-term speeding statistics.

3.3.11.1 Attributes

The attributes of the *Speeding Statistics – Short-Term Measurement* cluster are shown in Table 3.27.

Name	**Type**	**Subtype**	**Access**
Trigger Count – Front Camera	UnsignedInt16		read
Trigger Count – Rear Camera	UnsignedInt16		read
Pictures – Front Camera	Array	OctetString	read
Pictures – Rear Camera	Array	OctetString	read

Table 3.27: Attributes of the *Speeding Statistics – Short-Term Measurement* cluster

3.3.11.1.1 *Trigger Count – Front Camera* attribute The *Trigger Count – Front Camera* attribute is of type `Unsigned 16-bit Integer`. It is incremented every time the front camera is triggered.

3.3.11.1.2 *Trigger Count – Rear Camera* attribute The `Unsigned 16-bit Integer` typed *Trigger Count – Rear Camera* attribute counts how often the rear camera has been triggered.

3.3.11.1.3 *Pictures – Front Camera* attribute The *Pictures – Front Camera* array stores all pictures that have been made using the front camera. Its elements are of type `Octet String`.

3.3.11.1.4 *Pictures – Rear Camera* attribute Like the *Pictures – Front Camera* array, the *Pictures – Rear Camera* array stores all pictures that have been made using the rear camera.

3.3.11.2 Commands

Table 3.28 depicts the commands of the *Speeding Statistics – Short-Term Measurement* cluster.

R/G	**Name**	**Parameter**	**Type**	**Subtype**
R	GetSpeedingStatistics	Attribute List	Array	UnsignedInt16
G	ReportSpeedingStatistics	Data	Array	Struct
		Attribute ID	UnsignedInt16	
		Attribute Value	OctetString	

Table 3.28: Commands of the *Speeding Statistics – Short-Term Measurement* cluster

3.3.11.2.1 *GetSpeedingStatistics* command The *GetSpeedingStatistics* command can be used to retrieve short-term speeding statistics. Parameter *Attribute List* allows a client to query attribute values selectively.

3.3.11.2.2 *ReportSpeedingStatistics* command Short-term speeding statistics are transferred using the *ReportSpeedingStatistics* command. This command is generated either after the receipt of a *GetSpeedingStatistics* command or in consequence of a expired measurement period. The format of the *Data* parameter is the same as described in Section 3.3.3.2.1.

3.3.12 *Speeding Statistics – Long-Term Measurement* cluster

The *Speeding Statistics – Long-Term Measurement* cluster provides long-term speeding statistics.

3.3.12.1 Attributes

Table 3.29 lists the attributes of the *Speeding Statistics – Long-Term Measurement* cluster.

Name	**Type**	**Subtype**	**Access**
Trigger Count – Front Camera	UnsignedInt32		read
Trigger Count – Rear Camera	UnsignedInt32		read
Pictures – Front Camera	Array	OctetString	read
Pictures – Rear Camera	Array	OctetString	read

Table 3.29: Attributes of the *Speeding Statistics – Long-Term Measurement* cluster

3.3.12.1.1 *Trigger Count – Front Camera* attribute In contrast to the short-term version of this attribute, the *Trigger Count – Front Camera* attribute for long-term speeding statistics is of type `Unsigned 32-bit Integer`.

3.3.12.1.2 *Trigger Count – Rear Camera* attribute In contrast to the short-term version of this attribute, the *Trigger Count – Rear Camera* attribute for long-term speeding statistics is of type `Unsigned 32-bit Integer`.

3.3.12.1.3 *Pictures – Front Camera* attribute The *Pictures – Front Camera* array stores all pictures that have been made using the front camera. Its elements are of type `Octet String`.

3.3.12.1.4 *Pictures – Rear Camera* attribute As the *Pictures – Front Camera* array, the *Pictures – Rear Camera* array stores all pictures that have been made using the rear camera.

3.3.12.2 Commands

The commands of the *Speeding Statistics – Long-Term Measurement* cluster are summarized in Table 3.30.

R/G	Name	Parameter	Type	Subtype
R	GetSpeedingStatistics	Attribute List	Array	UnsignedInt16
G	ReportSpeedingStatistics	Data	Array	Struct
		Attribute ID	UnsignedInt16	
		Attribute Value	OctetString	

Table 3.30: Commands of the *Speeding Statistics – Long-Term Measurement* cluster

3.3.12.2.1 ***GetSpeedingStatistics*** **command** Long-term speeding statistics can be queried using the *GetSpeedingStatistics* command. The *Attribute List* parameter specifies which attribute values should be retrieved.

3.3.12.2.2 ***ReportSpeedingStatistics*** **command** The *ReportSpeedingStatistics* command is used to transfer long-term speeding statistics. It is generated either after receipt of a *GetSpeedingStatistics* command or in consequence of a expired measurement period. The format of the *Data* parameter is the same as described in Section 3.3.3.2.1.

3.4 FG2 – Environmental data acquisition

Apart from traffic data acquisition, acquisition of environmental data, e.g. air temperature, average wind velocity or precipitation intensity, is important, too. FG2 comprises all attributes and commands that are necessary to gather environmental data of all kind. Table 3.31 lists the clusters of FG2. The four measurement areas, i.e. temperature, wind, precipitation and visibility, are described in the subsequent sections.

Cluster name
General Configuration – Environmental Data Acquisition
Temperature Measurement
Wind Measurement
Precipitation Measurement
Visibility Measurement

Table 3.31: Clusters of FG2

3.4.1 *General Configuration – Environmental Data Acquisition* cluster

The *General Configuration – Environmental Data Acquisition* cluster defines parameters for devices collecting environmental data. This cluster is mandatory independent of the type of data to be gathered.

3.4.1.1 Attributes

The attributes of the *General Configuration – Environmental Data Acquisition* cluster are depicted in Table 3.32.

Name	**Type**	**Access**
Measurement Period	UnsignedInt16	read/write
Operating Mode	Enum8	read/write

Table 3.32: Attributes of the *General Configuration – Environmental Data Acquisition* cluster

3.4.1.1.1 *Measurement Period* attribute The *Measurement Period* gives the measurement period for environmental data acquisition in *seconds*. It is of type `Unsigned 16-bit Integer` and has a resolution of 1 s. Typical values range from 1 s to 600 s.

3.4.1.1.2 *Operating Mode* attribute The *Operating Mode* specifies the operating mode for sensors acquiring environmental data. A detailed description of operating modes for sensors can be found in Section 3.3.1.1.3.

3.4.2 *Temperature Measurement* cluster

The *Temperature Measurement* cluster comprises measurement values for temperature, humidity and atmospheric pressure.

3.4.2.1 Attributes

Table 3.33 shows the attributes of the *Temperature Measurement* cluster.

Name	**Type**	**Access**
Air Temperature	Int16	read
Lane Temperature	Int16	read
Soil Temperature	Int16	read
Freezing Temperature	Int16	read
DewpoInt Temperature	Int16	read
Relative Humidity	UnsignedInt8	read
Atmospheric Pressure	UnsignedInt16	read

Table 3.33: Attributes of the *Temperature Measurement* cluster

3.4.2.1.1 *Air Temperature* attribute The *Air Temperature* attribute is of type `Signed 16-bit Integer` and stores the measured air temperature in *degree Celsius* °C. It has a resolution of 0.1 °C and its values range from $-50\,°C$ to $+60\,°C$.

3.4.2.1.2 ***Lane Temperature* attribute** *Lane Temperature* is the temperature measured at the lane's surface. This attribute is of type `Signed 16-bit Integer` and has a resolution of $0.1\,°\mathrm{C}$. Its values range from $-50\,°\mathrm{C}$ to $+80\,°\mathrm{C}$.

3.4.2.1.3 ***Soil Temperature* attribute** The *Soil Temperature* is measured at some depth below the lane's surface. The sensor's exact depth can be retained using the *Sensor Height* element of the corresponding *Sensors/Actuators* record (see Section 3.2.1.1.7). This attribute is of type `Signed 16-bit Integer`. It has a resolution of $0.1\,°\mathrm{C}$ and ranges from $-50\,°\mathrm{C}$ to $+80\,°\mathrm{C}$.

3.4.2.1.4 ***Freezing Temperature* attribute** The *Freezing Temperature* is the temperature at which the watery solution at the sensor's surface freezes. It is measured in $°\mathrm{C}$ and has a resolution of $0.1\,°\mathrm{C}$. The value of this `Signed 16-bit Integer` typed attribute ranges from $-30\,°\mathrm{C}$ to $0\,°\mathrm{C}$.

3.4.2.1.5 ***Dewpoint Temperature* attribute** *Dewpoint Temperature* is defined as the temperature at which the moistness located in the sensor's ambient air condenses. This `Signed 16-bit Integer` typed attribute has a resolution of $0.1\,°\mathrm{C}$ and its values range from $-30\,°\mathrm{C}$ to $+60\,°\mathrm{C}$.

3.4.2.1.6 ***Relative Humidity* attribute** The `Unsigned 8-bit Integer` typed *Relative Humidity* attribute states the relative humidity of the sensor's ambient air in *percent*. It has a resolution of $1\,\%$ allowing values from $0\,\%$ to $100\,\%$.

3.4.2.1.7 ***Atmospheric Pressure* attribute** *Atmospheric Pressure* is measured in *hPa*. This `Unsigned 16-bit Integer` typed attribute has a resolution of $1\,\mathrm{hPa}$. Its value ranges from $300\,\mathrm{hPa}$ to $1200\,\mathrm{hPa}$.

3.4.2.2 Commands

The commands of the *Temperature Measurement* cluster are shown in Table 3.34.

R/G	Name	Parameter	Type	Subtype
R	GetTemperature	Attribute List	Array	UnsignedInt16
G	ReportTemperature	Data	Array	Struct
		Attribute ID	UnsignedInt16	
		Attribute Value	OctetString	

Table 3.34: Commands of the *Temperature Measurement* cluster

3.4.2.2.1 ***GetTemperature* command** The *GetTemperature* command is used to retrieve temperature measurement values. Parameter *Attribute List* allows a client to query measurement values selectively.

3.4.2.2.2 ***ReportTemperature* command** Temperature measurement values are transfered using the *ReportTemperature* command. This command is generated either after receipt of a *GetTemperature* command or in consequence of a expired measurement period. The format of the *Data* parameter is the same as described in Section 3.3.3.2.1.

3.4.3 *Wind Measurement* cluster

The *Wind Measurement* cluster comprises measurement values for wind direction and wind velocity.

3.4.3.1 Attributes

In Table 3.35 the attributes of the *Wind Measurement* cluster are summarized.

Name	**Type**	**Access**
Wind Direction	UnsignedInt16	read
Average Wind Velocity	UnsignedInt16	read
Peak Wind Velocity	UnsignedInt16	read

Table 3.35: Attributes of the *Wind Measurement* cluster

3.4.3.1.1 ***Wind Direction* attribute** The wind direction is measured in *degrees* (°). The `Unsigned 16-bit Integer` typed *Wind Direction* attribute states the measured wind direction with the resolution of 1 °. Its value ranges from 0 ° to 359 °, representing the four cardinal directions north, east, south and west by the values 0 °, 90 °, 180 ° and 270 °.

3.4.3.1.2 ***Average Wind Velocity* attribute** The *Average Wind Velocity* attribute is of type `Unsigned 16-bit Integer`. It stores the average wind velocity with a resolution of $0.1\,\mathrm{m/s}$ and ranges from $0\,\mathrm{m/s}$ up to $70\,\mathrm{m/s}$.

3.4.3.1.3 ***Peak Wind Velocity* attribute** This `Unsigned 16-bit Integer` typed attribute stores the *Peak Wind Velocity*, i.e. the highest measured wind velocity, with a resolution of $0.1\,\mathrm{m/s}$. Possible values range from $0\,\mathrm{m/s}$ up to $70\,\mathrm{m/s}$.

3.4.3.2 Commands

Table 3.36 lists the commands of the *Wind Measurement* cluster.

3.4.3.2.1 ***GetWind* command** The *GetWind* command is used to retrieve wind measurement values. Parameter *Attribute List* allows a client to query measurement values selectively.

R/G	Name	Parameter	Type	Subtype
R	GetWind	Attribute List	Array	UnsignedInt16
G	ReportWind	Data	Array	Struct
		Attribute ID	UnsignedInt16	
		Attribute Value	OctetString	

Table 3.36: Commands of the *Wind Measurement* cluster

3.4.3.2.2 ***ReportWind* command** Wind measurement values are transfered using the *ReportWind* command. This command is generated either after receipt of a *GetWind* command or in consequence of a expired measurement period. The format of the *Data* parameter is the same as described in Section 3.3.3.2.1.

3.4.4 *Precipitation Measurement* cluster

The *Precipitation Measurement* cluster comprises measurement values regarding precipitation.

3.4.4.1 Attributes

The attributes of the *Precipitation Measurement* cluster are shown in Table 3.37.

Name	Type	Access
Precipitation Indicator	Boolean	read
Precipitation Intensity	UnsignedInt16	read
Precipitation Type	Enum8	read
Water Film Thickness	UnsignedInt16	read
Snow Height	UnsignedInt8	read
Lane State	Enum8	read
Salt Rate	UnsignedInt8	read

Table 3.37: Attributes of the *Precipitation Measurement* cluster

3.4.4.1.1 ***Precipitation Indicator* attribute** The `Boolean` typed *Precipitation Indicator* attribute indicates whether or not precipitation of any type is detected or not.

3.4.4.1.2 ***Precipitation Intensity* attribute** Precipitation intensity is defined as the amount of precipitation during a certain period of time. The *Precipitation Intensity* attribute states the amount of precipitation in *mm/h* water equivalent. It has a resolution of $0.1\,\mathrm{mm/h}$ and its value ranges from $0\,\mathrm{mm/h}$ to $200\,\mathrm{mm/h}$.

3.4.4.1.3 ***Precipitation Type* attribute** The *Precipitation Type* indicates the type of precipitation detected. Table 3.38 lists possible precipitation types.

Value	Precipitation type
0	Other
1	Drizzle
2	Rain
3	Snow
4	Soft Hail
5	Hail

Table 3.38: Values of the *Precipitation Type* enumeration according to [37]

3.4.4.1.4 ***Water Film Thickness* attribute** The *Water Film Thickness* attribute states the thickness of the water film on top of the lane's surface in *mm*. This attribute is of type `Unsigned 16-bit Integer` and has a resolution of 0.01 mm. Its value ranges from 0 mm to 20 mm.

3.4.4.1.5 ***Snow Height* attribute** This attribute of type `Unsigned 8-bit Integer` indicates the height of the snow pack at the measuring point in *cm*. It ranges from 0 cm to val255cm with a resolution of 1 cm.

3.4.4.1.6 ***Lane State* attribute** The *Lane State* attribute indicates the state of the lane's surface. Possible lane states are shown in Table 3.39.

Value	Lane state
0	Other
1	Dry
2	Wet
3	Snowy
4	Icy

Table 3.39: Values of the *Lane State* enumeration according to [37]

3.4.4.1.7 ***Salt Rate* attribute** The *Salt Rate* attribute indicates the saturation level of the watery solution on top of the lane. It is of type `Unsigned 8-bit Integer` and has a resolution of 1 %. Its possible values range from 0 % to 100 %.

3.4.4.2 Commands

Table 3.40 depicts the commands of the *Precipitation Measurement* cluster.

3.4.4.2.1 ***GetTemperature* command** Precipitation measurement values can be retrieved using the *GetPrecipitation* command. The parameter *Attribute List* allows a client to query measurement values selectively.

R/G	Name	Parameter	Type	Subtype
R	GetPrecipitation	Attribute List	Array	UnsignedInt16
G	ReportPrecipitation	Data	Array	Struct
		Attribute ID	UnsignedInt16	
		Attribute Value	OctetString	

Table 3.40: Commands of the *Precipitation Measurement* cluster

3.4.4.2.2 *ReportPrecipitation* command The *ReportPrecipitation* command is used to transfer precipitation measurement values. This command is generated either after receipt of a *Get-Precipitation* command or in consequence of a expired measurement period. The format of the *Data* parameter is the same as described in Section 3.3.3.2.1.

3.4.5 *Visibility Measurement* cluster

The *Visibility Measurement* cluster comprises measurement values for brightness and visibility.

3.4.5.1 Attributes

Table 3.41 lists the attributes of the *Visibility Measurement* cluster.

Name	Type	Access
Visual Range	UnsignedInt16	read
Brightness	UnsignedInt16	read
Visibility Conditions	Enum8	read

Table 3.41: Attributes of the *Visibility Measurement* cluster

3.4.5.1.1 *Visibility Range* attribute The *Visibility Range* attribute stores the measured visibility range in *m*. It is of type `Unsigned 16-bit Integer` and its possible values range from $5\,\mathrm{m}$ to $1000\,\mathrm{m}$.

3.4.5.1.2 *Brightness* attribute The `Unsigned 16-bit Integer` typed *Brightness* attribute indicates the brightness in *Lx*. Possible values range from $0\,\mathrm{Lx}$ to $60000\,\mathrm{Lx}$.

3.4.5.1.3 *Visibility Conditions* attribute This attribute states the visibility conditions prevailing at the controlled section. Table 3.42 lists the possible values for this attribute.

3.4.5.2 Commands

In Table 3.43, the commands of the *Visibility Measurement* cluster are shown.

Value	Visibility conditions
0	Other
1	Clear
2	Fog patches
3	Heavy fog
4	Smoke
5	Blowing Snow
6	Blowing Dust
7	Sun glare

Table 3.42: Values of the *Visibility Conditions* enumeration according to [37]

R/G	Name	Parameter	Type	Subtype
R	GetVisibility	Attribute List	Array	UnsignedInt16
G	ReportVisibility	Data	Array	Struct
		Attribute ID	UnsignedInt16	
		Attribute Value	OctetString	

Table 3.43: Commands of the *Visibility Measurement* cluster

3.4.5.2.1 ***GetVisibility* command** The *GetVisibility* command is used to retrieve visibility measurement values. Parameter *Attribute List* allows a client to query measurement values selectively.

3.4.5.2.2 ***ReportVisibility* command** Visibility measurement values are transfered using the *ReportVisibility* command. This command is generated either after receipt of a *GetVisibility* command or in consequence of a expired measurement period. The format of the *Data* parameter is the same as described in Section 3.3.3.2.1.

3.5 FG3 – Traffic control

Last but not least, FG3 provides the means for traffic control. Table 3.44 summarizes the clusters of FG3. They are depicted in detail in the following sections.

Cluster name
General Configuration – Traffic Control
Variable Message Signs – Configuration
Variable Message Signs
Groups – Configuration
Groups

Table 3.44: Clusters of FG3

3.5.1 *General Configuration – Traffic Control* cluster

The *General Configuration – Traffic Control* cluster is mandatory and specifies parameters for devices used for traffic control.

3.5.1.1 Attributes

The attributes of the *General Configuration – Traffic Control* cluster are shown in Table 3.45.

Name	Type	Access
Operating Mode	Enum8	read/write
Brightness Control	Enum8	read/write

Table 3.45: Attributes of the *General Configuration – Traffic Control* cluster

3.5.1.1.1 ***Operating Mode* attribute** The *Operating Mode* attribute indicates the mode of the sign. Its possible values are depicted in Table 3.46.

Value	Operating mode
0	Normal operation
1	Manual operation

Table 3.46: Values of the *Operating Mode* enumeration for signs according to [4]

3.5.1.1.2 ***Brightness Control* attribute** This attribute specifies how the sign's brightness is controlled. Possible modes are depicted in Table 3.47. *Automatic* brightness control means that the sign adjusts its brightness automatically. However, this mode may not be supported by all signs. If the brightness is controlled *remotely*, the sign's brightness is changed after the receipt of the corresponding write command only.

Value	Brightness control
0	Remotely controlled
1	Automatic

Table 3.47: Values of the *Brightness Control* enumeration according to [4]

3.5.2 *Variable Message Signs – Configuration* cluster

The *Variable Message Signs – Configuration* cluster defines parameters for VMSs.

3.5.2.1 Attributes

Table 3.48 summarizes the attributes of the *Variable Message Signs – Configuration* cluster.

Name	Type	Subtype	Access
Sign Type	Enum8		read
Beacon Type	Enum8		read
Sign Technology	UnsignedInt8		read
Sign Height	UnsignedInt16		read
Sign Width	UnsignedInt16		read
Default Content Code	UnsignedInt8		read/write
Default Display Mode	Enum8		read/write
Default Beacon Mode	Enum8		read/write
Default Flashing Time	UnsignedInt8		read/write
Defined Contents	Array	Struct	read/write
Content Code	UnsignedInt8		
Content Definition	OctetString		
Default Font	UnsignedInt8		read/write
Default Foreground Color	Enum8		read/write
Default Background Color	Enum8		read/write
Default Horizontal Justification	Enum8		read/write
Default Vertical Justification	Enum8		read/write
Defined Fonts	Array	Struct	read/write
Font ID	UnsignedInt8		
Character Spacing	UnsignedInt8		
Line Spacing	UnsignedInt8		
Font Definition	OctetString		

Table 3.48: Attributes of the *Variable Message Signs – Configuration* cluster

3.5.2.1.1 *Sign Type* attribute The *Sign Type* attribute indicates the type of sign. Table 3.49 lists the different sign types.

3.5.2.1.2 *Beacon Type* attribute This attribute specifies numbers and arrangement of beacons on a sign. The beacon types are shown in Table 3.53.

3.5.2.1.3 *Sign Technology* attribute The *Sign Technology* attribute indicates all technologies utilized by the sign. It is a bit field of type `Unsigned 8-bit Integer`. Defined sign technologies are shown in Table 3.51. Note that more than one flag may be set.

3.5.2.1.4 *Sign Height* attribute The *Sign Height* attribute states the height of the sign in *pixels*. This `Unsigned 16-bit Integer` typed attribute is optional. Its presence depends on the value of the *Sign Type* attribute (see Section 3.5.2.1.1).

Value	Sign type
0	Other
1	Blank-Out Sign (BOS)
2	Changeable Message Sign (CMS)
3	VMS with character matrix setup
4	VMS with line matrix setup
5	VMS with full matrix setup
6	Other portable
7	Portable BOS
8	Portable CMS
9	Portable VMS with character matrix setup
10	Portable VMS with line matrix setup
11	Portable VMS with full matrix setup

Table 3.49: Values of the *Sign Type* enumeration according to [1]

Value	Beacon type
0	Other
1	None
2	One beacon, flashing
3	One beacon, strobe light
4	Two beacons, synchronized flashing
5	Two beacons, opposed flashing
6	Two beacons, strobe light
7	Four beacons, synchronized flashing
8	Four beacons, alternate row flashing
9	Four beacons, alternate column flashing
10	Four beacons, alternate diagonal flashing
11	Four beacons, strobe light

Table 3.50: Values of the *Beacon Type* enumeration accoding to [1]

3.5.2.1.5 ***Sign Width* attribute** The `Unsigned 16-bit Integer` typed *Sign Width* attribute states the width of the sign in *pixels*. This attribute is optional. It has to be present depending on the value of the *Sign Type* attribute (see Section 3.5.2.1.1).

3.5.2.1.6 ***Default Content Code* attribute** The *Default Content Code* attribute specifies the content that has to be displayed by default or in case of any error. Its value is of type `Unsigned 8-bit Integer` and must refer to a valid record of the *Defined Contents* attribute (see Section 3.5.2.1.10).

Bit	Technology
0	Other
1	LED
2	Flip disk
3	Fiber optics
4	Shuttered
5	Lamp
6	Drum
7	*reserved*

Table 3.51: Flags for utilized sign technologies according to [1]

3.5.2.1.7 ***Default Display Mode* attribute** This attribute defines the default display mode of the sign. Possible display modes are depicted in Table 3.52.

Value	Display mode
0	Off
1	On
2	Flashing

Table 3.52: Values of the *Display Mode* enumeration according to [4]

3.5.2.1.8 ***Default Beacon Mode* attribute** The *Default Beacon Mode* attribute specifies the default mode of beacons on the sign. Its values are shown in Table 3.53.

Value	Beacon mode
0	Inactive
1	Active

Table 3.53: Values of the *Beacon Mode* enumeration

3.5.2.1.9 ***Default Flashing Time* attribute** The *Default Flashing Time* determines the default flashing time for display mode *Flashing* in *ms*. This attribute is of type `Unsigned 8-bit Integer` and has resolution is 50 ms. Its typical values range from 100 ms to 1000 ms.

3.5.2.1.10 ***Defined Contents* attribute** The *Defined Contents* array stores a record for each displayable content of the sign. Each record consists of the `Unsigned 8-bit Integer` typed *Content Code* and a variable length `Octet String`, i.e. the *Content Definition*. Contents are stored in ascending order of the *Content Code*.

3.5.2.1.11 ***Default Font* attribute** The *Default Font* attribute specifies the font that has to be used by default. Its value is of type `Unsigned 8-bit Integer` and must refer to a valid record of the *Defined Fonts* attribute (see Section 3.5.2.1.16).

3.5.2.1.12 ***Default Foreground Color* attribute** This attribute determines the default foreground color, i.e. the default color used for text messages. Possible colors are enumerated in Table 3.54. However, not all of them have to be supported on the sign.

Value	Color
0	black
1	white
2	red
3	green
4	blue
5	yellow
6	cyan
7	magenta
8	orange

Table 3.54: Values of the *Color* enumeration according to [1]

3.5.2.1.13 ***Default Background Color* attribute** The *Default Background Color* defines the sign's default background color. It has to be one of the colors defined in Table 3.54.

3.5.2.1.14 ***Default Horizontal Alignment* attribute** The *Default Horizontal Alignment* attribute specifies the horizontal alignment for text messages used by default. Its possible values are shown in Table 3.55.

Value	Horizontal alignment
0	Left
1	Centered
2	Right

Table 3.55: Values of the *Horizontal Alignment* enumeration according to [1]

3.5.2.1.15 ***Default Vertical Alignment* attribute** This attribute defines the vertical alignment for text messages used by default. The values are defined in Table 3.56.

3.5.2.1.16 ***Defined Fonts* attribute** The *Defined Fonts* array stores a record for each eligible font for text messages. Each record consists of the `Unsigned 8-bit Integer` typed *Font ID*, *Character Spacing*, *Line Spacing* and a variable length `Octet String`, i.e. the *Font*

Value	Vertical alignment
0	Top
1	Centered
2	Bottom

Table 3.56: Values of the *Vertical Alignment* enumeration according to [1]

Definition. Elements *Character Spacing* and *Line Spacing* are of type `Unsigned 8-bit Integer`. Both are specified in *pixels*. Fonts are stored in ascending order of the *Font ID*.

3.5.2.2 Commands

The commands of the *Variable Message Signs – Configuration* cluster are shown in Table 3.57.

R/G	Name	Parameter	Type
R	SetContent	Content Code Content Definition Overwrite	UnsignedInt8 OctetString Boolean
R	GetContent	Content Code	UnsignedInt8
G	GetContentResponse	Content Definition	OctetString
R	DeleteContent	Content Code	UnsignedInt8
R	SetFont	Font ID Character Spacing Line Spacing Font Definition Overwrite	UnsignedInt8 UnsignedInt8 UnsignedInt8 OctetString Boolean
R	GetFont	Font ID	UnsignedInt8
G	GetFontResponse	Character Spacing Line Spacing Font Definition	UnsignedInt8 UnsignedInt8 OctetString
R	DeleteFont	Font ID	UnsignedInt8

Table 3.57: Commands of the *Variable Message Signs – Configuration* cluster

3.5.2.2.1 *SetContent* command The *SetContent* command allows uploading of new contents to the VMS. This command has three parameters, i.e. *Content Code*, *Content Definition* and *Overwrite*. The *Content Code* is of type `Unsigned 8-bit Integer`. Parameter *Overwrite* states whether or not an already existing content definition should be overwritten with the `Octet String` typed *Content Definition*.

3.5.2.2.2 *GetContent* command Existing contents can be downloaded from the sign's memory using the *GetContent* command. Its only parameter is the *Content Code* parameter specifying the content to be downloaded. If a *GetContent* command is received, a *GetContentResponse* is generated returning the content definition.

3.5.2.2.3 ***GetContentResponse* command** The *GetContentResponse* is generated after receipt of a *GetContent* command. It returns the `Octet String` typed *Content Definition* of the requested content.

3.5.2.2.4 ***DeleteContent* command** Contents that are no longer used can be deleted using the *DeleteContent* command. Parameter *Content Code* specifies the `Unsigned 8-bit Integer` content code of the content that is to be deleted. The corresponding record of the *Defined Contents* array has to be removed and allocated memory has to be freed.

3.5.2.2.5 ***SetFont* command** The *SetFont* command allows uploading of new fonts to the VMS. This command has five parameters, i.e. *Font ID*, *Character Spacing*, *Line Spacing*, *Font Definition* and *Overwrite*. Parameters *Font ID*, *Character Spacing* and *Line Spacing* are of type `Unsigned 8-bit Integer`. The *Overwrite* parameter states whether or not an already existing font should be overwritten.

3.5.2.2.6 ***GetFont* command** Existing fonts can be downloaded from the sign's memory using the *GetFont* command. Its only parameter is the *Font ID* parameter specifying the font to be downloaded. If a *GetFont* command is received, a *GetFontResponse* is generated returning the font definition.

3.5.2.2.7 ***GetFontResponse* command** The *GetFontResponse* is generated after receipt of a *GetFont* command. It returns the `Unsigned 8-bit Integer` typed *Character Spacing* and *Line Spacing* as well as the `Octet String` typed *Font Definition* of the requested font.

3.5.2.2.8 ***DeleteFont* command** Fonts that are no longer used can be deleted using the *DeleteFont* command. Parameter *Font ID* specifies the `Unsigned 8-bit Integer` font identifier of the font that is to be deleted. The corresponding record of the *Defined Fonts* array has to be removed and allocated memory has to be freed.

3.5.3 *Variable Message Signs* cluster

The *Variable Message Signs* cluster comprises status attributes and set values for VMSs.

3.5.3.1 Attributes

Table 3.58 depicts the attributes of the *Variable Message Signs* cluster.

3.5.3.1.1 ***Display Status* attribute** The *Display Status* indicates the sign's status. Possible values are shown in Table 3.4.

Name	**Type**	**Subtype**	**Access**
Display Status	Enum8		read
Beacon Status	Enum8		read
PoInt of Light Status	Array	UnsignedInt8	read
Faulty Content Codes	Array	UnsignedInt8	read
Faulty Text Positions	Array	Struct	read
Row	UnsignedInt8		
Column	UnsignedInt8		
Brightness	UnsignedInt8		read/write
Current Content Code	UnsignedInt8		read/write
Current Display Mode	Enum8		read/write
Current Beacon Mode	Enum8		read/write
Current Flashing Time	UnsignedInt8		read/write

Table 3.58: Attributes of the *Variable Message Signs* cluster

3.5.3.1.2 ***Beacon Status* attribute** The *Beacon Status* denotes the status of the beacons on the sign. Table 3.4 depicts the possible values.

3.5.3.1.3 ***Point of Light Status* attribute** The *Point of Light Status* array of type `Unsigned 8-bit Integer` encodes the status for each single point of light, e.g. a single LED, placed on the sign. Each point of light is represented by a single bit. 0 means *operational* whereas 1 stands for *faulty*.

3.5.3.1.4 ***Faulty Content Codes* attribute** Due to faulty points of light, some of the stored contents may not be displayed properly anymore. The *Faulty Content Codes* array maintains a list of all contents that can not be displayed properly. Its elements are of type `Unsigned 8-bit Integer` containing content codes of faulty contents.

3.5.3.1.5 ***Faulty Text Positions* attribute** The *Faulty Text Positions* array maintains a list of faulty text positions specified by *Row* and *Column* number. Both, row and column numbers are of type `Unsigned 8-bit Integer`.

3.5.3.1.6 ***Brightness* attribute** The *Brightness* of the sign is specified in *percent*. Its values range from 0 % to 100 %. If the *Brightness Control* (see Section 3.5.1.1.2) is set to *automatic*, this attribute is read-only and its value reflects the automatically determined brightness.

3.5.3.1.7 ***Current Content Code* attribute** The `Unsigned 8-bit Integer` typed *Current Content Code* attribute states the currently set content of the sign. Its value must refer to a valid record of the *Defined Contents* attribute (see Section 3.5.2.1.10).

3.5.3.1.8 *Current Display Mode* attribute

The sign's display is controlled by the *Current Display Mode* attribute. Its possible values are listed in Table 3.52.

3.5.3.1.9 *Current Beacon Mode* attribute

Beacons connected to the sign are controlled by the *Current Beacon Mode* attribute. Table 3.53 depicts the possible values for this attribute.

3.5.3.1.10 *Current Flashing Time* attribute

The *Current Flashing Time* attribute determines the flashing time for the currently displayed content in *ms*. This attribute is of type `Unsigned 8-bit Integer` and has a resolution of 50 ms. Its typical values range from 100 ms to 1000 ms.

3.5.3.2 Commands

The commands of the *Variable Message Signs* cluster are summarized in Table 3.59.

R/G	**Name**	**Parameter**	**Type**
R	Show	Content Code	UnsignedInt8
		Display Mode	Enum8
		Beacon Mode	Enum8
		Flashing Time	UnsignedInt8

Table 3.59: Commands of the *Variable Message Signs* cluster

3.5.3.2.1 *Show* command

The *Show* command is used to activate certain contents and display modes. It sets the *Current Content Code*, *Current Display Mode*, and *Current Beacon Mode* attributes according to the parameters *Content Code*, *Display Mode*, and *Beacon Mode*. If *Display Mode* is set to *Flashing*, the *Current Flashing Time* is set to *Flashing Time*. Otherwise it remains unchanged.

3.5.4 *Groups – Configuration* cluster

The *Groups – Configuration* cluster specifies parameters for groups of VMSs.

3.5.4.1 Attributes

The attributes of the *Groups – Configuration* cluster are summarized in Table 3.60.

3.5.4.1.1 *Default Group Code* attribute

The *Default Group Code* attribute specifies the group configuration that has to be displayed by default or in case of any error. Its value is of type `Unsigned 8-bit Integer` and must refer to a valid record of the *Defined Groups* attribute (see Section 3.5.4.1.2).

Name	Type	Subtype	Access
Default Group Code	UnsignedInt8		read/write
Defined Group Configurations	Array	Struct	read/write
Group Code	UnsignedInt8		
Components	Array	Struct	
Content Code	UnsignedInt8		
Display Mode	Enum8		
Beacon Mode	Enum8		
Flashing Time	UnsignedInt8		

Table 3.60: Attributes of the *Groups – Configuration* cluster

3.5.4.1.2 ***Defined Group Configurations* attribute** Group configurations are necessary to control a group of signs at a time. The *Defined Group Configurations* array stores a record for each group configuration. Each group record consists of the *Group Code* of type `Unsigned 8-bit Integer` and the *Components* array. The *Components* array contains a component record for each sign of the group. Components are defined by the *Content Code*, *Display Mode*, *Beacon Mode* and *Flashing Mode*. A functional description of these elements can be found in Sections 3.5.2.1.6 to 3.5.2.1.9.

3.5.4.2 Commands

Table 3.61 lists the commands of the *Groups – Configuration* cluster.

R/G	Name	Parameter	Type	Subtype
R	SetGroupConfiguration	Group Code	UnsignedInt8	
		Components	Array	Struct
		Content Code	UnsignedInt8	
		Display Mode	Enum8	
		Beacon Mode	Enum8	
		Flashing Time	UnsignedInt8	
		Overwrite	Boolean	
R	GetGroupConfiguration	Group Code	UnsignedInt8	
G	GetGroupResponse	Components	Array	Struct
		Content Code	UnsignedInt8	
		Display Mode	Enum8	
		Beacon Mode	Enum8	
		Flashing Time	UnsignedInt8	
R	DeleteGroupConfiguration	Group Code	UnsignedInt8	

Table 3.61: Commands of the *Groups – Configuration* cluster

3.5.4.2.1 ***SetGroupConfiguration* command** The *SetGroupConfiguration* command allows the upload of new group configurations. This command has three parameters, i.e. *Group Code*, *Components* and *Overwrite*. The *Group Code* is of type `Unsigned 8-bit Integer`. Parameter *Overwrite* states whether or not an already existing group configuration should be overwritten with the *Components* array. The format of the *Components* array is described in Section 3.5.4.1.2.

3.5.4.2.2 ***GetGroupConfiguration* command** Existing group configurations can be downloaded from the device's memory using the *GetGroupConfiguration* command. Its only parameter is the *Group Code* parameter specifying the group configuration to be downloaded. If a *GetGroupConfiguration* command is received, a *GetGroupConfigurationResponse* is generated returning the group configuration.

3.5.4.2.3 ***GetGroupConfigurationResponse* command** A *GetConfigurationResponse* is generated after receipt of a *GetGroupConfiguration* command. It returns the *Components* array of the requested group configuration. The format of the *Components* array is described in Section 3.5.4.1.2.

3.5.4.2.4 ***DeleteGroupConfiguration* command** No longer used group configurations can be deleted using the *DeleteGroupConfiguration* command. Parameter *Group Code* specifies the `Unsigned 8-bit Integer` group code of the group configuration that is to be deleted. The corresponding record of the *Defined Groups* array has to be removed and allocated memory has to be freed.

3.5.5 *Groups* cluster

The *Groups* cluster comprises set values and status attributes for VMS groups.

3.5.5.1 Attributes

The attributes of the *Groups* cluster are shown in Table 3.62.

Name	Type	Access
Display Status	UnsignedInt8	read
Brightness	UnsignedInt8	read/write
Current Group Code	UnsignedInt8	read/write

Table 3.62: Attributes of the *Groups* cluster

3.5.5.1.1 ***Display Status* attribute** The *Display Status* indicates the group's display status. Each bit indicates the status of a single VMS within the group. A value of 1 means that the VMS is erroneous.

3.5.5.1.2 ***Brightness* attribute** The *Brightness* for the group is specified in *percent*. Its values range from $0\,\%$ to $100\,\%$. If the *Brightness Control* (see Section 3.5.1.1.2) is set to *automatic*, this attribute is read-only and its value reflects the automatically determined brightness for the group.

3.5.5.1.3 ***Current Group Code* attribute** The `Unsigned 8-bit Integer` typed *Current Group Code* attribute specifies the currently set group configuration. Its value must refer to a valid record of the *Defined Group Configurations* attribute (see Section 3.5.4.1.2).

3.5.5.2 Commands

Table 3.63 depicts the commands of the *Groups* cluster.

R/G	**Name**	**Parameter**	**Type**
R	Show	Group Code	UnsignedInt8

Table 3.63: Commands of the *Groups* cluster

3.5.5.2.1 ***Show* command** The *Show* command is used to activate a certain group configuration. Its only parameter is the `Unsigned 8-bit Integer` typed *Group Code*. After receipt of this command, the *Current Group Code* is set to *Group Code*.

CHAPTER 4

Prototype implementation

4.1 Overview

As proof-of-concept of the information model proposed in Chapter 3, a traffic jam warning system has been implemented. The next section gives a concise use case description of traffic jam warning systems and shows the implementation details. Finally, other possible use cases are depicted briefly.

4.2 Traffic jam warning system

4.2.1 Use case description

Due to road works, traffic jams can occur especially at rush hours. A mobile traffic jam warning system can notify the driver early enough such that alternative routes may be chosen. Such a system consists of at least one Variable Message Sign (VMS), indicating a traffic jam, and a device collecting short-term traffic data, i.e. average velocity and traffic intensity (see Figure 4.1).

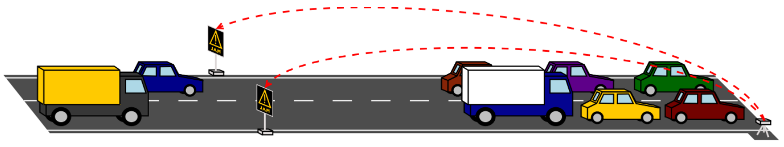


Figure 4.1: Traffic jam warning system

The device collecting short-term traffic data must support the clusters depicted in Table 4.1. A detailed description of clusters can be found in the particular sections of Chapter 3. Since the device definition shown in Table 4.1 includes the server sides of clusters *Vehicle Measurement – Configuration* and *Vehicle Measurement*, all necessary sensors are *directly* connected to the device. However, it might also be possible that the device simply collects data from independent sensors. In the latter case, the device definition would contain the client side of the *Vehicle Measurement* cluster.

Server Side	**Client Side**
General Device Information	General Device Information
General Configuration – Traffic Data Acquisition	
Vehicle Measurement – Configuration	
Vehicle Measurement	
Traffic Statistics – Configuration	
Traffic Statistics – Short-Term Measurement	

Table 4.1: Clusters supported by the traffic data collecting device

Table 4.2 shows the supported clusters of the VMS. The client side of the *Traffic Statistics – Short-Term Measurement* cluster is necessary to form a cluster binding between the VMS and the traffic data collecting device.

Server Side	**Client Side**
General Device Information	General Device Information
General Configuration – Traffic Control	Traffic Statistics – Short-Term Measurement
Variable Message Signs – Configuration	
Variable Message Signs	

Table 4.2: Clusters supported by VMS

It remains to show how the individual parts can be put together. Figure 4.2 depicts the setup of the traffic jam warning system. For this application, it is sufficient to set the attributes *Measurement Period – Short-Term Data*, *Operating Mode* and *Averaging Mode* of the traffic data collecting device. To allow autonomous control of the VMS, the traffic data collecting device must be in *Reporting* mode. Furthermore, the VMS has to subscribe for reporting of attributes *Average Velocity - Total* and *Traffic Intensity - Total*. With that, the traffic data collecting device reports the values of attributes *Average Velocity - Total* and *Traffic Intensity - Total* periodically, i.e. at the end of each measurement period. The VMS then decides whether or not a warning needs to be displayed to notify road users about an upcoming traffic jam.

4.2.2 Implementation details

As described in Section 2.2.2, ZigBee is a low-cost, low-power, wireless communication standard [40]. It specifies security methods for Network Layer (NWK) and Application Layer (APL) layers as well as a framework for distributed applications. The application framework contains simple mechanisms for data modeling amongst others. However, these data modeling mechanisms are sufficient to map the information model proposed in Chapter 3. Due to the built-in

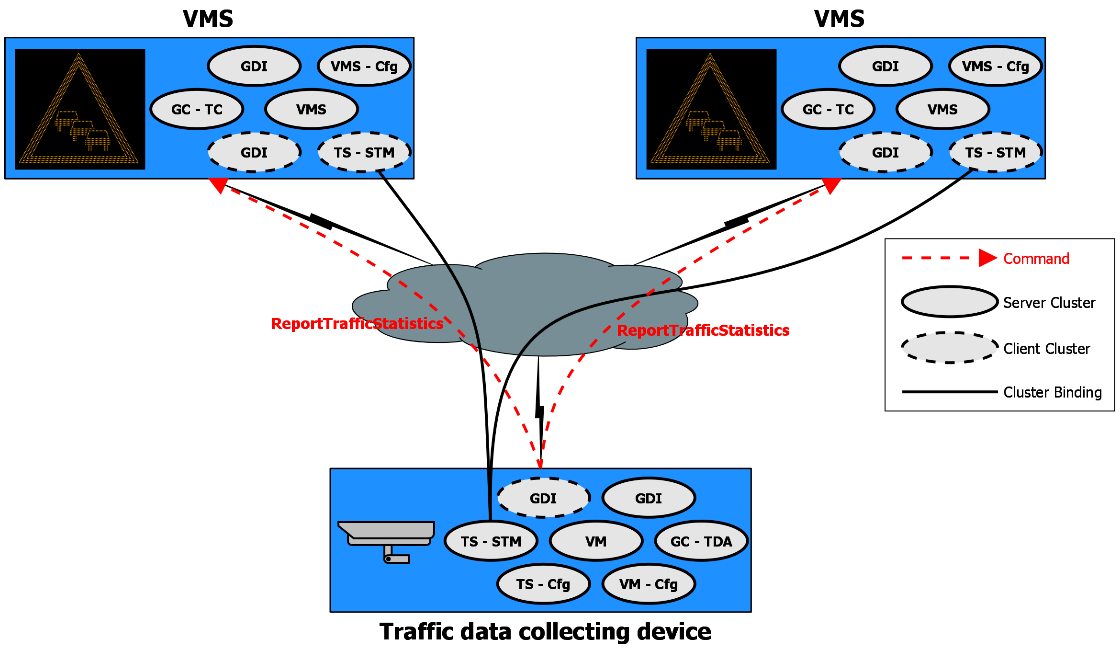


Figure 4.2: Setup of the traffic jam warning system

security and data modeling scheme, ZigBee seems to be well suited not only for prototype implementations.

After the wireless communication standard has been determined, a suitable hardware platform had to be picked. Without specific reasons, the *CC2520 Development Kit*[1] from Texas Instruments has been chosen. It contains the CC2520, a ZigBee/IEEE 802.15.4 RF transceiver, and all the necessary hardware for prototyping of ZigBee applications.

Last but not least a traffic data collecting device as well as a VMS have been needed. However, for lack of the required hardware, both devices have been simulated by applications running on a PC. Figure 4.3 depicts the hardware setup used for the prototype implementation. The CC2520 evaluation boards have been connected to the serial port of the respective PC. They implement the information model and act as gateway to the Wireless Sensor and Actuator Network (WSAN).

The traffic data collecting device has been simulated by the *TDAsim* application. Figure 4.4(a) shows the start screen of *TDAsim*. After choosing the serial port settings and pressing the *Connect* button, the application connects with the evaluation board (see Figure 4.4(b)). Using the + and - buttons, the values for traffic intensity and average velocity can be set. By pressing the *Report* button, the values for traffic intensity and average velocity are sent to the gateway and in turn reported to the VMS using the *ReportTrafficStatistics* command.

Application *VMSsim* has been used to simulate the VMS. Figure 4.6(a) shows the start screen of *VMSsim*. Again, the serial port settings have to be chosen before connecting to the

[1] http://www.ti.com/tool/cc2520dk

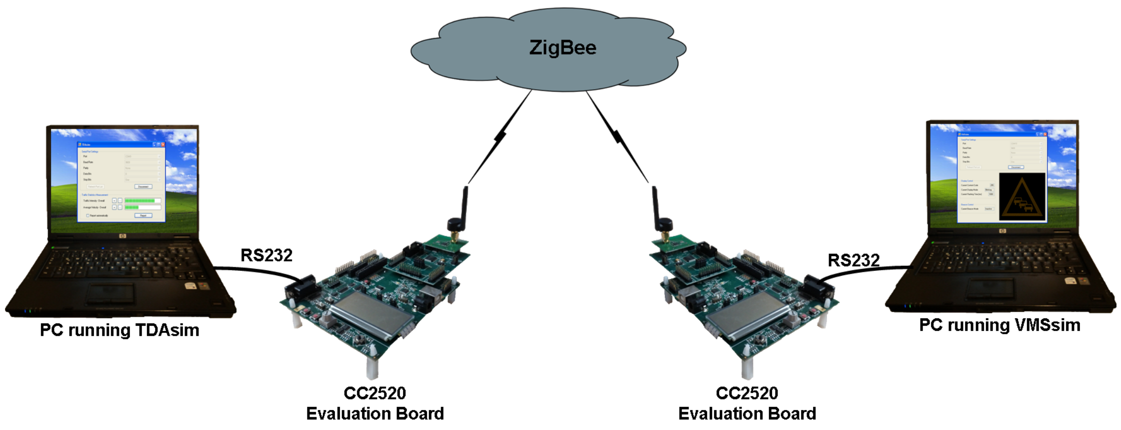
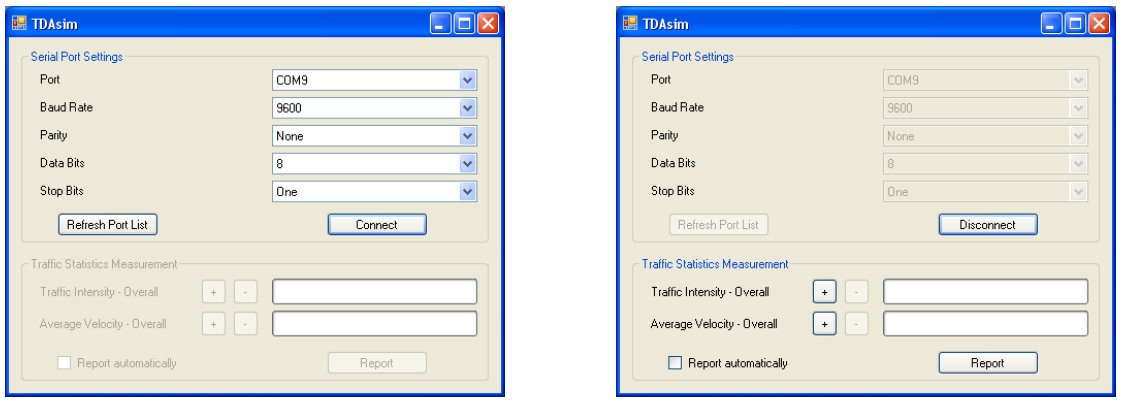


Figure 4.3: Hardware setup of the traffic jam warning system



TDAsim

Serial Port Settings

Port	COM9
Baud Rate	9600
Parity	None
Data Bits	8
Stop Bits	One

Refresh Port List · Connect

Traffic Statistics Measurement

Traffic Intensity - Overall · Average Velocity - Overall · Report automatically · Report

(a)

TDAsim

Serial Port Settings

Port	COM9
Baud Rate	9600
Parity	None
Data Bits	8
Stop Bits	One

Refresh Port List · Disconnect

Traffic Statistics Measurement

Traffic Intensity - Overall · Average Velocity - Overall · Report automatically · Report

(b)

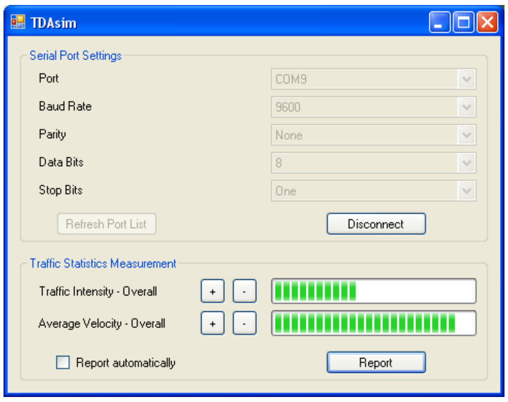
Figure 4.4: Starting the *TDAsim* application

evaluation board (see Figure 4.6(b)). Whenever the gateway receives a *ReportTrafficStatistics* command, it decides whether or not the jam warning needs to be displayed. If traffic intensity is low and average velocity is high (see Figure 4.5(a)), no warning needs to be displayed. However, if traffic intensity is high and average velocity is low (see Figure 4.5(b)), road users have to be notified about the upcoming traffic jam. After the decision has been taken, the gateway sends the new values for content code and display mode to the *VMSsim* application (see Figure 4.7).

4.3 Other use cases

4.3.1 Excess load warning system

There are some parts of the road infrastructure that are passable for vehicles with restricted total load only. A common example is a bridge. Vehicles exceeding the maximum total load of a


TDAsim

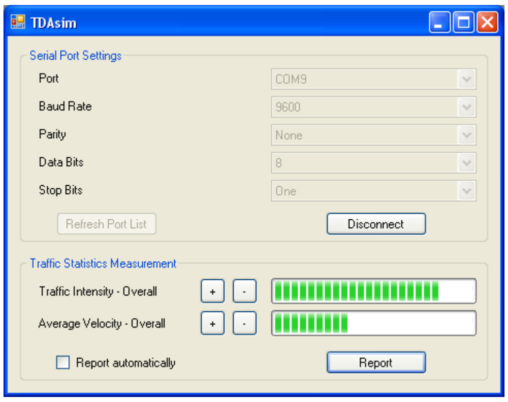
Serial Port Settings

Port: COM9
Baud Rate: 9600
Parity: None
Data Bits: 8
Stop Bits: One

Traffic Statistics Measurement

Traffic Intensity - Overall
Average Velocity - Overall

(a)


TDAsim

Serial Port Settings

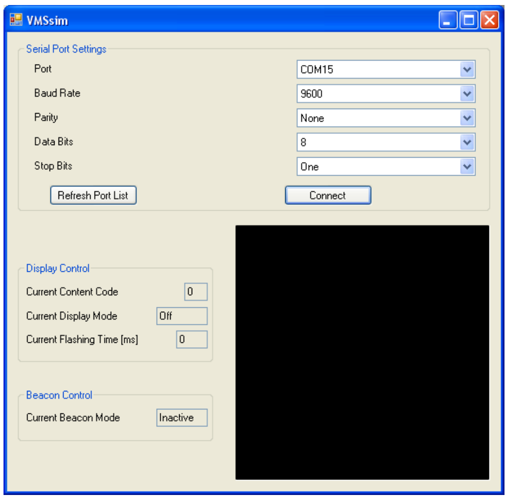
Port: COM9
Baud Rate: 9600
Parity: None
Data Bits: 8
Stop Bits: One

Traffic Statistics Measurement

Traffic Intensity - Overall
Average Velocity - Overall

(b)

Figure 4.5: Using the *TDAsim* application to simulate traffic


VMSsim

Serial Port Settings

Port: COM15
Baud Rate: 9600
Parity: None
Data Bits: 8
Stop Bits: One

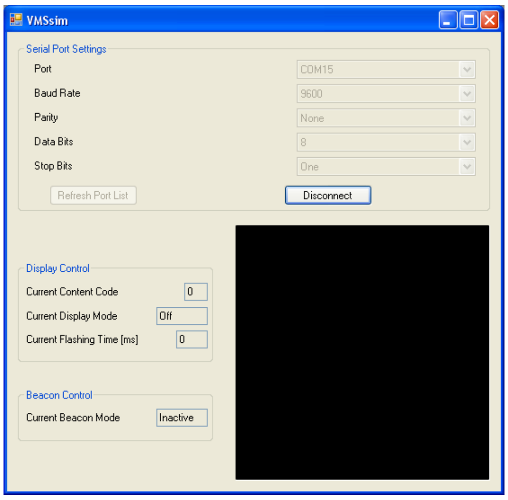
Display Control

Current Content Code: 0
Current Display Mode: Off
Current Flashing Time [ms]: 0

Beacon Control

Current Beacon Mode: Inactive

(a)


VMSsim

Serial Port Settings

Port: COM15
Baud Rate: 9600
Parity: None
Data Bits: 8
Stop Bits: One

Display Control

Current Content Code: 0
Current Display Mode: Off
Current Flashing Time [ms]: 0

Beacon Control

Current Beacon Mode: Inactive

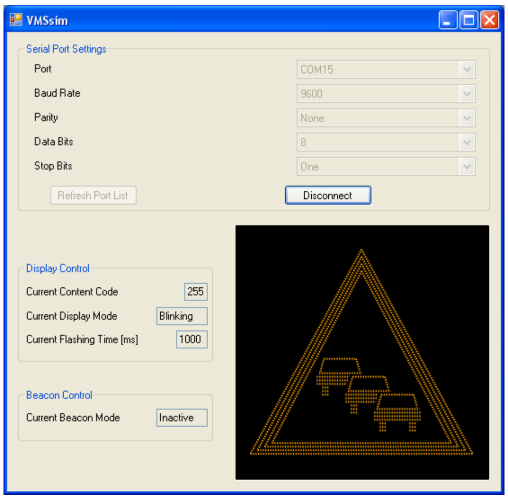
(b)

Figure 4.6: Starting the *VMSsim* application

bridge can cause severe damage. To avoid this, an excess load warning system can warn the driver in case of an excess load so that the bridge can be bypassed. This system consists of a device collecting detailed vehicle information, including total load and axle loads, and a VMS indicating the excess load (see Figure 4.8).

4.3.2 Fog warning system

For the last use case, regions are considered that are prone to limited visibility due to fog. Fog warning systems can notify road users early enough, such that accidents can be prevented. Such



Serial Port Settings

Port	COM15
Baud Rate	9600
Parity	None
Data Bits	8
Stop Bits	One

Display Control

Current Content Code	255
Current Display Mode	Blinking
Current Flashing Time [ms]	1000

Beacon Control

Current Beacon Mode	Inactive

Figure 4.7: Using the *VMSsim* application to simulate a VMS

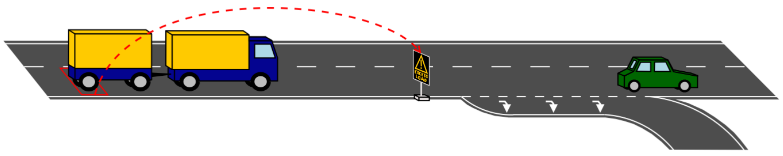


Figure 4.8: Excess load warning system

systems consist of a variety of visibility measuring devices and at least one VMS (see Figure 4.9).

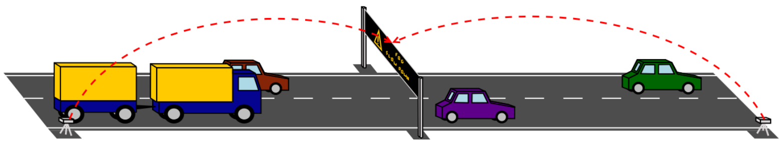


Figure 4.9: Fog warning system

Part III

Security concept for distributed traffic management applications

CHAPTER 5

Security concept specification

5.1 Overview

Security plays an important role in nearly every field of application. However, its requirements may vary from application to application. As mentioned in Section 2.4, security decisions have to be taken with care, especially if a broadcast medium is used for communication. The term *security* is a hypernym encompassing the five areas

- **confidentiality**,
- **integrity**,
- **availability**,
- **authentication**, and
- **freshness**.

Confidentiality means that some data or message can only be read by a person or network node the data or message is destined for. To ensure confidentiality, data needs to be encrypted and decrypted, respectively. This can be done either using a pair of keys (*public-key cryptography*) or a single key (*symmetric-key cryptography*). Details for public- and symmetric-key cryptography are explained in Sections 2.4.3 and 2.4.2.

Integrity denotes the correctness of a message, i.e., that it has neither been corrupted during transmission nor altered by a malicious third. The Message Integrity Code (MIC) is a common principle to ensure message integrity. A MIC can be computed using a cryptographic hash function or block cipher (see Sections 2.4.1.3 and 2.4.2, respectively). The computed MIC is sent along with the message and compared with the recomputed value by the receiver in order to verify message integrity.

In the context of this work, *availability* on the one hand means that a network node can not be brought to halt through a variety of bogus messages. Availability also means, that the communication medium can not be affected in a way that communication fails. The latter can be addressed using spread spectrum techniques (e.g., frequency hopping) mitigating the effects of jamming and noise.

Authentication terms the act of confirming the truth of a message's origin. The recipient of a message has to verify whether or not the message has actually been sent by the pretended originator. This allows to reject bogus messages sent by malicious thirds. More details can be found in Section 5.2.

Data *freshness* denotes the novelty of some data. Checking data freshness of a message helps to recognize and prevent replay attacks. Freshness can be ensured by adding a counter to the message.

The next section introduces a novel broadcast authentication scheme, solely based on symmetric cryptographic primitives. Sections 5.3 to 5.5 specify a security concept for distributed traffic management applications.

5.2 Broadcast authentication

As mentioned above, authentication denotes the act of confirming the truth of a message's origin. There exist different approaches to verify the originator of a message. Public-key cryptography is basically well-suited for authentication. Prior to transmission, the message's hash value is computed and encrypted using the originator's private key. The signature is transmitted along with the message and after receipt, it is decrypted using the originator's public key. If the resulting hash value matches the message's actual hash value, the message's originator corresponds to the generator of the key pair. Latter implication is valid since a private key is only known to the generator of the key pair. However, procedures based on public-key cryptography are computationally expensive and seem to be inappropriate for sensors with low processing power. A more detailed description of public-key cryptography can be found in Section 2.4.3.

In the last years, a variety of broadcast authentication schemes based on symmetric-key cryptography primitives have been proposed. Plenty of them are based on the Time Efficient Stream Loss-tolerant Authentication (TESLA) protocol presented by Perrig, et al [27]. TESLA uses a one-way key chain with delayed disclosure of keys and time synchronization to achieve asymmetry which is crucial for broadcast authentication. However, the delayed disclosure of keys leads to an authentication delay which in turn makes it necessary to buffer received messages. Due to this characteristic, TESLA is prone to denial-of-service attacks.

However, there also exist approaches that allow instantaneous message authentication. One of them is Bins and Balls (BiBa) proposed by Perrig [26]. The principle of BiBa is simple and can be illustrated as follows: n balls are thrown into m bins randomly. The goal is to have at least one bin containing more than one ball. In more detail, the n balls are in turn n different values s_i that are mapped into m bins by hash function $G_{H(msg||cnt)}$. $G_{H(msg||cnt)}$ is an instance in a hash function family G selected by the hash value $H(msg||cnt)$ computed over the concatenation of message msg and a counter cnt. Any k-way collision ($k \geq 2$) of values s_{i_1} to s_{i_k} together with counter cnt form a signature for message msg. If there are no collisions after the mapping of values s_i, the counter gets incremented and the values s_i are mapped into the bins again. This step has to be repeated until at least one k-way collision occurs. The main disadvantage of

BiBa is the high signing costs of at least $n + 1$ hash function computations. Furthermore, BiBa requires time synchronization.

Another scheme that does not require time synchronization is Hash to Obtain Random Subset (HORS) introduced by Reyzin and Reyzin [28]. It is based on the r-subset-resilience property of hash functions. The n values s_i form the private key while the public key is calculated as $v_i = f(s_i), 0 \leq i \leq n - 1$ using one-way function f. To sign message msg, the computed hash value of msg is divided into k parts of length $log_2\, n$. Each part represents the index of one of the private key values s_i, i.e., a subset of at most k values s_i is chosen. This subset forms the message's signature. Since the private key is disclosed more and more by signing messages, the r-subset-resilience property of the hash function is crucial. It states that the probability that the signature of the $(r + 1)^{th}$ message msg_{r+1} is a subset of the set of all disclosed private key values s_i through signing messages msg_1 to msg_r is sufficiently low. The advantages of HORS are the fast signing and verifying of messages. Only 1 and $1 + k$ hash function computations are necessary for signing and verifying messages. Main disadvantage of HORS is the large size of private and public keys.

5.2.1 Scheme specification

The proposed broadcast authentication scheme is based on one-way chains as described in Section 2.4.1.2. There are five parameters allowing to adjust the scheme according to the application's requirements with regard to the *security level*, *communication overhead*, *computational effort* and *storage* (see Table 5.1). Parameter σ_{nbr} specifies the number of one-way chain pools from which the signature has to be chosen. Each pool contains c_{nbr} distinct one-way chains of length c_{len}. The size of each one-way chain element is specified by f_{len}. Parameter s_{max} defines the maximum number of elements a single one-way chain can be moved during signature generation. The movement along a single one-way chain is subsequently called a *shift*.

Parameter	Description
σ_{nbr}	the number of pools, i.e., the signature size
c_{nbr}	the number of one-way chains per pool
c_{len}	the length of the one-way chains
f_{len}	the size of the one-way function's input/output in bits
s_{max}	the maximum shift

Table 5.1: Parameters of the broadcast authentication scheme

Figure 5.1 shows the basic principle of the proposed scheme. To sign a message m, the message's hash value $h(m)$ is used to choose σ_{nbr} one-way chains, one from each pool. If $s_{max} > 1$, the shift for each chosen one-way chain needs to be determined, too. The signature $\sigma = \{v_0, ..., v_{\sigma_{nbr}-1}\}$ of message m is the set of the σ_{nbr} chosen one-way chains shifted by S_i, $0 \leq i \leq \sigma_{nbr}$. To verify a message, the receiver has to compute the hash value $h(m)$ to identify the one-way chains C_i and their respective shift S_i. Verification succeeds only if $C_i = f^{S_i}(v_i)$ holds for each i, $0 \leq i \leq \sigma_{nbr}$. The simplest way to choose the σ_{nbr} one-way chains and their shift is to truncate the hash value after the first $\sigma_{nbr} \cdot (log_2(c_{nbr}) + log_2(s_{max}))$

bits. However, once the one-way chain elements are disclosed, messages may be forged if the first $\sigma_{nbr} \cdot (log_2(c_{nbr}) + log_2(s_{max}))$ bits of their hash value are equal to the corresponding bits of the original hash value. Clearly, the less bits of the hash value are used, the higher the probability of finding such a message.

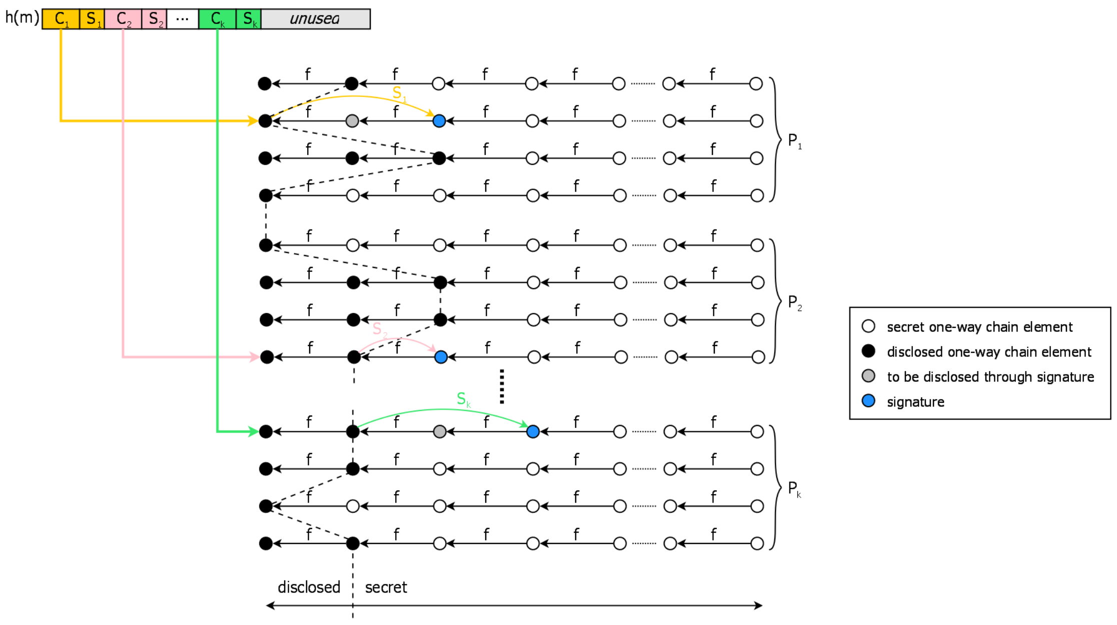

h(m)
C1
S1
C2
S2
...
Ck
Sk
unused
f
S1
S2
Sk
P1
P2
Pk
secret one-way chain element
disclosed one-way chain element
to be disclosed through signature
signature
disclosed
secret

Figure 5.1: Principle of the broadcast authentication scheme

To increase the number of used bits, one can increase parameters σ_{nbr}, c_{nbr} and/or s_{max}. However, the gain of security follows at the expense of communication overhead, computational effort and/or storage requirements. Furthermore, increasing s_{max} does not raise the security level necessarily. The disclosure of one-way chain elements allows to forge messages if $\forall i,\ 0 \leq i \leq \sigma_{nbr} : S'_i \leq S_i$. In the worst case, all shifts S_i are maximal allowing all bits of the hash value used to determine the shift to be different. Thus, the security gain in the worst case is 0 bits. If all shifts are minimal, i.e., in the best case, the security gain is given by $\sigma_{nbr} \cdot log_2(s_{max})$ bits and is maximal. The real security gain through shifting is covered in Section 6.2.1.

In the following, an approach to increase the security level without affecting communication overhead and storage requirements is described. The basic idea is to use $m \cdot n, m > 1$ bits of the hash value h to generate a bit string d of length $n = \sigma_{nbr} \cdot log_2(c_{nbr})$ which in turn determines the signature. Since bit strings of length $m \cdot n$ can not be mapped to bit strings of length n without collisions, messages can be forged with less than $m \cdot n$ bits of the hash value h' equal to the hash value h of the original message. However, through proper construction of the mapping, the probability that d remains unchanged if $k \geq k_0$ arbitrary bits of h are changed is sufficiently small. It can also be shown that at least n bits of h and h' must correspond. While this approach ensures a security level of at least n bits, the security level can be raised to $m \cdot n - k_0$ with a high degree of probability.

Algorithm 5.1 shows the construction of a mapping for $m = 2$. It divides the hash code h into n parts of length 2. Each part $p_i, 0 \leq i < n$ decides to bit d_i of decision bit string[1] d. If $p_i = 11_2$, d_i is set to 1 whereas $p_i = 00_2$ yields $d_i = 0$[2]. The remaining two cases for p_i decide to d_{i+1} and $\neg d_{i+1}$, respectively. An important property of this mapping is that $p'_i = \neg p_i$ yields the decided value $d'_i = \neg d_i$. This property is subsequently referred to as *negation resilience*. Refer to Section 6.2.1 for the importance of negation resilience. Algorithm 5.2 depicts the construction of a negation-resilient mapping for $m = 4$. It can be used to further increase the security level. A detailed analysis of both mappings can be found in Section 6.2.1.

```
   **Input**: hash code $h$, output length $n$
   **Output**: bit string $d$ of length $n$
1  $d_n \leftarrow 0$
2  $i \leftarrow n - 1$
3  for ($i \geq 0$) do
4      $p_i \leftarrow (h_{2i+1}h_{2i})$
5      switch ($p_i$) do
6          case '11': $d_i \leftarrow 1$
7          case '10': $d_i \leftarrow d_{i+1}$
8          case '01': $d_i \leftarrow \neg d_{i+1}$
9          case '00': $d_i \leftarrow 0$
10     endsw
11     $i \leftarrow i - 1$
12 end
13 return ($d_{n-1} \cdots d_0$)
```

Algorithm 5.1: Negation-resilient mapping *decide2*

In the following sections initialization, signature generation and signature verification are described in detail.

5.2.1.1 Initialization

Algorithm 30 illustrates the initialization tasks of source A that wants to broadcast messages. The main task is the generation of the $\sigma_{nbr} \cdot c_{nbr}$ one-way chains of length c_{len}. One-way chains are initialized using a random seed retrieved by random number generator $RAND()$. All initial seeds of the one-way chains need to be unique. After having found a unique initial seed, the one-way chain is computed using one-way function $f(x)$. Note that the begin of the one-way chain, i.e., the initial seed, is indexed with c_{len} while the end is indexed with 0 (in reverse to the definition of one-way chains in Section 2.4.1.2). The current index $current_{p,i}$ of each chain $C_{p,i,}$ is set to 0. It points to the previously disclosed one-way chain element. This means that all one-way chain elements $C_{p,i,j}, j \leq current_{p,i}$ are disclosed. After the one-way chains have been generated, their ends, i.e., $C_{p,i,0}$, are sent to all possible sinks via unicast. This requires the existence of a link key K_{link}^{AB} between source A and sink B.

[1]Bit strings are sequences of zeroes and ones representing arbitrary data, e.g., numbers.

[2]11_2 and 00_2 denote bit strings. Their decimal values are 3 and 0, respectively.

```
Input: hash code $h$, output length $n$
Output: bit string $d$ of length $n$
$d_{n+2} \leftarrow 0$
$d_{n+1} \leftarrow 1$
$d_n \leftarrow 0$
$i \leftarrow n-1$
for $(i \geq 0)$ do
    $p_i \leftarrow (h_{2i+3}h_{2i+2}h_{2i+1}h_{2i})$
    switch $(p_i)$ do
        case '1111':  $d_i \leftarrow 1$
        case '1110':  $d_i \leftarrow 1$
        case '1101':  $d_i \leftarrow 1$
        case '1100':  $d_i \leftarrow d_{i+1}$
        case '1011':  $d_i \leftarrow 1$
        case '1010':  $d_i \leftarrow d_{i+2}$
        case '1001':  $d_i \leftarrow d_{i+3}$
        case '1000':  $d_i \leftarrow 0$
        case '0111':  $d_i \leftarrow 1$
        case '0110':  $d_i \leftarrow \neg d_{i+3}$
        case '0101':  $d_i \leftarrow \neg d_{i+2}$
        case '0100':  $d_i \leftarrow 0$
        case '0011':  $d_i \leftarrow \neg d_{i+1}$
        case '0010':  $d_i \leftarrow 0$
        case '0001':  $d_i \leftarrow 0$
        case '0000':  $d_i \leftarrow 0$
    endsw
    $i \leftarrow i-1$
end
return $(d_{n-1} \cdots d_0)$
```

Algorithm 5.2: Negation-resilient mapping *decide4*

Initialization at sink B simply requires the receipt of the one-way chain elements $C_{p,i,0}$. They are stored as authenticated values $auth_{p,i}$ to verify future broadcast messages. The initialization tasks of sink B are depicted in Algorithm 5.4.

5.2.1.2 Signing broadcast messages

The procedure illustrated in Algorithm 5.5 is used to generate the signature which needs to be sent along with message m. Signature generation is based on the hash value $h(m)$ of message m. The choice of the hash function h severely affects the security level of the proposed scheme. For details about hash functions refer to Section 2.4.1.3. At this point, Hash-based Message Authentication Code (HMAC)-Secure Hash Algorithm Version 1 (SHA-1) is chosen for h. It requires two computations of SHA-1 and generates a hash code of length 160bits. Besides the

```
**Output**: one-way chains $C_{p,i,}$, disclosed elements $current_{p,i}$ of one-way chain $C_{p,i,}$
$p \leftarrow 0$
for $(p < \sigma_{nbr})$ do /* initialize one-way chains */
    $i \leftarrow 0$
    for $(i < c_{nbr})$ do
        $C_{p,i,c_{len}} \leftarrow RAND()$
        $unique \leftarrow True$
        $k \leftarrow 0$
        for $((k \leq p) \wedge unique)$ do /* check if one-way chain $C_{p,i,}$ is unique */
            if $(k < p)$ then $lim \leftarrow c_{nbr}$
            else $lim \leftarrow i$
            $j \leftarrow 0$
            for $((j < lim) \wedge unique)$ do
                if $(C_{k,j,c_{len}-1} = C_{p,i,c_{len}})$ then $unique \leftarrow False$
                $j \leftarrow j + 1$
            end
            $k \leftarrow k + 1$
        end
        if $(unique)$ then
            $j \leftarrow c_{len}$
            for $(j > 0)$ do /* compute one-way chain $C_{p,i,}$ */
                $C_{p,i,j-1} = f(C_{p,i,j})$
                $j \leftarrow j - 1$
            end
            $current_{p,i} \leftarrow 0$
            $i \leftarrow i + 1$
        end
    end
    $p \leftarrow p + 1$
end
send $C_{p,i,0}, 0 \leq p < \sigma_{nbr}, 0 \leq i < c_{nbr}$ via unicast to all possible sinks
/* Note: Unicasts are authenticated and secured using the link key
    $K_{link}^{AB}$.
                                                                      */
```

Algorithm 5.3: Initialization at source A

```
Output: authenticated values $auth_{p,i}$ of one-way chains $C_{p,i,}$
1  receive $C_{p,i,0}, 0 \leq p < \sigma_{nbr}, 0 \leq i < c_{nbr}$ via unicast from source $A$
2  $p \leftarrow 0$
3  for $(p < \sigma_{nbr})$ do /* initialize authenticated values */
4      $i \leftarrow 0$
5      for $(i < c_{nbr})$ do
6          $auth_{p,i} \leftarrow C_{p,i,0}$
7          $i \leftarrow i + 1$
8      end
9      $p \leftarrow p + 1$
10 end
```

Algorithm 5.4: Initialization at sink B

message itself, HMAC-SHA-1 requires an additional input, i.e., a key. The generated hash code depends on both inputs. After computing the hash code using the network key $K_{network}$, *decide2* produces the bit string necessary for generating the signature. The signature σ is the ordered set of $C_{p,i,}$ shifted by S_p, i.e., $C_{p,i,current_{p,i}+S_p}$, for $0 \leq p < \sigma nbr$. Current indexes $current_{p,i}$ of the chosen chains are incremented by S_p. If a one-way chain nears its beginning, i.e., $current_{p,i}$ is within the distance of $2s_{max}$ to c_{len}, it is called exhausted and needs to be renewed. Algorithm 5.6 depicts the task of renewing exhausted one-way chains. Note that exhausted one-way chains are broadcasted after renewal. However, the signature for the renewal message may choose exhausted one-way chains themself. Thus, the renewing actions have to be taken as soon as $current_{p,i}$ is within the distance of $2s_{max}$ to c_{len}. Otherwise, it can not be guaranteed that the required signature for the renewal message can be generated.

5.2.1.3 Verifying broadcast messages

Signature verification, as illustrated in Algorithm 5.7, requires the same initial steps as signature generation. At first the message's hash code h has to be computed using message m and network key $K_{network}$. Subsequently, *decide2* produces the bit string necessary for identifying the one-way chains $C_{p,i,}$ and their respective shifts S_p. The message is accepted only if $f^{S_p}(v_p)$ matches the authenticated value $auth_{p,i}$ for each one-way chain $C_{p,i,}, 0 \leq p < \sigma_{nbr}$ and discarded otherwise. If it is accepted, the authenticated values $auth_{p,i}$ are overwritten by $v_p \in \sigma$.

5.3 Secure communication

As mentioned before, security encompasses confidentiality, integrity, availability, authentication and data freshness. Following sections specify how these key areas can be ensured for the purposes of distributed traffic management applications. For now, the following assumptions are made:

A1 There exists a link key K_{link}^{AB} between each two nodes A and B that want to communicate with each other. K_{link}^{AB} is only known to nodes A and B.

```
Input: message $m$, network key $K_{network}$
Output: signature $\sigma$ as ordered set, exhausted one-way chains $exhaustedChains$ as set
        of 2-tuples
1  $\sigma \leftarrow \emptyset$
2  $exhaustedChains \leftarrow \emptyset$
3  $size \leftarrow log_2(c_{nbr}) + log_2(s_{max})$
4  $h \leftarrow HMAC(K_{network}, m)$
5  $d \leftarrow decide2(h, \sigma_{nbr} \cdot size)$
6  $p \leftarrow 0$
7  for $(p < \sigma_{nbr})$ do
8      $i \leftarrow (d_{k \cdot size + log_2(c_{nbr}) - 1} \cdots d_{k \cdot size})$
9      $s \leftarrow 1$
10     if $(log_2(s_{max}) > 0)$ then $s \leftarrow s + (d_{(k+1) \cdot size - 1} \cdots d_{k \cdot size + log_2(c_{nbr})})$
11     $current_{p,i} \leftarrow current_{p,i} + s$
12     append $C_{p,i,current_{p,i}}$ to $\sigma$
13     if $(current_{p,i} > (c_{len} - 2s_{max}))$ then append $(p, i)$ to $exhaustedChains$
       /* Note: Exhausted one-way chains have to be sent via broadcast
          after renewal. Thus, at least one last signature has to be
          provided by each exhausted one-way chain.                  */
14     $p \leftarrow p + 1$
15 end
16 return $(\sigma, exhaustedChains)$
```
Algorithm 5.5: Signature generation

A2 There exists a network key $K_{network}$ that is known to authorized nodes of the network only, i.e., $K_{network}$ is not known by any eavesdropping attacker.

A3 A security sublayer is deployed beneath the Application Layer (APL) layer. The security sublayer protects APL layer messages before passing them to the Network Layer (NWK) layer and verifies NWK layer messages before passing them to the APL layer, respectively.

A4 Let `HMAC` be some instantiation of HMAC as defined in Section 2.4.1.4.

A5 Let `BC` be some symmetric-key cryptographic block cipher as described in Section 2.4.2. Furthermore, let `Encrypt` denote block cipher `BC` applied to plaintext while `Decrypt` denotes block cipher `BC` applied to ciphertext.

5.3.1 Unicast messages

Figure 5.2 illustrates how unicast messages are protected. Security processing of outgoing unicast messages has to be performed as follows:

1. Obtain link key K_{link}^{AB} and frame counter $frameCounter_{out}^{B}$ associated to K_{link}^{AB}.

2. Construct auxiliary header *AuxHDR*:

```
Input: exhausted one-way chains $exhaustedChains$ as set of 2-tuples
1  initialize renewal message $m$
2  $k \leftarrow 0$
3  for $(k < length(exhaustedChains))$ do /* initialize temporary one-way chains */
4      $p, i \leftarrow exhaustedChains_k$
5      $T_{p,i,c_{len}} \leftarrow RAND()$
6      $j \leftarrow c_{len} - 1$
7      for $(j > 0)$ do /* compute temporary one-way chain $T_{p,i,}$ */
8          $T_{p,i,j-1} = f(T_{p,i,j})$
9          $j \leftarrow j - 1$
10     end
11     $current_{p,i} \leftarrow 0$
12     append $(p, i, T_{p,i,0})$ to $m$
13     $k \leftarrow k + 1$
14 end
15 send renewal message $m$ via broadcast
   /* Note: Since the exhausted one-way chains may be required for m's
      signature, the temporary one-way chains have to be made permanent
      after sending the renewal message.                              */
16 make temporary one-way chains $T_{p,i,}$ permanent
```
Algorithm 5.6: Renewing exhausted one-way chains

a) Set the *Ctrl* field to 0[3].

b) Set the *Frame Counter* field to $frameCounter^B_{out}$.

3. Compute the message authentication code *MAC* of (*AuxHDR* || $m_{unicast}$) using link key K^{AB}_{link}, i.e., *MAC* = HMAC(K^{AB}_{link}, *AuxHDR* || $m_{unicast}$).

4. Encrypt ($m_{unicast}$ || *MAC*) using link key K^{AB}_{link}, i.e., *Encrypted Payload* = `Encrypt`(K^{AB}_{link}, $m_{unicast}$ || *MAC*).

5. Increment and store $frameCounter^B_{out}$.

6. Pass (*AuxHDR* || *Encrypted Payload*) to the network layer.

Using the steps above to protect unicast messages, authentication and integrity are ensured through the Message Authentication Code (MAC). However, authentication and integrity are invalidated if assumption A1 does not hold. Using a frame counter guarantees data freshness allowing to recognize replay attacks. Note that the computation of the message authentication code must include the frame counter. If not, the frame counter may unnoticeably be altered revoking data freshness. Encrypting the message and its authentication code using link key K^{AB}_{link} ensures confidentiality as long as assumption A1 holds.

Incoming unicast messages have to be processed as follows:

[3] reserved for future use

```
Input: message $m$, signature $\sigma = \{v_0, \cdots, v_{\sigma_{nbr}-1}\}$, network key $K_{network}$
Output: authentication result $authenticated$
1  $authenticated \leftarrow True$
2  $size \leftarrow log_2(c_{nbr}) + log_2(s_{max})$
3  $h \leftarrow HMAC(K_{network}, m)$
4  $d \leftarrow decide2(h, \sigma_{nbr} \cdot size)$
5  $p \leftarrow 0$
6  for $((p < \sigma_{nbr}) \wedge authenticated)$ do
7      $chain_p \leftarrow (d_{k \cdot size + log_2(c_{nbr})-1} \cdots d_{k \cdot size})$
8      $s \leftarrow 1$
9      if $(log_2(s_{max}) > 0)$ then $s \leftarrow s + (d_{(k+1) \cdot size-1} \cdots d_{k \cdot size + log_2(c_{nbr})})$
10     if $(auth_{p,chain_p} \neq f^s(v_p))$ then $authenticated \leftarrow False$
11     $p \leftarrow p + 1$
12 end
13 if $(authenticated)$ then
14     $p \leftarrow 0$
15     for $(p < \sigma_{nbr})$ do
16         $auth_{p,chain_p} \leftarrow v_p$
17         $p \leftarrow p + 1$
18     end
19 end
20 return $authenticated$
```

Algorithm 5.7: Signature verification

1. Obtain link key K_{link}^{AB} and frame counter $frameCounter_{in}^{B}$ associated to K_{link}^{AB}.
2. If *AuxHDR.Frame Counter* is smaller than $frameCounter_{in}^{B}$, discard the frame and indicate a failure to the application layer. Otherwise proceed with next step.
3. Decrypt $EncryptedPayload$ to obtain $m_{unicast}$ and *MAC* using link key K_{link}^{AB}, i.e., $(m_{unicast} \,||\, textitMAC) =$ Decrypt(K_{link}^{AB}, *Encrypted Payload*).
4. Compute the message authentication code *MAC′* of (*AuxHDR* $||\, m_{unicast}$) using link key K_{link}^{AB}, i.e., *MAC′* $=$ HMAC(K_{link}^{AB}, *AuxHDR* $||\, m_{unicast}$).
5. If *MAC′* matches *MAC*, proceed with next step. Otherwise discard the frame and indicate a failure to the application layer.
6. Set $frameCounter_{in}^{B}$ to *AuxHDR.Frame Counter* + 1 and store $frameCounter_{in}^{B}$.
7. Pass $m_{unicast}$ to the APL layer.

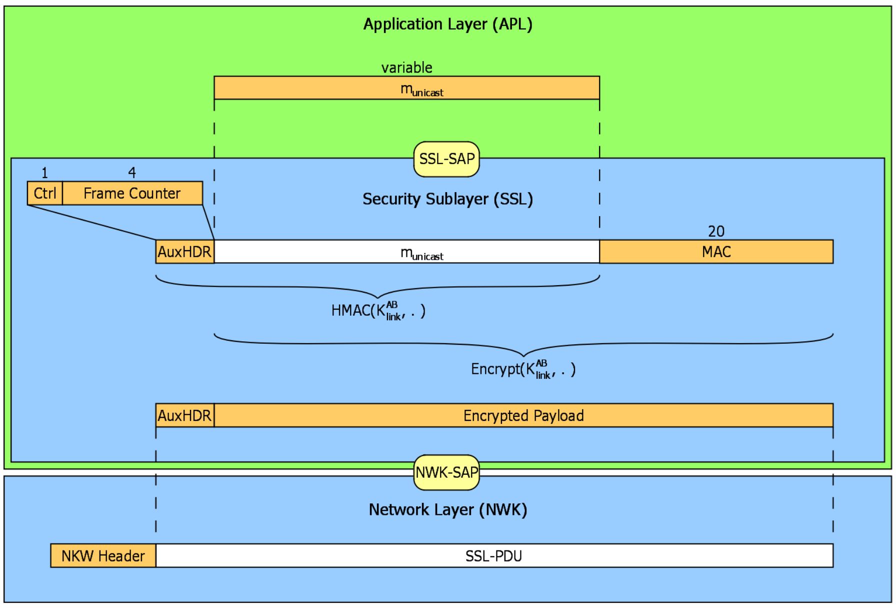


Figure 5.2: Protecting unicast messages

5.3.2 Broadcast messages

Figure 5.3 illustrates how broadcast messages are protected. Security processing of outgoing broadcast messages has to be performed as follows:

1. Obtain network key $K_{network}$ and frame counter $frameCounter^{A}_{network}$ associated to $K_{network}$.

2. Construct auxiliary header *AuxHDR*:

   a) Set the *Ctrl* field to 0[4].

   b) Set the *Frame Counter* field to $frameCounter^{A}_{network}$.

3. Generate signature σ using Algorithm 5.5 with parameters $(\mathit{AuxHDR}\,||\,m_{broadcast})$ and $K_{network}$.

4. Encrypt $(m_{broadcast}\,||\,\sigma)$ using network key $K_{network}$,
   i.e., $\mathit{Encrypted\ Payload} = \mathtt{Encrypt}(K_{network}, m_{broadcast}\,||\,\sigma)$.

5. Increment and store $frameCounter^{A}_{network}$.

6. Pass $(\mathit{AuxHDR}\,||\,\mathit{Encrypted\ Payload})$ to the network layer.

[4] reserved for future use

Security processing for broadcast messages ensures authentication and integrity through signature σ. Using a frame counter guarantees data freshness allowing to recognize replay attacks. Note that the signature generation must include the frame counter. If not, the frame counter may unnoticeably be altered revoking data freshness. As long as assumption A2 holds, confidentiality is ensured by encryption of the message and its signature using network key $K_{network}$.

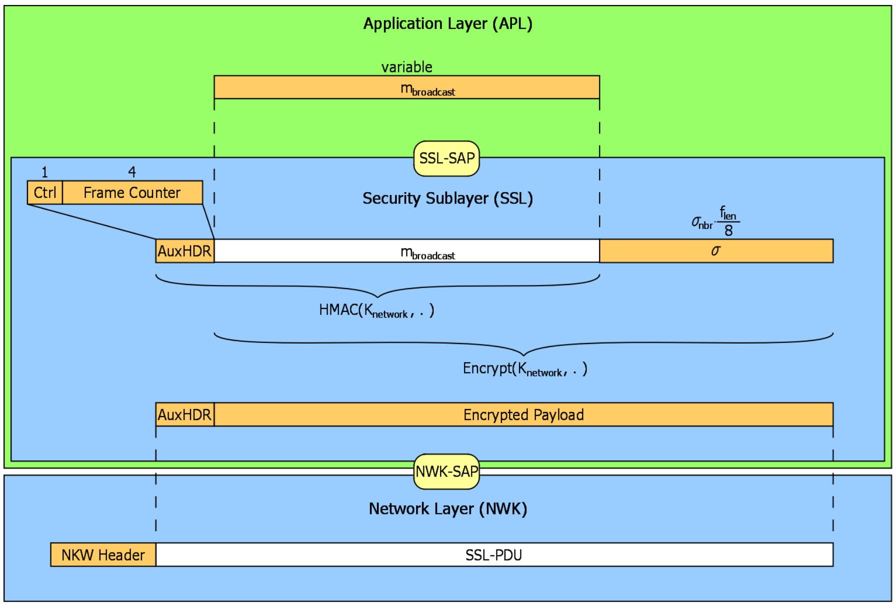


Figure 5.3: Protecting broadcast messages

Incoming broadcast messages have to be processed as follows:

1. Obtain network key $K_{network}$ and frame counter $frameCounter^{B}_{network}$ associated to source node B.
2. If *AuxHDR.Frame Counter* is smaller than $frameCounter^{B}_{network}$, discard the frame and indicate a failure to the application layer. Otherwise proceed with next step.
3. Decrypt $EncryptedPayload$ to obtain $m_{broadcast}$ and σ using network key $K_{network}$, i.e., $(m_{broadcast} \,||\, \sigma) = \text{Decrypt}(K_{network}, \textit{Encrypted Payload})$.
4. Verify signature σ using Algorithm 5.7 with parameters $(\textit{AuxHDR} \,||\, m_{broadcast})$, σ and $K_{network}$.
5. If verification fails, discard the frame and indicate a failure to the application layer. Otherwise, proceed with next step.
6. Overwrite $frameCounter^{B}_{network}$ with *AuxHDR.Frame Counter* + 1.
7. Pass $m_{broadcast}$ to the application layer.

5.4 Key management

As mentioned above, secure communication relies on the existence of pairwise link keys K_{link}^{AB} and a network key $K_{network}$. Until now, it is not clear how these keys can be obtained and updated in a secure way. The following assumption has to be made prior for specifying the procedures of key distribution, key establishment and key update:

B1 There exists a Trust Center (TC), i.e., a distinguished node that is trusted by each other node, responsible for key distribution, key establishment and key update. Furthermore, one can assume that the TC is not compromised. The TC's address has to be pre-configured for each node prior to its deployment.

5.4.1 Key distribution

Let K_{TC}^{A} denote the link key between node A and TC. Link keys K_{TC}^{A} have to be pre-installed to each node A. This ensures the existence of a secure channel between node A and TC that can be taken as a basis for all further steps. Furthermore, the pre-installed keys are used to authorize network nodes. Unauthorized nodes, i.e., potential attackers, are not able to join the network since they lack a mutual key with TC. All other link keys K_{link}^{AB} between two nodes A and B have to be established after deployment as described in Section 5.4.2. Since not every node needs to communicate with each other node, the establishment phase usually will not take too much effort.

The network key $K_{network}$ has to be obtained from TC using the secure channel as described above. For details regarding network keys refer to Section 5.4.3.

5.4.2 Key establishment

To establish a link key K_{link}^{AB} between two nodes A and B, the Symmetric-Key Key Establishment (SKKE) procedure specified by ZigBee [40] is used. Figure 5.4 depicts the message sequence of the SKKE procedure assuming node A wants to establish a link key with node B. The procedure starts with a *key request* sent from node A to TC. A key request message contains the *type* of key requested, i.e., link key or network key, and the partner node's *address* if a link key is requested. The TC responds with a *transport key* message with parameters $Addr_B$ and $initiator = TRUE$ containing a temporary key K_{temp}. It also sends a transport key message with parameters $Addr_A$ and $initiator = FALSE$ containing K_{temp} to B. All further messages are secured using temporary key K_{temp}. A sends a *SKKE request* to B. If B is willing to establish the new link key, it replies with a *SKKE response* containing a positive acknowledge. With following two messages *SKKE-1* and *SKKE-2*, A and B exchange random challenges N_A and N_B. Using random challenges N_A, N_B and partner addresses $Addr_A, Addr_B$, A and B compute K_{link}^{AB}, MAC_1 and MAC_2 as follows:

$$K_{link}^{AB} = \text{HMAC}(K_{temp},\ Addr_A \,||\, Addr_B \,||\, N_A \,||\, N_B)$$

$$MAC_1 = \text{HMAC}(K_{link}^{AB}, 10_2 \,||\, Addr_B \,||\, Addr_A \,||\, N_B \,||\, N_A)$$
$$MAC_2 = \text{HMAC}(K_{link}^{AB}, 11_2 \,||\, Addr_A \,||\, Addr_B \,||\, N_A \,||\, N_B)$$

Message *SKKE-3* contains MAC_2 as computed by A. B checks whether MAC_2 matches its own computation and sends MAC_1 to A (message *SKKE-4*). Analogous, A verifies MAC_1. If all verifications succeeded, K_{link}^{AB} is the new link key established between nodes A and B.

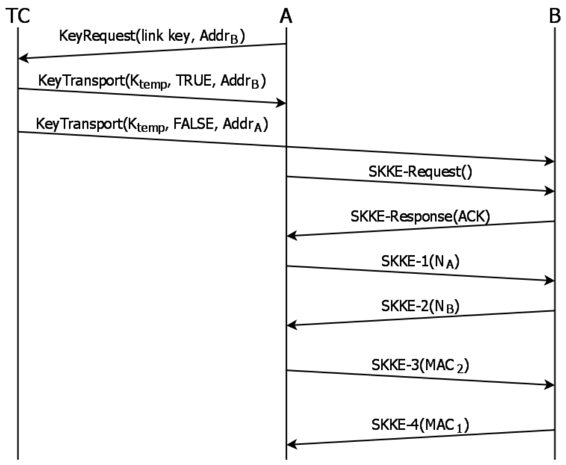

TC
A
B
KeyRequest(link key, Addr_B)
KeyTransport(K_temp, TRUE, Addr_B)
KeyTransport(K_temp, FALSE, Addr_A)
SKKE-Request()
SKKE-Response(ACK)
SKKE-1(N_A)
SKKE-2(N_B)
SKKE-3(MAC_2)
SKKE-4(MAC_1)

Figure 5.4: Symmetric-Key Key Exchange (SKKE) procedure

5.4.3 Key update

In general, a key has to be updated at the very latest, if any frame counter associated to it reaches the maximum value. However, updating keys more frequently reduces the probability that a key can be compromised by an eavesdropping attacker. This can be done by using threshold values for frame counters. Though, threshold values should be chosen with caution since updating keys too frequently affects communication efficiency.

The update of network keys requires that compromised keys can be revoked. Wang *et al.* propose a key revocation scheme that renders compromised keys useless through the use of sophisticated key update techniques [36]. The key revocation scheme is based on the personal key share distribution proposed by Liu *et al.* [21]. During setup, the network manager randomly picks m $2t$-degree masking polyomials, $h_j(x)$, $1 \leq j \leq m$, from $\mathbb{F}_q[x]$ where q is a sufficently large prime number. It then computes $h_j(i)$ for each node A_i where i is the identifier for node A_i. The set $\{h_1(i), h_2(i), \cdots, h_m(i)\}$ is the personal secret of node A_i and has to be sent to A_i using a secure channel. Furthermore, the network manager randomly picks keys $\{K_1, K_2, \cdots, K_m\} \subset \mathbb{F}_q$ and t-degree polynomials $p_1(x), p_2(x), \cdots, p_m(x)$ from $\mathbb{F}_q[x]$ and constructs $q_j(x) = K_j - p_j(x)$.

At the end of each session $j, 1 \leq j \leq m$, the network manager broadcasts a key update message containing the set of revoked nodes $R = \{r_1, r_2, \cdots, r_w\}, w \leq t$ and two $2t$-degree polynomials $P_j(x), Q_j(x)$ defined as follows:

$$P_j(x) = g_j(x)p_j(x) + h_j(x)$$
$$Q_j(x) = g_j(x)q_j(x) + h_j(x)$$

$$\text{where} \quad g_j(x) = (x - r_1)(x - r_2) \cdots (x - r_w)$$

Since the revocation polynomial $g_j(x)$ evaluates to 0 for each revoked node $A_i \in R$, revoked nodes are not able to compute $p_j(i) = \frac{P_j(i) - h_j(i)}{g_j(i)}$ and $q_j(i) = \frac{Q_j(i) - h_j(i)}{g_j(i)}$ and thus $K_j = p_j(i) + q_j(i)$. All other nodes can compute the new network key K_j. Although this key revocation scheme seems to be efficient, it requires to renew the personal secrets after m sessions. Unfortunately this has to be done through sending unicast messages to all nodes.

In the following some modifications of the key revocation scheme proposed in [36] are made to overcome the limitation of sessions. Subsequently, a *round* denotes the period of m sessions. Additionally to the masking polynomials $h_j(x)$ and polynomials $p_j(x)$, the network manager randomly picks m key seeds S_j and some $s \in \mathbb{F}_q$ and constructs $q_j(x) = S_j - p_j(x)$. The new personal secret of node A_i is $\{s, h_1(i), h_2(i), \cdots, h_m(i)\}$. During key update, key seeds S_j are obtained as before and the new network key is computed as $K_j = f(S_j)$ where f is a one-way function as defined in Section 2.4.1.1. After updating the network key for session m, the network manager has to alter masking polynomials $h_j(x)$ before the next round can start. To this end, the network manager randomly picks $(2t + 1)$ nonces $n_k \in \mathbb{F}_q$ and constructs $c_k = s \cdot n_k$ for $0 \leq k \leq 2t$. Values c_k form the coefficients of update polynom $c(x)$. Using $c(x)$, masking polynomials $h_j(x)$ are updated as $h_j(x) = h_j(x) + j \cdot c(x)$. As during setup phase, the network manager then chooses m new key seeds S_j and polynomials $p_j(x)$ at random. To notify nodes A_i about the altered masking polynomials, the network manager broadcasts random nonces n_k. Using n_k, nodes A_i construct coefficients $c_k = s \cdot n_k$ and hence $c(x)$. Having $c(x)$, A_i updates its personal secrets as $h_j(i) = h_j(i) + j \cdot c(i)$.

The modifications described above allow to efficiently reinitialize the key revocation scheme to start a new round. To revoke compromised nodes from the network, an intrusion detection system, such as [12], has to be implemented to detect compromised nodes. However, intrusion detection is beyond the scope of this thesis.

Updating link keys is done quite similar. Let $H_i(x)$ be a t-degree masking polynomial for node A_i. During setup, the network manager randomly picks some $s_i \in \mathbb{F}_q$ and $H_i(x)$ from $\mathbb{F}_q[x]$ for all authorized nodes A_i. Furthermore, $H_i(i)$ is computed and $s_i, H_i(i)$ are sentt to each node A_i via the secure channel. If a link key $K_{TC}^{A_i}$ needs to be updated, the network manager randomly picks t-degree polynomial $p(x)$ from $\mathbb{F}_q[x]$ and key seed $S \in \mathbb{F}_q$ and constructs $q(x) = S - p(x)$. Furthermore, the network manager randomly chooses $(t+1)$ nonces $n_k \in \mathbb{F}_q$ and constructs $c_k = s_i \cdot n_k$ for $0 \leq k \leq t$. Values c_k form the coefficients of update polynom $c(x)$. Using $c(x)$, masking polynomial $H_i(x)$ is updated as $H_i(x) = H_i(x) + c(x)$. Prior updating $H_i(x)$, the network manager sends a key update message to node A_i containing the random nonces n_k and two t-degree polynomials $P(x), Q(x)$ defined as follows:

$$P(x) = p(x) + H_i(x)$$
$$Q(x) = q(x) + H_i(x)$$

Note that $H_i(x)$ is the masking polynomial for node A_i. Node A_i computes $p(i) = P(i) - H_i(i)$ and $q(i) = Q(i) - H_i(i)$ and thus $S = p(i) + q(i)$. Subsequently, A revokes key K_{TC}^{A} by replacing it with $K_{TC}^{A'} = f(S)$ where f is a one-way function. Furthermore, node A_i constructs coefficients $c_k = s_i \cdot n_k$ and obtains $c(x)$. Having $c(x)$, A_i updates its personal secret as $H_i(i) = H_i(i) + c(i)$.

For all other link keys K_{link}^{AB}, either A or B sends a key request to the TC. The procedure for updating link keys K_{link}^{AB} is the same as for the initial establishment of link keys (see Section 5.4.2). For the sake of completeness, note that the network manager usually coincides with the TC.

5.5 Implementation choices

The security concept for distributed traffic management applications has been specified assuming that conditions A1 to A5 and B1 hold. While assumptions A3 to A5 actually are implementation choices that have to be made during application design, assumptions A1 and A2 are not. However, A1 and A2 depend on a proper key management and thus are subsumed by assumption B1. This means that a TC, implementing the key management schemes described above, exists and that it is highly protected against node compromise. Obviously, the Local Control Unit (LCU) seems to be a good choice to take in the role of TC.

CHAPTER 6

Analysis

6.1 Overview

This chapter provides an analysis of the security concept for distributed traffic management applications based on wireless systems specified in Chapter 5. While the following section gives a detailed analysis of the broadcast authentication scheme described in Section 5.2.1, Sections 6.3 and 6.4 evaluate frame protection and key management mechanisms, respectively.

6.2 Broadcast authentication scheme

6.2.1 Security level

As already mentioned, shifting a one-way chain $C_{p,i,}$ by S_p discloses the next S_p elements of that chain. Unfortunately, this fact allows an attacker to intercept a signed message m and forge some message m' as long as $h(m')$ yields the same one-way chains as $h(m)$, i.e., $C'_{p,i,} = C_{p,i,}$, and $S'_p \leq S_p$ for $0 \leq p < \sigma_{nbr}$. To express the real security gain for shifting in bits, it is necessary to know the probability that a message can be forged, if the σ_{nbr} shifts S_p and S'_p differ in x bits in total. This probability is denoted as $P_{forgable}(k, m, x)$ where k is the number of fields, m is the field length in bits and x is the number of bits that differ at arbitrary positions of the $k \cdot m$ bits. Before defining $P_{forgable}(k, m, x)$, let B be a uniformly distributed random variable representing a bit string of length m. Furthermore, let b be a bit string of length m that differs from B by a single but arbitrary bit $i, 0 \leq i < m$. This means that $B_i \neq b_i$ and $B_j = b_j$ for all $j \neq i$. With that, the probability that the value of b is smaller than the value of B, denoted as $P(b < B)$, is given by Equation 6.1. However, what is if b' differs from B by $k > 1$ arbitrary bits $i_1, \cdots, i_k$? Actually, only the most significant bit $i_{max} = max(\{i_1, \cdots, i_k\})$ decides whether or not $b' < B$. With that, $P(b' < B) = P(B_{i_{max}} = 1)$ and by uniformly distribution of B, $P(B_{i_{max}} = 1) = \frac{1}{2}$.

$$P(b < B) = P(B_i = 1) = \frac{1}{2} \tag{6.1}$$

Knowing $P(b < B)$, the definition of $P_{forgable}(k, m, x)$ is given by Equation 6.2. The sum is made over all valid possibilities of distributing x differing bits over k fields with at most m bits per field. Each $k_i, 1 \leq i \leq m$ holds the number of fields that differ in i bits. A distribution is valid, if $\sum i \cdot k_i = x$ and $\sum k_i \leq k$ hold. The fraction states the probability of a certain distribution, while $P(b < B)^\beta$ with $\beta = \sum_{i=1}^{m} k_i$ states the probability that the values of all affected fields have been decreased.

$$P_{forgable}(k, m, x) = \sum_{\substack{k_i \\ 1 \leq i \leq m \\ \sum i \cdot k_i = x \\ \sum k_i \leq k}} \left(\frac{\prod_{i=1}^{m} \left(\binom{k-\alpha(i)}{k_i} \cdot \binom{m}{i}^{k_i} \right)}{\binom{k \cdot m}{x}} \cdot P(b < B)^\beta \right) \tag{6.2}$$

$$\text{where} \quad \alpha(i) = \sum_{j=1}^{i-1} k_j$$

$$\beta = \sum_{i=1}^{m} k_i$$

Now the security gain for shifting can be defined by Equation 6.3. Note that the security gain is probabilistic, i.e., messages with less than $Gain_{shift}(\sigma_{nbr}, s_{max})$ bits of the shift fields matching can be forged with probability $P_{forgable}(\sigma_{nbr}, log_2(s_{max}), x)$, $x > \sigma_{nbr} \cdot log_2(s_{max}) - Gain_{shift}(\sigma_{nbr}, s_{max})$. However, since $P_{forgable}(\sigma_{nbr}, log_2(s_{max}), x) \leq P_{threshold}$, the probability of message forgery is negligible if $P_{threshold}$ has been chosen properly.

$$Gain_{shift}(\sigma_{nbr}, s_{max}) = \sigma_{nbr} \cdot log_2(s_{max}) - \alpha(\sigma_{nbr}, log_2(s_{max}))$$

$$\text{where} \quad \alpha(k, m) = min\Big(\Big\{x \in \{1, \ldots, k \cdot m\} \Big| \; P_{forgable}(k, m, x) \leq P_{threshold}\Big\}\Big) \tag{6.3}$$

Figure 6.1 illustrates the security gain through shifting for $P_{threshold} = 10^{-3}$. Unfortunately, there is no security gain for small signature sizes σ_{nbr} ensuring probability for message forgery to be lower than 10^{-3}.

Equation 6.4 states the security level if the one-way chains are chosen directly from the hash value.

$$secLevel(\sigma_{nbr}, c_{nbr}, s_{max}) = \sigma_{nbr} \cdot log_2(c_{nbr}) + Gain_{shift}(\sigma_{nbr}, s_{max}) \tag{6.4}$$

In the following, negation-resilient mappings generating short decision bit strings are examined in detail. As mentioned in Section 5.2.1, such a mapping generates a decision bit string d of length d_{len} using input bit string h of length $h_{len} = b \cdot d_{len}$ for some $b > 1$. To generate

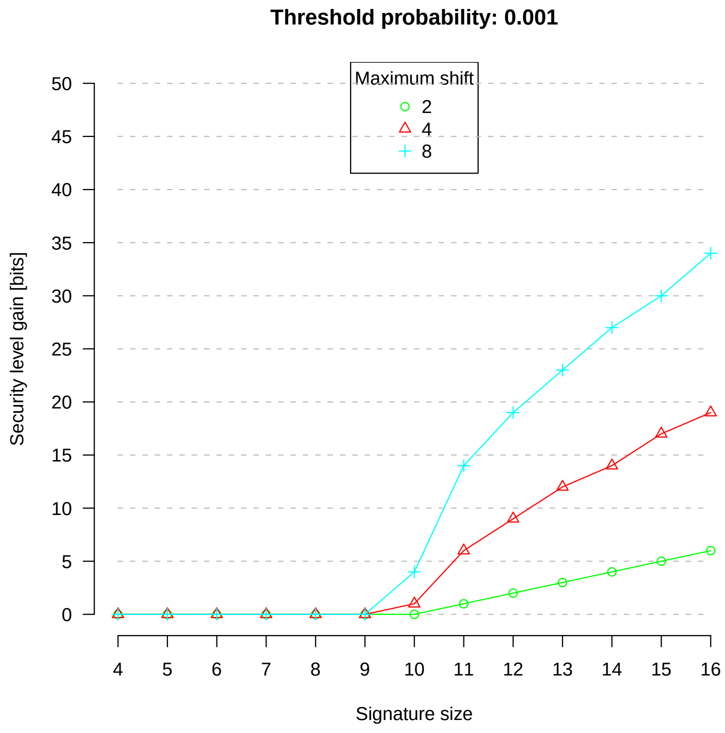


Figure 6.1: Security gain for shifting $Gain_{shift}(\sigma_{nbr}, s_{max})$

the decision bit string, it is crucial that h is divided into d_{len} parts p_i of length b and each part p_i corresponds to bit i of d, i.e., d_i. Negation resilience means that $p'_i = \neg p_i$ yields $d'_i = \neg d_i$. Clearly, negating p_i means changing all b bits of p_i. Assume that $x > (b-1) \cdot d_{len}$ arbitrary bits of input bit string h change their value. Let h' be the resulting bit string. Using the pigeonhole principle, there exists at least one part p_j of h that needs to change the values of all b bits, i.e., $p'_j = \neg p_j$. By negation resilience, $d'_j = \neg d_j$ and hence $d' \neq d$. This means that at most $(b-1) \cdot d_{len}$ bits of the input bit string h may be changed without changing decision bit string d. Or rephrased, at least d_{len} bits of input bit string h must remain unchanged such that d remains unchanged. Hence, negation resilience is vital to guarantee a security level of at least d_{len} bits. Note that both proposed mappings, *decide2* and *decide4*, are negation-resilient by construction (see Algorithms 5.1 and 5.2, respectively).

Let $P_b(d'_j = d_j \,|\, \delta), 0 \leq \delta \leq b$ denote the probability that bit d_j remains unchanged if δ bits of p_j change their value. Obviously, $P_b(d'_j = d_j \,|\, 0) = 1$ and $P_b(d'_j = d_j \,|\, b) = 0$ hold for any negation-resilient mapping. The only constraint for $\delta \neq 0, \delta \neq b$ is $P_b(d'_j = d_j \,|\, \delta) < 1$. However, "good" mappings require to be further constrainted by $P_b(d'_j = d_j \,|\, \delta) < P_b(d'_j = d_j \,|\, \delta - 1)$ for all $\delta \neq 0, \delta \neq b$.

Consider negation-resilient mapping *decide2* defined by Algorithm 5.1. Equation 6.5 states probability $P_2(d'_j = d_j \,|\, 1)$. Let $S_{(x+y)} = \{s \,|\, s$ bit string of length $(x+y)$, x bits set to v, y bits set to $\neg v, x \geq y\}$ be the set of bit strings of length $x + y$ with x bits set to v and y bits set to $\neg v$[1]. $P_{(x+y)}$ denotes the probability that $p_j \in S_{(x+y)}$ whereas $P_{(x+y)\to(x'+y')}$ states the probability that d remains unchanged if $p_j \in S_{(x+y)}$ and $p'_j \in S_{(x'+y')}$. Assuming uniformly distributed $p_i, 0 \leq i < d_{len}$, $P(d_i = 1) = P(d_i = 0) = \frac{1}{2}$ and hence $P_{(2+0)\to(1+1)} = P_{(1+1)\to(2+0)} = \frac{1}{2}$.

$$P_2(d'_j = d_j \mid 1) = \underbrace{P_{(2+0)}}_{\frac{1}{2}} \cdot \underbrace{P_{(2+0)\to(1+1)}}_{\frac{1}{2}} + \underbrace{P_{(1+1)}}_{\frac{1}{2}} \cdot \underbrace{P_{(1+1)\to(2+0)}}_{\frac{1}{2}} = \frac{1}{2} \tag{6.5}$$

Equation 6.6 defines the probability that decision bit string d of length n remains unchanged, if x arbitrary bits of the input bit string h of length $b \cdot n$ are changed. The sum is made over all valid possibilities of distributing x differing bits over n parts with less than b bits per part. Each $n_i, 1 \leq i < b$ holds the number of parts that differ in i bits. A distribution is valid if $\sum i \cdot n_i = x$ and $\sum n_i \leq n$ hold. The fraction states the probability of a certain distribution, while the product $\prod_{i=1}^{b-1} P_b(d'_j = d_j \mid i)^{n_i}$ states the probability that all affected bits of the decision string d remain unchanged.

$$P_{unchanged}(b, n, x) = \sum_{\substack{n_i \\ 1 \leq i < b \\ \sum i \cdot n_i = x \\ \sum n_i \leq n}} \left(\frac{\prod_{i=1}^{b-1} \left(\binom{n-\alpha(i)}{n_i} \cdot \binom{b}{i}^{n_i} \right)}{\binom{b \cdot n}{x}} \cdot \prod_{i=1}^{b-1} P_b(d'_j = d_j \mid i)^{n_i} \right) \tag{6.6}$$

$$\text{where} \quad \alpha(i) = \sum_{j=1}^{i-1} n_j$$

With $P_{unchanged}(b, n, x)$, the security level using a decision bit string generated by *decide2* is defined by Equation 6.7.

$$\begin{aligned} secLevel_2(\sigma_{nbr}, c_{nbr}, s_{max}) = 2 \cdot \sigma_{nbr} \cdot log_2(c_{nbr}) + Gain_{shift}(\sigma_{nbr}, s_{max}) \\ - \alpha(\sigma_{nbr} \cdot log_2(c_{nbr})) \\ \text{where} \quad \alpha(a) = min\Big(\Big\{ x \in \{1, \ldots, a\} \Big| \\ P_{unchanged}(2, a, x) \leq P_{threshold} \Big\} \Big) \end{aligned} \tag{6.7}$$

The security level using a decision bit string generated by *decide4* can analogously be defined. Algorithm 5.2 illustrates the definition of negation-resilient mapping *decide4*. Equations 6.8 to 6.10 state the probabilities for the basic cases $P_4(d'_j = d_j \mid 1)$, $P_4(d'_j = d_j \mid 2)$

[1]For example, $S_{(2+0)} = \{00_2, 11_2\}$ and $S_{(1+1)} = \{01_2, 10_2\}$

and $P_4(d'_j = d_j \,|\, 3)$. Clearly, the additional constraints mentioned above hold, i.e., $P_4(d'_j = d_j \,|\, 3) < P_4(d'_j = d_j \,|\, 2) < P_4(d'_j = d_j \,|\, 1) < P_4(d'_j = d_j \,|\, 0) = 1$.

$$\begin{aligned}
P_4(d'_j = d_j \,|\, 1) = \underbrace{P_{(4+0)}}_{\frac{2}{16}} \cdot \underbrace{P_{(4+0)\to(3+1)}}_{1} + \\
\underbrace{P_{(3+1)}}_{\frac{8}{16}} \cdot \Big(\frac{1}{4} \cdot \underbrace{P_{(3+1)\to(4+0)}}_{1} + \frac{3}{4} \cdot \underbrace{P_{(3+1)\to(2+2)}}_{\frac{1}{2}}\Big) + \\
\underbrace{P_{(2+2)}}_{\frac{6}{16}} \cdot \underbrace{P_{(2+2)\to(3+1)}}_{\frac{1}{2}} = \frac{5}{8}
\end{aligned} \tag{6.8}$$

$$\begin{aligned}
P_4(d'_j = d_j \,|\, 2) = \underbrace{P_{(4+0)}}_{\frac{2}{16}} \cdot \underbrace{P_{(4+0)\to(2+2)}}_{\frac{1}{2}} + \\
\underbrace{P_{(3+1)}}_{\frac{8}{16}} \cdot \underbrace{P_{(3+1)\to(3+1)}}_{\frac{1}{2}} + \\
\underbrace{P_{(2+2)}}_{\frac{6}{16}} \cdot \left(\frac{2}{\binom{4}{2}} \cdot \underbrace{P_{(2+2)\to(4+0)}}_{\frac{1}{2}} + \frac{\binom{4}{2} - 2}{\binom{4}{2}} \cdot \underbrace{P_{(2+2)\to(2+2)}}_{\frac{1}{2}}\right) = \frac{1}{2}
\end{aligned} \tag{6.9}$$

$$\begin{aligned}
P_4(d'_j = d_j \,|\, 3) = \underbrace{P_{(4+0)}}_{\frac{2}{16}} \cdot \underbrace{P_{(4+0)\to(3+1)}}_{0} + \\
\underbrace{P_{(3+1)}}_{\frac{8}{16}} \cdot \left(\frac{1}{\binom{4}{3}} \cdot \underbrace{P_{(3+1)\to(4+0)}}_{0} + \frac{\binom{4}{3} - 1}{\binom{4}{3}} \cdot \underbrace{P_{(3+1)\to(2+2)}}_{\frac{1}{2}}\right) + \\
\underbrace{P_{(2+2)}}_{\frac{6}{16}} \cdot \underbrace{P_{(2+2)\to(3+1)}}_{\frac{1}{2}} = \frac{3}{8}
\end{aligned} \tag{6.10}$$

With $P_{unchanged}(b, n, x)$, the security level using *decide4* is defined by Equation 6.11.

$$\begin{aligned}
secLevel_4(\sigma_{nbr}, c_{nbr}, s_{max}) = 4 \cdot \sigma_{nbr} \cdot log_2(c_{nbr}) + Gain_{shift}(\sigma_{nbr}, s_{max}) \\
- \alpha(\sigma_{nbr} \cdot log_2(c_{nbr})) \\
\text{where} \quad \alpha(a) = min\Big(\Big\{x \in \{1, \ldots, 3a\} \,\Big|\, \\
P_{unchanged}(4, a, x) \leq P_{threshold}\Big\}\Big)
\end{aligned} \tag{6.11}$$

Figure 6.2 depicts the comparison of $secLevel$, $secLevel_2$ and $secLevel_4$. Using decision bit strings raises the security level. When plotted against the signature size, $secLevel_4$ turns out to have the steepest slope. This means that increasing the signature size increases the security level at most when using $secLevel_4$. However, both, $secLevel_2$ and $secLevel_4$, are probabilistic. This means that messages can be forged with probability $P_{unchanged}(b, \sigma_{nbr} \cdot log_2(c_{nbr}), x)$, $x > 2 \cdot \sigma_{nbr} \cdot log_2(c_{nbr}) + Gain_{shift}(\sigma_{nbr}, s_{max}) - secLevel_b(\sigma_{nbr}, c_{nbr}, s_{max})$ matching less than $secLevel_b$ bits of the hash value. However, since $P_{unchanged}(b, \sigma_{nbr} \cdot log_2(c_{nbr}), x) \leq P_{threshold}$, the probability of message forgery is negligible if $P_{threshold}$ has been chosen properly. Figure 6.3 illustrates $secLevel_4$ for different values of c_{nbr} plotted against the signature size.

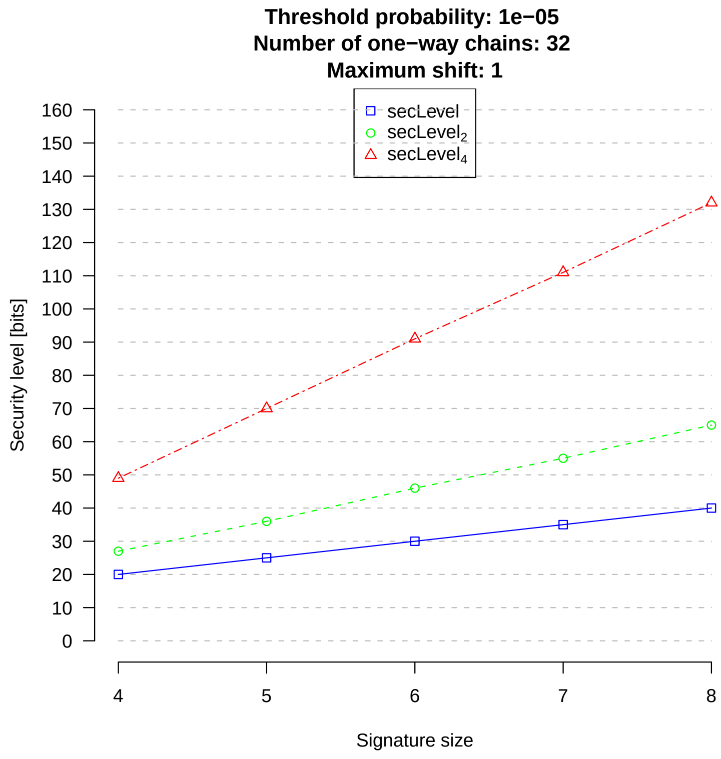


Figure 6.2: Security level comparison

6.2.2 Storage requirements

Storage requirements at sources are different as compared with sinks. While sources need to store whole one-way chains, sinks need to store the most recently disclosed element of each one-way chain, only. Equation 6.12 states the storage requirements for sources. Unfortunately, keeping whole one-way chains in memory leads to nearly prohibitive amount of memory needed. However, there exist more sophisticated approaches allowing to trade off storage against compu-

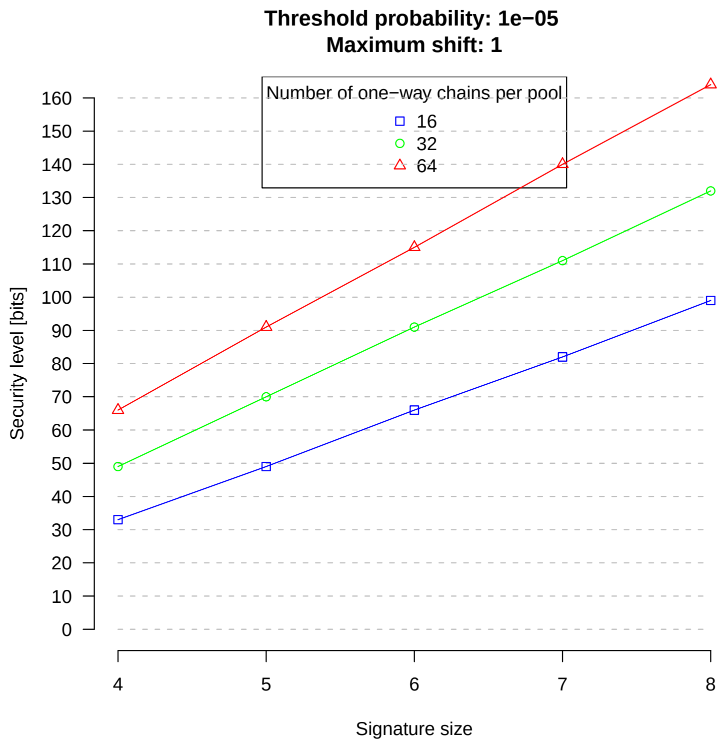


Figure 6.3: Enhanced security level $secLevel_4$

tation complexity. Sella proposed a hash chain traversal with storage requirements $k \cdot \sqrt[k]{n}$ where n is the number of elements of the chain and each element can be reached with at most $k-1$ hash function computations [31]. Furthermore, a special protocol for $k = 2$ with storage requirement l (assuming $n = \frac{l \cdot (l+1)}{2}$, $l > 1$) is presented. In [7], Coppersmith and Jakobsson propose a hash chain traversal with storage complexity $\lceil log_2(n) \rceil + \lceil log_2(log_2(n) + 1) \rceil$ and computation complexity $\lfloor log_2(\sqrt{n}) \rfloor$ for each element. Equation 6.13 states the enhanced storage requirements for sources using the hash chain traversal technique presented in [7].

$$storageSrc(\sigma_{nbr}, c_{nbr}, c_{len}, f_{len}) = \sigma_{nbr} \cdot c_{nbr} \cdot (c_{len} + 1) \cdot \frac{f_{len}}{8} \tag{6.12}$$

$$storageSrc^*(\sigma_{nbr}, c_{nbr}, c_{len}, f_{len}) = \sigma_{nbr} \cdot c_{nbr} \cdot \alpha(c_{len} + 1) \cdot \frac{f_{len}}{8}$$
$$\text{where} \quad \alpha(n) = \lceil log_2(n) \rceil + \lceil log_2(log_2(n) + 1) \rceil \tag{6.13}$$

Figure 6.4 depicts the storage requirements at source A for $\sigma_{nbr} = 4$ and $f_{len} = 80$ bits in KBytes. The enhanced storage requirements using the hash chain traversal proposed in [7] are illustrated in Figure 6.5. For smaller one-way chain lengths c_{len}, $storageSrc^*$ is slightly better than $storageSrc$. However, with increasing c_{len}, the separation between $storageSrc^*$ and $storageSrc$ grows rapidly.

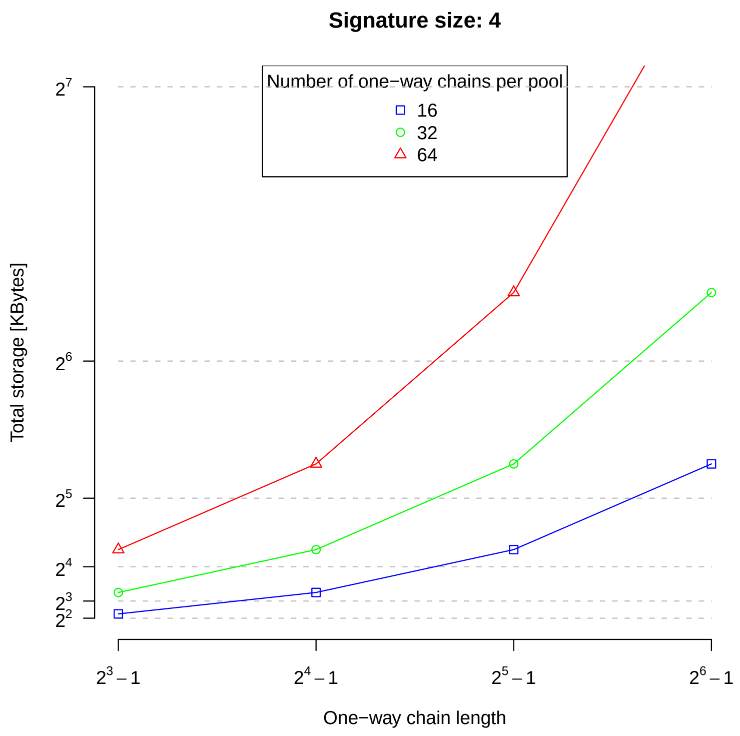


Figure 6.4: Storage requirements at source A for $\sigma_{nbr} = 4$, $f_{len} = 80$ bits

The amount of memory required at sinks is defined by Equation 6.14. It can only be decreased by reducing at least one of its parameters. However, decreasing σ_{nbr}, c_{nbr} or f_{len} affects security. Figure 6.6 shows the storage requirements at sink b for $f_{len} = 80$ bits.

$$storageSnk(\sigma_{nbr}, c_{nbr}, f_{len}) = \sigma_{nbr} \cdot c_{nbr} \cdot \frac{f_{len}}{8} \tag{6.14}$$

6.2.3 Computational effort

The computational effort for signature generation depends, wheter or not all one-way chain elements are kept in memory. If whole one-way chains are stored, signature generation requires the computation of a message's hash value, only. When Hash-based Message Authentication Code (HMAC)-Secure Hash Algorithm Version 1 (SHA-1) is used to generate hash values, the computational effort relating to number of hash function computations is constant. However, if the hash chain traversal proposed in [7] is used to reduce storage requirements, the number

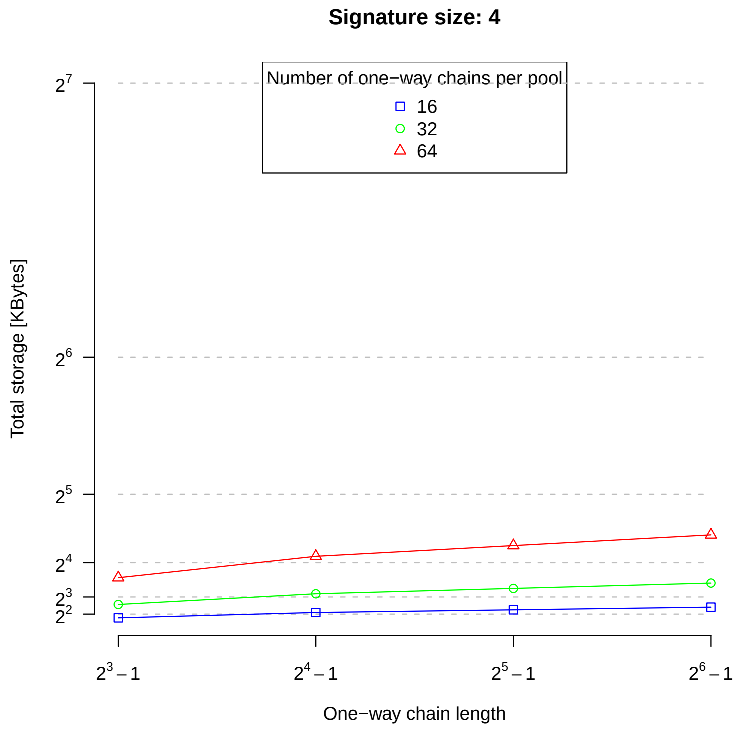


Figure 6.5: Enhanced storage requirements at source A for $\sigma_{nbr} = 4$, $f_{len} = 80$ bits

of hash function computations to retrieve a certain element is given by $\lfloor log_2(\sqrt{n})\rfloor$. Equation 6.15 states the number of hash function computations necessary for signature generation. The constant amount is given by HMAC-SHA-1 which is used to generate the message's hash value.

$$compSign^*(\sigma_{nbr}, c_{len}) = 2 + \sigma_{nbr} \cdot \lfloor log_2(\sqrt{c_{len}+1})\rfloor \tag{6.15}$$

Figure 6.7 depicts the computational effort $compSign^*$ for signature generation using the hash chain traversal proposed in [7].

Signature verification depends on the signature size σ_{nbr} and the maximum shift s_{max}. Equation 6.16 defines the upper bound of hash function computations necessary for signature verification. Again, the constant amount is given by HMAC-SHA-1 which is used to generate the message's hash value. Figure 6.8 shows the upper bound of hash function computations for signature verification.

$$compVerify(\sigma_{nbr}, s_{max}) = 2 + \sigma_{nbr} \cdot s_{max} \tag{6.16}$$

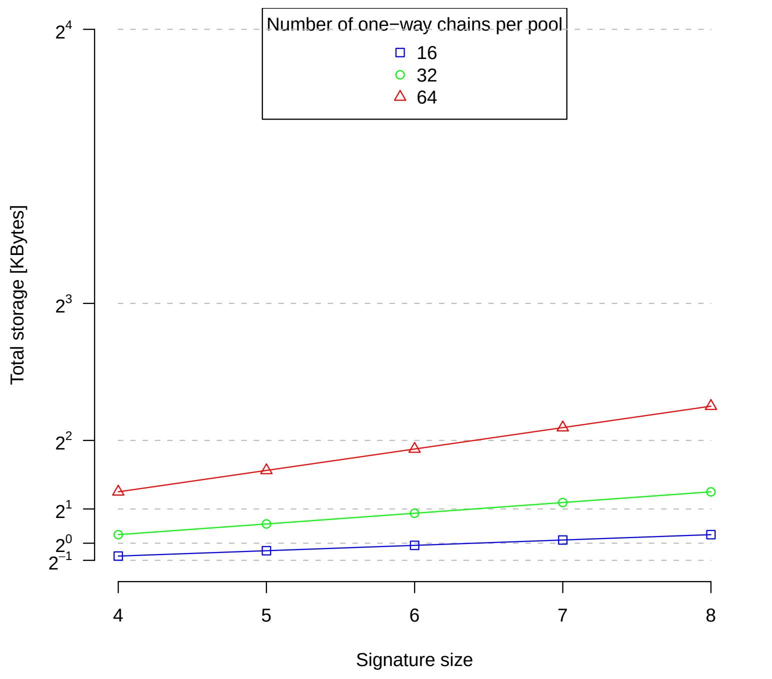


Figure 6.6: Storage requirements at sink B for $f_{len} = 80bits$

6.3 Secure communication

The frame protection mechanisms specified in Section 5.3 mainly rely on the secrecy of all involved keys. Unicast messages are secured through link keys. Link keys are assumed to be only known by the two nodes forming the link. As long as this assumption holds, confidentiality is ensured by encrypting the message using the secret link key. Furthermore, message authentication and integrity are achieved through the use of a Message Authentication Code (MAC). By including a frame counter into the calculation of the MAC, data freshness can also be guaranteed. However, once an attacker gets to know the secret link key, none of these properties hold. In general, there are two possible ways for attackers to obtain secret link keys. Due to the open medium, attackers are able to capture messages that are sent between two nodes. Subsequently, an attacker may disclose the link key based on the captured messages. However, the second attack, node compromise, is a far more dangerous threat. If an attacker is able to compromise an authorized network node, access to all security materials stored at that node is obtained. While the latter attack demands tamper-resistant and/or inaccessible hardware, the first requires proper key management (see Section 6.4).

Protecting broadcast messages is as similar as to protect unicast messages. However, instead of a link key, the network key is used to secure broadcast messages. Another difference is

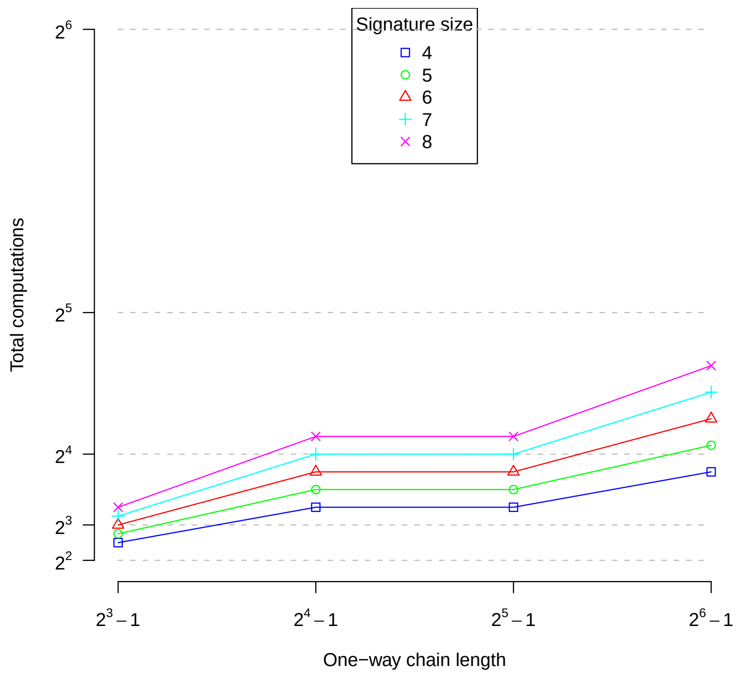


Figure 6.7: Hash function computations for signature generation

that authentication can not be achieved using a symmetric key that is known by more than two entities. Thus, a novel broadcast authentication scheme has been proposed in Section 5.2. A detailed analysis of the broadcast authentication scheme can be found in Section 6.2. As with the MAC, signature σ ensures message authentication and integrity. Confidentiality and data freshness are achieved by encryption using the network key and an integrity protected frame counter, respectively. In contrast to disclosed link keys, disclosed network keys only revoke confidentiality. Authentication, integrity and freshness are still protected by the signature. In case of node compromise, an attacker may forge arbitrary broadcast messages using the node's security materials. However, authentication, integrity and freshness of broadcast messages sent by non-compromised nodes is not affected.

6.4 Key management

Through the use of pre-installed link keys between each node and the Trust Center (TC), unauthorized nodes are not able to join the network. Furthermore, a secure channel between each authorized node and the TC exists right after deployment. The network key and all required security materials can be obtained from the TC using the secure channel.

The Symmetric-Key Key Establishment (SKKE) procedure used to establish (or update) a

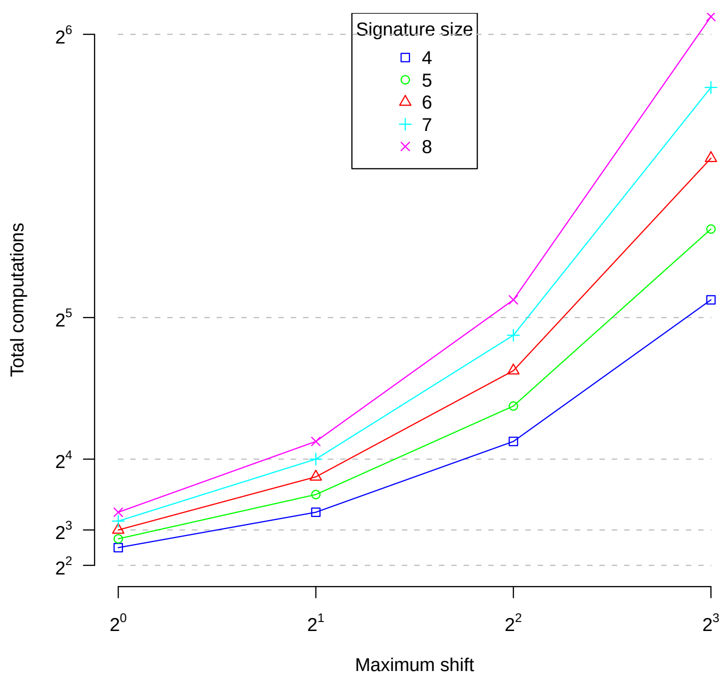


Figure 6.8: Hash function computations for signature verification

link key between two nodes A and B relies on the secrecy of the link keys K_{TC}^{A} shared between A and TC and K_{TC}^{B} between B and TC, respectively. If this assumption holds, the temporary key K_{temp} generated by the TC is only known to nodes A and B (as well as to the TC itself). Hence, the established link key K_{link}^{AB} is only known to nodes A and B (as well as to the TC). However, attackers may try to disclose any of the keys K_{TC}^{A}, K_{TC}^{B} and K_{temp} or key K_{link}^{AB} itself. To impede key disclosure based on captured messages, keys of sufficient length have to be chosen and proper key update mechanisms ensuring forward and backward secrecy need to be employed.

Both key update mechanisms proposed in Section 5.4.3 provide forward and backward secrecy. Consider the key update of link keys $K_{TC}^{A_i}$ shared between nodes A_i and the TC. A_i obtains the initial secret $H_i(i)$ using the secure channel described above. Assuming that the TC acts as network manager, the masking polynomial $H_i(x)$ is only known to the TC. Thus given $P(x)$ and $Q(x)$, $p(i)$ and $q(i)$ can only be computed if $H_i(i)$ is known. The same is true for key seed $S = p(i) + q(i)$. However, given a link key $K_{TC}^{A_i}$, $H_i(i)$ can be derived by inverting one-way function f to obtain key seed S. Having S, $H_i(i)$ can be computed as $H_i(i) = \frac{P(i)+Q(i)-S}{2}$. Although having $H_i(i)$ at a certain point in time, s_i is required to compute the update polynomial $c(x)$ and hence to update $H_i(i)$ after renewing the link key. This means that as long as either $H_i(i)$, s_i or both are not known by an attacker, it is hard to derive future and previous link keys even if multiple link keys are known.

Basically, the key update mechanism for network keys is an extended version of the link key update mechanism. Although it allows multiple sessions per round as well as revocation of compromised nodes, its properties concerning forward and backward secrecy of keys are the same as described above.

Part IV

Conclusion and bibliography

CHAPTER 7

Conclusion and outlook

At the beginning of this thesis, the problem statement and the aims of this thesis were formulated. Briefly, the centralized approach at the field level of modern Traffic Management Systems (TMSs) as well as the lack of a standardized communication protocol for sensors and actuators in the traffic domain have major drawbacks. A decentralized network consisting of autonomously cooperating sensors and actuators may replace the centralized approach. To this end, a generic information model needs to be defined building the basis for a standardized communication protocol. Furhermore, wireless technologies allow the production of flexible TMS facilities while reducing installation costs. Thus, a security concept for distributed Traffic Management (TM)-applications based on wireless communication systems has to be specified.

Subsequently, basic notions were explained providing the basis for this thesis' main objectives. At first an overiew of modern TMSs was given. Furthermore, some of the most popular wireless standards for low-rate and low-power Wireless Personal Area Networks (WPANs) were outlined. Finally, the notion of information modeling was explained and basic security concepts were introduced.

After the basic concepts have been introduced, a generic information model for sensors and actuators in the traffic domain was presented. The information model is structured on the basis of Functional Groups (FGs). Each FG represents a specific field of TMSs. While the first FG, i.e., FG0, comprises attributes and commands which are used for general device configuration and information, FG1 defines attributes and commands for the traffic data acquisition. In more detail, FG1 comprises clusters for vehicle measurement as well as load, traffic and speeding statistics. FG2 specifies clusters for environmental data acquisition including temperature, wind, precipitation and visibility measurements. Finally, FG3 defines attributes and commands that can be used for traffic control through Variable Message Signs (VMSs) and groups of VMSs. Based on the presented information model, a common communication protocol can easily be developed.

In addition to the theoretical specification of the information model, a proof-of-concept implementation of a traffic jam warning system was presented. It was shown how the information model can be used to design distributed TM-applications. Additionally, further use cases were outlined.

Once the information model was specified, a security concept for distributed TM-applications based on wireless communication systems was presented. Unfortunately, the need for security contradicts the requirements of Wireless Sensor and Actuator Networks (WSANs). The main security challenges in WSANs are limited resources, large-scale networks, dynamical network topologies and wireless communication characteristics. Thus, the specification of a proper security concept for WSANs is not trivial.

Broadcast authentication is a crucial part of WSAN security. After existing mechanisms, i.e., Time Efficient Stream Loss-tolerant Authentication (TESLA) [27], Bins and Balls (BiBa) [26] and Hash to Obtain Random Subset (HORS) [28], were outlined, a novel broadcast authentication scheme, solely based on one-way chains, was proposed. It was shown that for the basic scheme, storage requirements and communication overhead, i.e., signature size, are far too high if a "high" security level needs to be achieved. Thus, two extensions of the basic scheme were presented. Both extensions are based on negation-resilient mappings and hence are probabilistic. Probabilistic means that signatures can be forged with some probability even if less bits of the forged hash value coincide with the original hash value. However, it was shown that the security level of the basic scheme is a lower bound for both extensions and that the probability of message forgery is negligible if the threshold probability is chosen properly. It was also shown that the extensions offer a far better security level with same storage requirements and signature sizes.

Furthermore, frame protection mechanisms as well as key management procedures were presented. The security concept relies on the presence of a non-compromised trust center. Each node stores a pre-installed link key with the trust center such that a secure channel between the trust center and each node exists. Based on this secure channel, key establishment and key update mechanisms were defined. The update of keys is a crucial security task. Once a key is compromised it needs to be revoked. Furthermore, forward and backward secrecy need to be ensured, i.e., it must be "hard" to derive previous and future keys based on a compromised key. It was shown that both presented key update mechanisms guarantee forward and backward secrecy. The key update mechanism for network keys further allows to revoke compromised nodes from the network.

Future steps could include the extension of the proposed information model as well as improvements of the security concept with respect to storage requirements, computational effort and communication overhead. Although the information model covers the bigger part of the most important fields of modern TMSs, it is by far not complete. However, it builds a good basis for designing distributed TM-applications.

As already mentioned, the security concept, especially the broadcast authentication scheme, could also be improved. Although storage requirements and signature sizes of the broadcast authentication scheme are reasonable, further enhancements are desirable. Broadcast authentication in WSANs is a interesting research topic and one can assume that other, probably better, schemes will be proposed in the next years. However, the proposed security concept is independent from the specific broadcast authentication mechanism used. Thus, the broadcast authentication scheme could easily be replaced.

Finally, as mentioned at the beginning of this thesis, it would be interesting to replace the the centralized approach at the field level of modern TMSs with a decentralized network of autonomously cooperating sensors and actuators. This requires the proposed standardized communication protocol for sensors and actuators in the traffic domain. However, it is doubtful that such a standardization in real systems will be initiated in the near future.

Bibliography

[1] American Association of State Highway and Transportation Officials (AASHTO), Institute of Transportation Engineers (ITE), and National Electrical Manufacturers Association (NEMA). *National Transportation Communications for ITS Protocol - The NTCIP Guide*, July 2009.

[2] Autobahnen und Schnellstraßen-Finanzierungs-Aktiengesellschaft (ASFINAG). *TLS over IP*, 2004.

[3] P. Baronti, P. Pillai, V.W.C. Chook, S. Chessa, A. Gotta, and Y. Fun Hu. Wireless Sensor Networks: A Survey on the State of the Art and the 802.15.4 and ZigBee Standards. *Computer Communications*, 30(7):1655 – 1695, 2007.

[4] Bundesanstalt für Straßenwesen. *Technische Lieferbedingungen für Streckenstationen*. Bundesministerium für Verkehr, Bau- und Wohnungswesen, 2002.

[5] P.P.S. Chen. The Entity-Relationship Model – Toward a Unified View of Data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976.

[6] X. Chen, K. Makki, K. Yen, and N. Pissinou. Sensor Network Security: A Survey. *Communications Surveys Tutorials*, 11(2):52–73, 2009.

[7] D. Coppersmith and M. Jakobsson. Almost Optimal Hash Sequence Traversal. In Matt Blaze, editor, *Financial Cryptography*, volume 2357 of *Lecture Notes in Computer Science*, pages 102–119. Springer Berlin Heidelberg, 2003.

[8] G. Dimitrakopoulos and P. Demestichas. Intelligent Transportation Systems. *Vehicular Technology Magazine*, 5(1):77–84, March 2010.

[9] L. Figueiredo, I. Jesus, J.A.T. Machado, J.R. Ferreira, and J.L. Martins de Carvalho. Towards the Development of Intelligent Transportation Systems. In *Proceedings of the IEEE International Conference on Intelligent Transportation Systems (ITS)*, pages 1206–1211, August 2001.

[10] N. Gura, A. Patel, A. Wander, H. Eberle, and S.C. Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. *Cryptographic Hardware and Embedded Systems-CHES*, pages 925–943, 2004.

[11] IEEE Computer Society. *IEEE Standard 802.15.4*. IEEE, 2011.

[12] M. Khanafer, M. Guennoun, and H.T. Mouftah. Intrusion Detection System for WSN-Based Intelligent Transportation Systems. In *Proceedings of the IEEE International Conference on Global Telecommunications (GLOBECOM)*, pages 1–6, December 2010.

[13] A.N. Kim, F. Hekland, S. Petersen, and P. Doyle. When HART Goes Wireless: Understanding and Implementing the WirelessHART Standard. In *Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 899–907. IEEE, 2008.

[14] H. Kirschfink, J. Hernandez, and M. Boero. Intelligent Traffic Management Models. *ESIT*, pages 36–45, 2000.

[15] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[16] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication, February 1997. RFC 2104. http://tools.ietf.org/pdf/rfc2104.pdf.

[17] H. Kulovits, C. Stogerer, and W. Kastner. System Architecture for Variable Message Signs. In *Proceedings of the 10th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 2, pages 903–909, September 2005.

[18] N. Kushalnagar, G. Montenegro, and C. Schumacher. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statements, and Goals.

[19] L. Lamport. Password Authentication with Insecure Communication. *Communications of the ACM*, 24(11):770–772, 1981.

[20] Y.T. Lee. Information Modeling: From Design to Implementation. In *Proceedings of the 2nd World Manufacturing Congress (WMC)*, pages 315–321, 1999.

[21] D. Liu, P. Ning, and K. Sun. Efficient Self-Healing Group Key Distribution with Revocation Capability. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)*, pages 231–240, New York, NY, USA, 2003. ACM.

[22] A.J. Menezes, P.C. Van Oorschot, and S.A. Vanstone. *Handbook of applied cryptography*. CRC, 1996.

[23] V. Miller. Use of Elliptic Curves in Cryptography. In *Proceedings of the International Conference on Advances in Cryptology (CRYPTO)*, pages 417–426. Springer, 1986.

[24] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks, September 2007. RFC 4944. http://tools.ietf.org/pdf/rfc4944.pdf.

[25] Object Management Group. *Unified Modeling Language (UML)*, August 2011.

[26] A. Perrig. The BiBa One-Time Signature and Broadcast Authentication Protocol. In *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS)*, pages 28–37, 2001.

[27] A. Perrig, R. Canetti, J.D. Tygar, and D. Song. *The TESLA Broadcast Authentication Protocol*. Department of Engineering and Public Policy, 2005.

[28] L. Reyzin and N. Reyzin. Better than BiBa: Short One-Time Signatures with Fast Signing and Verifying. In Lynn Batten and Jennifer Seberry, editors, *Information Security and Privacy*, volume 2384 of *Lecture Notes in Computer Science*, pages 144–153. Springer Berlin Heidelberg, 2002.

[29] R.L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[30] N. Sastry and D. Wagner. Security Considerations for IEEE 802.15.4 Networks. In *Proceedings of the 3rd ACM Workshop on Wireless Security (WiSe)*, pages 32–42, New York, NY, USA, 2004. ACM.

[31] Y. Sella. On The Computation-Storage Trade-Offs of Hash Chain Traversal. In Rebecca N. Wright, editor, *Financial Cryptography*, volume 2742 of *Lecture Notes in Computer Science*, pages 270–285. Springer Berlin Heidelberg, 2003.

[32] E. Shi and A. Perrig. Designing Secure Sensor Networks. *Wireless Communications*, 11(6):38–43, December 2004.

[33] J. Song, S. Han, A.K. Mok, D. Chen, M. Lucas, and M. Nixon. WirelessHART: Applying Wireless Technology in Real-Time Industrial Process Control. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 377–386. IEEE, 2008.

[34] TZWED. *Dynamisch Route Informatie Paneel Applicatielaag Protocol (DAP)*, 1999.

[35] United States National Institute of Standards and Technology (NIST). Announcing the Advanced Encryption Standard (AES). *Federal Information Processing Standards (FIBS)*, Publication 197, 2001.

[36] Y. Wang, B. Ramamurthy, and X. Zou. KeyRev: An Efficient Key Revocation Scheme for Wireless Sensor Networks. In *Proceedings of the IEEE International Conference on Communications (ICC)*, pages 1260–1265, June 2007.

[37] World Meteorological Organization (WMO). *Manual on Codes – International Codes*, 2011.

[38] X. Zhang, H.M. Heys, and C. Li. Energy Efficiency of Symmetric Key Cryptographic Algorithms in Wireless Sensor Networks. In *Proceedings of the 25th Biennial Symposium on Communications (QBSC)*, pages 168–172, May 2010.

[39] ZigBee Alliance. ZigBee Cluster Library Specification. *ZigBee document 075123r02ZB*, 1, 2008.

[40] ZigBee Alliance. ZigBee Specification. *ZigBee document 053474r17*, 1, 2008.

[41] ZigBee Alliance. ZigBee Home Automation Public Application Profile. *ZigBee document 053520r26*, 1, 2010.

Part V

Appendixes

APPENDIX A

Acronyms

6LoWPAN	IPv6 over Low-Power Wireless Personal Area Network
AES	Advanced Encryption Standard
AO	Application Object
AP	Application Profile
APL	Application Layer
APS	Application Support Layer
BiBa	Bins and Balls
CAP	Contention Access Period
CBC	Cipher Block Chaining
CCM*	Counter with Cipher Block Chaining Message Authentication Code
CFP	Contention Free Period
CSMA-CA	Carrier Sense Multiple Access – Collision Avoidance
DSSS	Direct Sequence Spread Spectrum
ECC	Elliptic Curve Cryptography
ER	Entity-Relationship
FFD	Full Function Device
FG	Functional Group

GTS	Guaranteed Time Slot
HMAC	Hash-based Message Authentication Code
HORS	Hash to Obtain Random Subset
ITMS	Intelligent Traffic Management System
LCU	Local Control Unit
LR-WPAN	Low-Rate Wireless Personal Area Networks
MAC	Medium Access Control Layer
MAC	Message Authentication Code
MD5	Message Digest Algorithm Version 5
MIC	Message Integrity Code
MTU	Maximum Transmission Unit
NTCIP	National Transportation Communications for ITS Protocol
NWK	Network Layer
OMG	Object Management Group
OO	Object-Oriented
PAN	Personal Area Network
PHY	Physical Layer
RFD	Reduced Function Device
SHA-1	Secure Hash Algorithm Version 1
SKKE	Symmetric-Key Key Establishment
SS	Sub-Station
SSP	Security Service Provider
TC	Trust Center
TDMA	Time Division Multiple Access
TESLA	Time Efficient Stream Loss-tolerant Authentication
TLS	“Technische Lieferbedingungen für Streckenstationen”
TM	Traffic Management

TMIC	Traffic Management and Information Center
TMS	Traffic Management System
UML	Unified Modeling Language
VMS	Variable Message Sign
WPAN	Wireless Personal Area Network
WSAN	Wireless Sensor and Actuator Network
ZDO	ZigBee Device Object